GA21-9331-6

File No. S38-01

IBM System/38

IBM System/38
Functional Reference Manual–Volume 1

GA21-9331-6

File No. S38-01

IBM System/38

IBM System/38
Functional Reference Manual–Volume 1

Seventh Edition (September 1985)

This major revision makes obsolete GA21-9331-5. See *About This Manual* for a summary of major changes in this edition.

The information in this publication applies to the IBM System/38 Instruction Set. The information herein is subject to change. These changes will be reported in technical newsletters or in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department 245, Rochester, Minnesota, U.S.A. 55901. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

About This Manual

PURPOSE OF THIS MANUAL

This publication describes the System/38 machine interface instruction set. It describes the functions that can be performed by each instruction and also the necessary information to code each instruction. It provides reference information for the systems engineer and the program support customer engineer.

ORGANIZATION OF THIS MANUAL

The information in this publication is arranged as follows:

- Chapter 1 describes the basic information for coding instructions.
- Chapters 2 through 22 contain detailed descriptions of all the instructions.
- Chapter 23 contains explanations for the possible exceptions that error conditions may signal.
- Chapter 24 contains detailed descriptions of the events that the user can monitor.
- Chapter 25 contains the attributes; specifications; and ODT (object definition table), ODV (ODT directory vector), and OES (ODT entry string) formats for each program object of the machine interface.
- Appendix A provides a summary of all the instructions and an abbreviated format for each instruction.

HOW TO USE THIS MANUAL

Refer to Chapters 2 through 22 to find the information needed to code the various instructions.

Refer to Chapters 23 through 25 to find detailed specifications for the exceptions, events, and program objects.

Refer to Appendix A for a summary of all instructions, which contains the abbreviated description of the instruction and the page number where the detailed description of the instruction can be found.

SUMMARY OF CHANGES

The following new instructions have been added to this manual:

- Cipher (CIPHER)
- Cipher Key (CIPHERKY)
- Create Dump Space (CRTDMPS)
- Destroy Dump Space (DESDMPS)
- Estimate Size of Data Space Index Key Range (ESTDSIKR)
- Insert Dump Data (INSDMPD)
- Materialize Dump Space (MATDMPS)
- Modify Dump Space (MODDMPS)
- Retrieve Dump Data (RETDMPD)
- Scan with Control (SCANWC)
- Terminate Instruction (TERMINST)

Chapter 22 contains descriptions of the new dump space management instructions.

Also, miscellaneous changes have been made throughout this manual.

WHAT YOU SHOULD KNOW

You should read the *IBM System/38 Functional Concepts Manual,* GA21-9330, in its entirety. This manual provides information for the machine interface and its functions.

IF YOU NEED MORE INFORMATION

IBM System/38 Functional Reference Manual—Volume 2, GA21-9800

IBM Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic, SC30-3112

IBM Synchronous Data Link Control General Information Manual, GA27-3093

Abbreviations and Acronyms

ABI	address bus in
ABO	address bus out
ACK	alternating positive acknowledgment
ACR	abandon call and retry
ACTLU	activate logical unit
ACTPU	activate physical unit
ACU	autocall unit
AIMPL	alternate initial microprogram load
AIPL	alternate initial process load
ALU	arithmetic and logic unit
ANSI	American National Standards Institute
APPC	advanced program to program communications
ASCII	American National Standard Code for Information Interchange
B	byte
Bin	binary
BOT	beginning of tape
bpi	bits per inch
bps	bits per second
BSC	binary synchronous communications
BSCT	binary synchronous communications tributary
BSTAT	basic status
CA	channel address
CCITT	The International Telegraph and Telephone Consultative Committee
CD	controller description
Char	character
CPU	central processing unit
CRC	cyclic redundancy check
CRT	cathode-ray tube
CSA	control storage address
CTS	clear to send
DAF	destination address field
DBI	data bus in
DCE	data communications equipment
DEA	data encryption algorithm
DLE	data link escape
DS	data space
DSI	data space index
DSR	data set ready
DSTAT	device status
DTE	data terminal equipment
DTR	data terminal ready

EBCDIC	extended binary-coded decimal interchange code
ENQ	enquiry
EOF	end of file
EOR	end of record
EOT	end of tape
EOV	end of volume
EPA	encapsulated program architecture
ERP	error recovery procedure
ETB	end of text block
ETX	end of text
FBR	feedback record
FIFO	first in, first out
FOB	function operation block
FM	frequency modulation
FMD	function manager data
HDLC	high-level data link control
HDR	header
I/O	input/output
IAR	instruction address register
IC	insert cursor
ID	identification
IDL	instruction definition list
IOC	input/output controller
IOM	input/output manager
IMPL	initial microprogram load
IMPLA	initial microprogram load abbreviated
IPL	initial program load
ITB	intermediate text block
K	1024 bytes
L/D	load/dump
LEAR	lock exclusive allow read
LENR	lock exclusive no read
LIFO	last in, first out
LRC	longitudinal redundancy check
LSRD	lock shared read
LSRO	lock shared read only
LSUP	lock shared update
LU	logical unit
LUD	logical unit description

MB	megabyte
MCR	machine configuration record
MDT	modified data tag
MFM	modified frequency modulation
MISR	machine initialization status record
MPL	multiprogramming level
MRJE	MULTI-LEAVING remote job entry
MSCP	machine services control point
MTAM	MULTI-LEAVING telecommunication access method
NaN	not-a-number
ND	network description
NRL	name resolution list
NRZI	non-return-to-zero (inverted)
ODT	object definition table
ODV	ODT directory vector
OEM	original equipment manufacture
OES	ODT entry string
OMT	object mapping table
ORE	operation request element
OU	operational unit
OU#	operational unit number
PAG	process access group
PASA	process automatic storage area
PCO	process communication object
PCS	process control space
PDEH	process default exception handler
PE	phase encoding
PIN	personal identification number
PP	presentation position
PSSA	process static storage area
PU	physical unit
RCB	record control block
RD	request descriptor
RFT	request for text
RH	request/response header
RI	ring indicator
RIU	request information unit
RNR	receive not ready
ROS	ready-only storage
RPS	rotation position sensor
RTS	request to send
RU	request/response unit
RVI	reverse interrupt
S-PTR	system pointer
SBA	set buffer address
SCB	string control byte
SCS	standard character stream
SDLC	synchronous data link control
SNA	systems network architecture
SOH	start of header
SRCB	sub record control byte
SSCP	system service control point
SSD	source/sink data
SSR	source/sink request
STX	start of text
TH	transmission header
TTD	temporary text delay
UCSB	universal character set buffer
VAT	virtual address table
VLOG	VMC log
VOL	volume
VTOC	volume table of contents
WACK	wait before transmitting
WSC	work station controller
XID	exchange identification

Chapter 1. Introduction

This chapter contains the following:

- Detailed descriptions of the System/38 machine interface instruction fields and the formats of these fields
- A description of the format used in describing each instruction
- A list of the terms in the syntax that define the characteristics of the operands

You should read this chapter in its entirety before attempting to write instructions.

INSTRUCTION FORMAT

This section describes the formats for the three fields in an instruction. The three fields are:

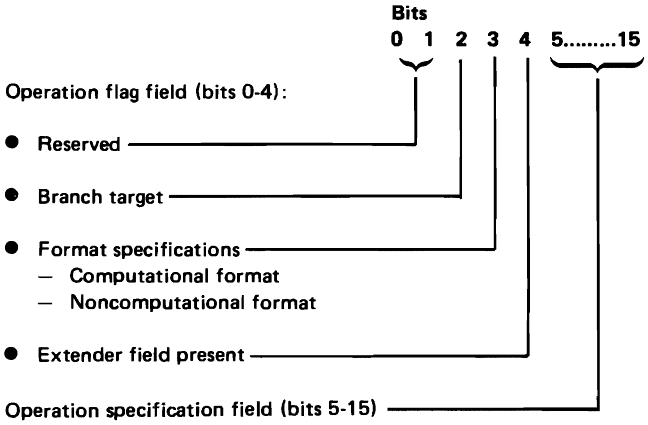
- Operation code
- Operation code extender
- Operand

See the *Functional Concepts Manual* for an explanation of how particular instruction fields are used, the relationships between the fields, and other basic concepts concerning instructions.

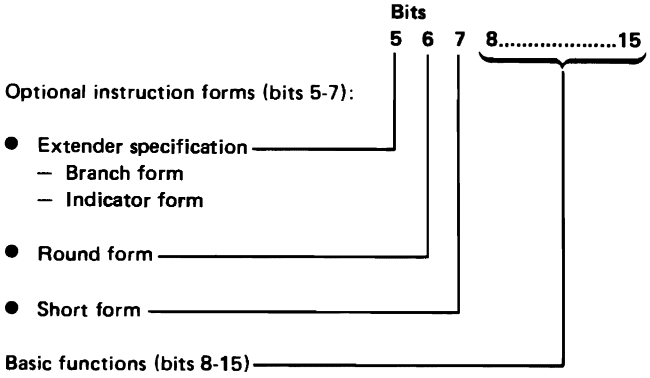
Operation Code Field

The operation code field of an instruction is a 2-byte field that supplies information about the instruction format, the instruction status, and the basic operation to be performed by the instruction.

The format of the operation code field is as follows:



The format of the operation specification field is as follows for the computational format (bit 3 equals 1):



For the noncomputational format (bit 3 equals 0), bits 5-15 define the basic function.

Operation Flag Field (Bits 0-4)

The operation flag field (bits 0-4) specifies the following:

Bits	**Meaning**
0-1	These bits are reserved. They must be 00.
2	Branch target 0 = This instruction is not a branch target. 1 = This instruction is a branch target operand in some branch instructions elsewhere in the instruction stream. This branch target includes branch points defined in the ODT (object definition table), branch targets defined in an IDL (instruction definition list), branch targets assigned to an instruction pointer, immediate instruction numbers used as branch operands, and instructions referenced as entry points. **Note:** The bit encoding of the operation code for each instruction assumes a 0 for this bit.
3	Format specification 0 = Noncomputational–The instruction does not have the format of the computational instructions and does not allow any optional forms. The definition of the operation and the format of the instruction are completely defined by the operation code specification field (bits 5-15). 1 = Computational–The instruction has the computational instruction format. The basic operation is defined in the basic function field (bits 8-15) of the operation code. However, the instruction may allow one or more of the optional instruction forms (indicated by bits 5-7) that define additional information about the operation to be performed, the number of operands, or the format of the instruction.
4	Extender field present 0 = The instruction does not have an operation code extender field. 1 = The instruction has an operation code extender field.

Operation Code Specification Field (Bits 5-15)

The operation code specification field contains information describing the operation to be performed by the instruction and possibly information about the instruction. Its contents depend upon whether this instruction has a computational or a noncomputational format.

- Computational format:

Bits	Meaning
5	Extender specification–The extender field present flag must be on (bit 4 equals 1) before this field has meaning. If bit 4 equals 0, then bit 5 must equal 0. 0 = Indicator form–The format of this instruction is an indicator form of the computational format. An indicator form instruction uses an operation extender field and a character scalar indicator(s) to specify the conditional indicator option(s) and the indicators to be set, respectively. 1 = Branch form–The format of this instruction is a branch form of the computational instruction form. A branch form instruction uses a standard format operation extender field and branch target operand field(s) to specify the conditional branch option(s) and location(s), respectively.
6	Round form 0 = This instruction is not a round form. 1 = The fractional portion of the result of the operation defined for this instruction is to be rounded before being truncated and placed in the field specified by the receiver operand field. A floating-point receiver is not allowed for instructions specified with the round form.
7	Short form 0 = This instruction is not a short form. The format of this instruction is in its normal form with all its required operand fields present. 1 = The format is in the optional short form in which the receiver operand field acts as the first source operand field and is not duplicated as an operand.
8-15	Basic function–These bits indicate the operation to be performed by this instruction (for example, add numeric).

- Noncomputational format:

Bits	Meaning
5-15	Basic function–These bits indicate the operation to be performed by this instruction (for example, create program or set space pointer).

Operation Code Extender Field

The operation code extender field of an instruction is a 2-byte field that further defines the operation to be performed by the instruction and/or the format of the instruction. The extender field is indicated by a 1 in bit 4 of the operation code.

The format and contents of this field are determined by the specific instruction in which it appears. The two types of operation codes extender fields, branch options and indicator options, are described on the following pages.

Branch Options

The branch options operation code extender field contains information needed by instructions that involve conditional branching (comparison instructions and optional branch forms of computational instructions). This field indicates how many branch target operand fields are in the instruction and which of the resulting status conditions relate to each of these target operands.

The following are allowed as branch targets:

- Branch point
- Absolute instruction number (unsigned immediate operand value)
- Relative instruction number (signed immediate operand value)
- Instruction pointer (simple operand that is not an element of an array)

Up to three mutually exclusive status conditions can be specified for a given instruction. The status conditions can be one of the following:

- Ignored
- Associated with a branch target operand field such that:
  - The branch occurs if the condition occurs.
  - The branch occurs if the condition does not occur.

Only one of these three actions can be specified for each condition. Only those conditions meaningful for a particular instruction can have the last two actions specified for them. Conditions that have either of the last two actions specified for them are associated with their branch target operands in left-to-right order.

Branch option operation code extender fields consist of four 4-bit fields. Each of the fields defines one branch condition. The fields must be specified in left-to-right order and correspond to the order of the branch target operands. A field of hex 0 indicates that no branch target is associated with this condition and that no more conditions are defined in any field to the right.

The following codes are valid for branch conditions:

Bit	**Hex**	**Meaning**
0000	0	No branch target, no further fields are checked
0001	1	High, positive, mixed, zero and carry, truncated record, normalized
0010	2	Low, negative, ones, not-zero and no carry, exception ignored, completed record, receiver overrun, denormalized, null compare operand
0011	3	Reserved
0100	4	Equal, zero, zeros, zero and no carry, signaled, exception deferred, dequeued, authorized, source exhausted, infinity
0101	5	Reserved
0110	6	Reserved
0111	7	Unequal, not-zero and carry, escape code encountered, unordered, NaN
1000	8	Reserved
1001	9	Not high, not positive, not mixed, not-zero and carry, not truncated record, not normalized
1010	A	Not low, not negative, not ones, not not-zero and no carry, not completed record, not receiver overrun, not denormalized, not null compare operand
1011	B	Reserved
1100	C	Not equal, not-zero, not zeros, not dequeued, not-zero and no carry, not signaled, not authorized, not source exhausted, not infinity
1101	D	Reserved
1110	E	Reserved
1111	F	Not unequal, not not-zero and carry, not escape code encountered, not unordered, not NaN, not found

The branch options specified for an instruction must be mutually exclusive. The user must not specify a branch to more than one branch target on the same condition; that is, two 4-bit fields cannot specify the same condition.

A not condition refers to any condition other than the one specified. That is, not equal is satisfied with a high, low, or unordered condition. Therefore, the same condition cannot be specified as negative and positive in the same extender (for example, not equal and high cannot be specified together).

If unordered, negation of unordered, equal, negation of equal, zero, or negation of zero conditions are not specified on the instruction and an unordered resultant condition occurs, the floating-point invalid operand exception is signaled. If the exception is masked at the time of detection, the instruction resumes execution and performs the specified branch or indicator processing. For the optional branch forms of computational instructions, this exception occurs because the implicit comparison operation is performed after the assignment of the result of the operation to the receiver operand. Since the receiver is implicitly being used as a source operand for the comparison operation, the receiver is not set with a NaN value as it would be for a masked occurrence of the floating-point invalid operand exception.

The same branch target can be used for multiple conditions. For example, if branch conditions high and equal are specified separately, each of the corresponding branch targets can reference the same instruction. An instruction supporting the high, low, and equal resultant conditions could be accomplished by just a not low condition.

Examples

Hex 4000 means:

- One branch target is present in the instruction.
- Branch to the first branch target operand if an equal condition occurs.
- Otherwise, execute the next sequential instruction.

Hex 1900 means:

- Two branch targets are present in the instruction.
- Branch to the first branch target operand if a high condition occurs.
- Branch to the second branch target operand if a high condition does not occur.

Hex 1210 is not allowed because branch condition 1 (high) is specified twice.

Hex 1A00 is not allowed because condition 1 (high) is also specified as part of condition A (not low).

Indicator Options

The indicator options operation code extender field contains information needed by instructions that allow conditional indicator setting (comparison instructions and optional indicator forms of computational format instructions). The field indicates how many indicator operand fields are in the instruction and which of the resulting status conditions relate to each of these indicator operands.

The preceding discussion of the usage, conditions, ordering, and encoding of branch options also applies to indicator options.

If a condition that is being monitored by the indicator option occurs, the leftmost byte of the associated indicator target is assigned a value of hex F1; otherwise, the leftmost byte of the indicator target is assigned a value of hex F0.

Example

Hex 4000 means:

- One indicator target is present in the instruction.
- Assign a value of hex F1 to the indicator target if the equal condition occurs.
- Assign a value of hex F0 to the indicator target if the equal condition does not occur.

In this example, the indicator form of the operand must be a character or a numeric scalar data object. Only the first byte of the operand is used. This operand must be a simple operand and cannot be a compound subscript operand, a compound substring operand, or a compound based operand.

Instruction Operands

Each instruction requires from zero to four operands. Each operand may consist of one or more fields that contain either a null operand specification, an immediate data value, or a reference to an ODT object. The size of the operand field depends on the version of the program template. If the version number is 0, the size of the operand field is 2 bytes. If the version number is 1, the size of the operand field is 3 bytes.

Null Operands

Certain instructions allow certain operands to be null. In general, a null operand means that some optional function of the instruction is not to be performed or that a default action is to be performed by the instruction.

Immediate Operands

The value of this type of operand is encoded in the instruction operand. Immediate operands may have the following values:

- Signed binary–representing a binary value of negative 4096 to positive 4095.
- Unsigned binary–representing a binary value of 0 to 8191.
- Byte string–representing a single byte value from hex 00 to hex FF.
- Absolute instruction number–representing an instruction number in the range of 1 to 8191.
- Relative instruction number–representing a displacement of an instruction relative to the instruction in which the operand occurs. This operand value may identify an instruction displacement of negative 4096 to positive 4095.

ODT Object References

This type of operand contains a reference (possibly qualified) to an object in the ODT. Operands that are ODT object references may be simple operands or compound operands.

Simple Operands: The value encoded in the operand refers to a specific object defined in the ODT. Simple operands consist of a single 2-byte operand entry.

Compound Operands: A compound operand consists of a primary (2-byte) operand and a series of one to three secondary (2-byte) operands. The primary operand is an ODT reference to a base object while the secondary operands serve as qualifiers to the base object.

A compound operand may have the following uses:

- Subscript references

  An individual element of a data object array, a pointer array, or an instruction definition list may be referenced with a subscript compound operand. The operand consists of a primary reference to the array and a secondary operand to specify the index value to an element of the array.

- Substring references

  A portion of a character scalar data object may be referenced as an instruction operand through a substring compound operand. The operand consists of a primary operand to reference the base string object and secondary references to specify the value of an index (position) and a value for the length of the substring.

  The length secondary operand field can specify whether to allow or not allow for a null substring reference (a length value of zero).

- Explicit base references

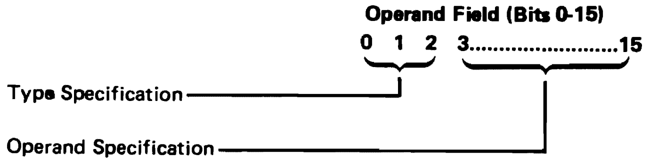
  An instruction operand may specify an explicit override for the base pointer for a based data object or a based addressing object. The operand consists of a primary operand reference to the based object and a secondary operand reference to the pointer on which to base the object for this operand. The override is in effect for the single operand. The displacement implicit in the ODT definition of the primary operand and the addressability contained in the explicit pointer are combined to provide an address for the operand.

The explicit base may be combined with either the subscript or the substring compound operands to provide a based subscript compound operand or a based substring compound operand.

Format of Instruction Operand

The format for an instruction operand depends on the version of the program template.

For program template version number 0, the format for an instruction operand (primary or secondary) field is as follows:



Type Specification Field: The type specification field occupies bits 0-2 of the operand. It indicates whether the operand is an immediate data value, a simple ODT reference, or a compound ODT reference.

The following illustrations show the type specifications allowed for primary operands and secondary operands.

	Primary Operand		**Secondary Operand**			
Operand Function	**Type Bits**	**Operand**	**Number of Secondary Operands**	**1**	**2**	**3**
Simple ODT Reference or Null Operand	000	ODT reference or null	0			
Unsigned Immediate Value	001	Unsigned immediate value	0			
Subscript Compound Operand	010	Array ODT reference	1	Index		
Substring Compound Operand	011	String ODT reference	2	Index	Length	
Explicit Base Compound Operand	100	Based ODT object reference	1	Base pointer		
Signed Immediate Value	101	Signed immediate value	0			
Explicit Based Subscript Compound Operand	110	Based array ODT reference	2	Base pointer	Index	
Explicit Based Substring Compound Operand	111	Based string ODT reference	3	Base pointer	Index	Length

Operand Function	**Type Bits**	**Secondary Operand**
Index	000	ODT reference
	001	Unsigned immediate value
Length (Disallow Null Substring)	000	ODT reference
	001	Unsigned immediate value
Length (Allow Null Substring)	010	ODT reference
	011	Unsigned immediate value
Base Pointer	000	Pointer ODT reference

Operand Specification Field: The operand specification field occupies bits 3-15. It can be an ODT reference or an immediate value. The ODT reference occupies bits 3-15 of the operand field. It contains a binary integer value indicating which ODV (object definition vector) entry in the ODT to use for this operand's definition. This value is an index value for the one-dimensional array ODV, not a byte displacement into the ODT. Thus, a maximum of 8191 ODV objects are addressable in any program. The first ODT reference is 1. If the value of the operand specification field is 0, the operand is null.

The following primary operands are allowed:

- ODT reference (type bits equal 000)

  The operand consists of a simple ODT reference. The value of bits 3-15 of the operand defines an index into the ODT. The range of this value may be from 1 to the size of the ODT (maximum size of 8191).

- Null (type bits equal 000)

  A null operand consists of a 0 value for bits 3-15 of the operand. The null operand is used in several instructions to indicate that a function is not to be performed or that a default action is to occur.

- Unsigned immediate value (type bits equal 001)

  The operand is interpreted as an unsigned immediate data value. Three uses can be made of this form:
  - For numeric operands, an unsigned binary value from 0 to 8191 can be specified in bits 3-15 of the operand.
  - For character (or byte) operands, a single 8-bit value can be specified in bits 8-15 of the operand.
  - For branch target operands, an unsigned binary value of 1 to 8191 can be specified in bits 3-15; that value is interpreted to contain an instruction number. A value of 0 is invalid.

- Array ODT reference (type bits equal 010)

  When the operand type bits are 010, the operand specification (bits 3-15) must be an ODT reference to an array of scalars, an array of pointers, a data pointer that defines an array of scalars, or an instruction definition list. The array indexing operation is performed on this ODT object. If the referenced ODT object is an array of data pointers that define arrays, the indexing operation is performed on the array of data pointers only (this combination is invalid on instructions that require scalar operands). If the ODT object is a data pointer that defines an array, the indexing operation is performed on the defined array.

  A secondary operand is required to specify the array index value.

- String ODT reference (type bits equal 011)

  When the operand type bits are 011, the operand specification (bits 3-15) must be an ODT reference to a data object, data pointer, or a constant data object that has the attributes of a character scalar. The substring operation refers to a portion of this ODT object.

  Two secondary operands are required: one for the index (position) and one for the length of the substring.

  The length secondary operand field can specify whether to allow or not allow for a null substring reference (a length value of zero).

- Based ODT object reference (type bits equal 100)

  When the operand type bits are 100, this operand specification (bits 3-15) must be an ODT reference to a data object with based addressability.

  A secondary operand is required to specify the overriding base pointer.

- Signed immediate value (type bits equal 101)

  The operand is interpreted as a signed immediate data value. Negative values are represented in twos complement form in bits 3-15. Bit 3 is the sign bit. Two uses can be made of this form:
  - For numeric operands, a signed value can be specified in the range of negative 4096 to positive 4095.
  - For branch target operands, a signed binary value of negative 4096 to positive 4095 can be specified, and it is interpreted as a relative instruction number.

- Based array ODT reference (type bits equal 110)

  When the operand type bits are 110, the operand specification (bits 3-15) must be an ODT reference to an array of scalars or an array of pointers with the array based on a space pointer. Explicit basing and array indexing are performed for the operand.

  Two secondary operands are required: one for the base pointer and one for the index value.

- Based string ODT reference (type bits equal 111)

  When the operand type bits are 111, the operand specification must be an ODT reference to either a character scalar data object based on a space pointer or a character scalar data pointer based on a space pointer. Explicit basing and the substring function are performed for the operand.

  Three secondary operands are required: a base pointer, an index value, and a length value.

  The length secondary operand field can specify whether to allow or not allow for a null substring reference (a length value of zero).

The following are allowed as secondary operands. Secondary operands that have the same type value as a primary operand also have the same format for the operand specification field. (Note that secondary operands cannot be compound operands.)

- Index

  A secondary operand representing an index value may be one of the following:
  - An ODT reference to a binary data object (type bits equal 000)
  - An ODT reference to a binary constant data object (type bits equal 000)
  - An unsigned immediate binary value (type bits equal 001)

  An exception is signaled if the value of the index is not greater than 0 or if it is greater than the size of the primary operand (number of bytes for strings, number of elements for arrays, or number of elements for an instruction definition list). The user can suppress the verification of this valid index value for substrings of character strings and elements of arrays by specifying the appropriate constraint attribute when the program is created.

- Length

  A secondary operand representing a length value that disallows a null substring reference (a zero value length) may be one of the following:
  - An ODT reference to a binary data object (type bits equal 000)
  - An ODT reference to a binary constant data object (type bits equal 000)
  - An unsigned immediate binary value (type bits equal 001)

  A secondary operand representing a length value that allows a null substring reference (a zero value length) may be one of the following:
  - An ODT reference to a binary scalar (type bits equal 010)
  - An ODT reference to a binary constant scalar object (type bits equal 010)
  - An unsigned immediate binary value (type bits equal 011)

The operand specification field formats for length secondary operands with type bits equal to 010 and 011 are the same as those for primary operands with type bits equal to 000 and 001, respectively.

An exception is signaled if the length value is not greater than 0 when a null substring reference is not allowed or is not greater than or equal to 0 when a null substring reference is allowed, or if the sum of the index and length values describes a string that cannot be contained in the bytes of the primary operand. The user can suppress verification of this valid index value for substrings of character strings by specifying the appropriate constraint attribute on program creation.

- Base pointer

  If the primary operand is a data object, the base pointer secondary operand must be an ODT reference to a space pointer data object or a space pointer machine object (type bits equal 000).

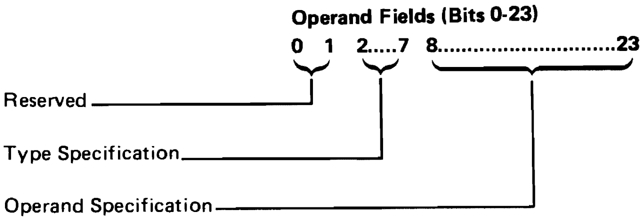
Examples

The following are examples of instruction operands:

Operand Values (Hex)	**Meaning**
0007	A simple ODT reference to ODT object 7
0000	A null operand
2000	An unsigned immediate value of 0 (type bits equal 001)
3FFF	An unsigned immediate value of 8191 (type bits equal 001)
A000	A signed immediate value of 0 (type bits equal 101)
AFFF	A signed immediate value of 4095 (type bits equal 101)
BFFF	A signed immediate value of minus 1 (type bits equal 101)
400A2006	A subscript compound operand reference to array element 6 of the array defined in ODT object 10
E009000800070006	An explicit based substring compound operand that disallows a null substring reference: • ODT object 9 is a based string. • ODT object 8 is a space pointer. • ODT object 7 is a binary data object that provides the index. • ODT object 6 is a binary data object that provides the length.
600900074006	A substring compound operand that allows a null substring reference: • ODT object 9 is a scalar string. • ODT object 7 is a binary scalar index. • ODT object 6 is a binary scalar length.

The format for an instruction operand depends on the version of the program template.

For program template version number 1, the format for an instruction operand (primary or secondary) field is as follows:



Type Specification Field: The type specification field occupies bits 2-7 of the operand. It indicates whether the operand is an immediate data value, a simple ODT reference, or a compound ODT reference.

The following illustrations show the type specifications allowed for primary operands and secondary operands.

	Primary Operand		Secondary Operand			
Operand Function	**Type Bits**	**Operand**	**Number of Secondary Operands**	**1**	**2**	**3**
Simple ODT Reference or Null Operand	000 000	ODT reference or null	0			
Unsigned Immediate Value	000 001	Unsigned immediate value	0			
Subscript Compound Operand	000 010	Array ODT reference	1	Index		
Substring Compound Operand	000 011	String ODT reference	2	Index	Length	
Explicit Base Compound Operand	000 100	Based ODT object reference	1	Base pointer		
Signed Immediate Value	000 101	Signed immediate value	0			
Explicit Based Subscript Compound Operand	000 110	Based array ODT reference	2	Base pointer	Index	
Explicit Based Substring Compound Operand	000 111	Based string ODT reference	3	Base pointer	Index	Length

Operand Function	Type Bits	Secondary Operand
Index	000 000	ODT reference
	000 001	Unsigned immediate value
Length (Disallow Null Substring)	000 000	ODT reference
	000 001	Unsigned immediate value
Length (Allow Null Substring)	000 010	ODT reference
	000 011	Unsigned immediate value
Base Pointer	000 000	Pointer ODT reference

Operand Specification Field: The operand specification field occupies bits 8-23. It can be an ODT reference or an immediate value. The ODT reference occupies bits 8-23 of the operand field. It contains a binary integer value indicating which ODV (object definition vector) entry in the ODT to use for this operand's definition. This value is an index value for the one-dimensional array ODV, not a byte displacement into the ODT. Thus, a maximum of 65 526 ODV objects are addressable in any program. The first ODT reference is 1. If the value of the operand specification field is 0, the operand is null.

The following primary operands are allowed:

- ODT reference (type bits equal 000 000)

  The operand consists of a simple ODT reference. The value of bits 8-23 of the operand defines an index into the ODT. The range of this value may be from 1 to the size of the ODT (maximum size of 65 526).

- Null (type bits equal 000 000)

  A null operand consists of a 0 value for bits 8-23 of the operand. The null operand is used in several instructions to indicate that a function is not to be performed or that a default action is to occur.

- Unsigned immediate value (type bits equal 000 001)

  The operand is interpreted as an unsigned immediate data value. Three uses can be made of this form:
  - For numeric operands, an unsigned binary value from 0 to 8191 can be specified in bits 8-23 of the operand.
  - For character (or byte) operands, a single 8-bit value can be specified in bits 16-23 of the operand.
  - For branch target operands, an unsigned binary value of 1 to 8191 can be specified in bits 8-23; that value is interpreted to contain an instruction number. A value of 0 is invalid.

- Array ODT reference (type bits equal 000 010)

  When the operand type bits are 000 010, the operand specification (bits 8-23) must be an ODT reference to an array of scalars, an array of pointers, a data pointer that defines an array of scalars, or an instruction definition list. The array indexing operation is performed on this ODT object. If the referenced ODT object is an array of data pointers that define arrays, the indexing operation is performed on the array of data pointers only (this combination is invalid on instructions that require scalar operands). If the ODT object is a data pointer that defines an array, the indexing operation is performed on the defined array.

  A secondary operand is required to specify the array index value.

- String ODT reference (type bits equal 000 011)

  When the operand type bits are 000 011, the operand specification (bits 8-23) must be an ODT reference to a data object, data pointer, or a constant data object that has the attributes of a character scalar. The substring operation refers to a portion of this ODT object.

  Two secondary operands are required: one for the index (position) and one for the length of the substring.

  The length secondary operand field can specify whether to allow or not allow for a null substring reference (a length value of zero).

- Based ODT object reference (type bits equal 000 100)

  When the operand type bits are 000 100, this operand specification (bits 8-23) must be an ODT reference to a data object with based addressability.

  A secondary operand is required to specify the overriding base pointer.

- Signed immediate value (type bits equal 000 101)

  The operand is interpreted as a signed immediate data value. Negative values are represented in twos complement form in bits 8-23. Bit 8 is the sign bit. Two uses can be made of this form:
  - For numeric operands, a signed value can be specified in the range of negative 4096 to positive 4095.
  - For branch target operands, a signed binary value of negative 4096 to positive 4095 can be specified, and it is interpreted as a relative instruction number.

- Based array ODT reference (type bits equal 000 110)

  When the operand type bits are 000 110, the operand specification (bits 8-23) must be an ODT reference to an array of scalars or an array of pointers with the array based on a space pointer. Explicit basing and array indexing are performed for the operand.

  Two secondary operands are required: one for the base pointer and one for the index value.

- Based string ODT reference (type bits equal 000 111)

  When the operand type bits are 000 111, the operand specification must be an ODT reference to either a character scalar data object based on a space pointer or a character scalar data pointer based on a space pointer. Explicit basing and the substring function are performed for the operand.

  Three secondary operands are required: a base pointer, an index value, and a length value.

  The length secondary operand field can specify whether to allow or not allow for a null substring reference (a length value of zero).

The following are allowed as secondary operands. Secondary operands that have the same type value as a primary operand also have the same format for the operand specification field. (Note that secondary operands cannot be compound operands.)

- Index

  A secondary operand representing an index value may be one of the following:
  - An ODT reference to a binary scalar (type bits equal 000 000)
  - An ODT reference to a binary data pointer that defines a binary scalar (type bits equal 000 000)
  - An ODT reference to a binary constant scalar object (type bits equal 000 000)
  - An unsigned immediate binary value (type bits equal 000 001)

  An exception is signaled if the value of the index is not greater than 0 or if it is greater than the size of the primary operand (number of bytes for strings, number of elements for arrays, or number of elements for an instruction definition list). The user can suppress the verification of this valid index value for substrings of character strings and elements of arrays by specifying the appropriate constraint attribute when the program is created.

- Length

  A secondary operand representing a length value that disallows a null substring reference (a length value of zero) may be one of the following:
  - An ODT reference to a binary scalar (type bits equal 000 000)
  - An ODT reference to a binary data pointer that defines a binary scalar (type bits equal 000 000)
  - An ODT reference to a binary constant scalar object (type bits equal 000 000)
  - An unsigned immediate binary value (type bits equal 000 001)

  A secondary operand representing a length value that allows a null substring reference (a length value of zero) may be one of the following:
  - An ODT reference to a binary scalar (type bits equal 000 010)
  - An ODT reference to a binary constant scalar object (type bits equal 000 010)
  - An unsigned immediate binary value (type bits equal 000 011)

The operand specification field formats for length secondary operands with type bits equal to 000 010 and 000 011 are the same as those for primary operands with type bits equal to 000 000 and 000 001, respectively.

An exception is signaled if the value of the length is not greater than 0 when a null substring reference is not allowed or is not greater than or equal to 0 when a null substring reference is allowed, or if the index plus the value of the length is greater than the number of bytes in the primary operand. The user can suppress verification of this valid index value for substrings of character strings by specifying the appropriate constraint attribute on program creation.

- Base pointer

  If the primary operand is a data object, the base pointer secondary operand must be an ODT reference to a space pointer data object or a space pointer machine object (type bits equal 000 000).

Examples

The following are examples of instruction operands:

Operand Values (Hex)	**Meaning**
000007	A simple ODT reference to ODT object 7
000000	A null operand
010000	An unsigned immediate value of 0 (type bits equal 000 001)
011FFF	An unsigned immediate value of 8191 (type bits equal 000 001)
050000	A signed immediate value of 0 (type bits equal 000 101)
050FFF	A signed immediate value of 4095 (type bits equal 000 101)
05FFFF	A signed immediate value of minus 1 (type bits equal 000 101)
02000A010006	A subscript compound operand reference to array element 6 of the array defined in ODT object 10 (type bits for the primary operand equal 000 010 and type bits for the secondary operand equal 000 001)
070009000008000007000006	An explicit based substring compound operand that disallows a null substring reference: • ODT object 9 is a based string. • ODT object 8 is a space pointer. • ODT object 7 is a binary data object that provides the index. • ODT object 6 is a binary data object that provides the length.
030009000007020006	A substring compound operand that allows a null substring reference: • ODT object 9 is a character scalar string. • ODT object 7 is a binary scalar index. • ODT object 6 is a binary scalar length.

INSTRUCTION FORMAT CONVENTIONS USED IN THIS MANUAL

The user of this manual must be aware that not every instruction uses every field described in this section. Only the information pertaining to the fields that are used by an instruction is provided for each instruction.

In this manual, each instruction is formatted with the instruction name followed by its base mnemonic. Following this is the operation code (op code) in hexadecimal and the number of operands with their general meaning.

Example:

ADD NUMERIC (ADDN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1043	Sum	Addend 1	Addend 2

This information is followed by the operands and their syntax. See *Definition of the Operand Syntax* later in this chapter for a detailed discussion of the syntax of instruction operands.

Example:

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms: The mnemonics and bit encodings for the optional instruction operation codes are given along with a brief description of the options.

The optional forms are short form, round form, branch form, and indicator form. For a more detailed description of these forms see *Operation Code Field* earlier in this chapter.

Extender: A brief description of the extender options is given.

Description: A detailed description and a functional definition of the instruction is given.

Authorization Required: A list of the object authorization required for each of the operands in the instruction or for any objects subsequently referenced by the instruction is given.

Lock Enforcement: Describes the specification of the lock states that are to be enforced during execution of the instruction.

The following states of enforcement can be specified for an instruction:

- Enforcement for materialization

  Access to a system object is allowed if no other process is holding a locked exclusive no read (LENR) lock on the object. In general, this rule applies to instructions that access an object for materialization and retrieval.

- Enforcement for modification

  Access to a system object is allowed if no other process is holding a locked exclusive no read (LENR) or locked exclusive allow read (LEAR) lock. In general, this rule applies to instructions that modify or alter the contents of a system object.

- Enforcement of object control

  Access is prohibited if another process is holding any lock on the system object. In general, this rule applies to instructions that destroy or rename a system object.

Resultant Conditions: These are the conditions that can be set at the end of the standard operation in order to perform a conditional branch or set a conditional indicator.

Events

The *Events* sections contain a list of events and the corresponding event numbers (in hexadecimal form) that can be caused by the instruction.

A detailed description of the events is in Chapter 24.

Exceptions

The *Exceptions* sections contain a list of exceptions that can be caused by the instruction. (The detailed description of exceptions is in Chapter 23.) Exceptions related to specific operands are indicated for each exception by the Xs under the heading operand. An entry under the word, Other, indicates that the exception applies to the instruction but not to a particular operand.

DEFINITION OF THE OPERAND SYNTAX

Syntax consists of the allowable choices for each instruction operand. The following are the common terms used in the syntax and the meanings of those terms:

- *Numeric*: Numeric attribute of binary, packed decimal, zoned-decimal, or floating-point

- *Character*: Character attribute

- *Scalar*:
  - Scalar data object that is not an array (see Note 1)
  - Constant scalar object
  - Immediate operand (signed or unsigned)
  - Element of an array of scalars (see Notes 1 and 2)
  - Substring of a character scalar or a character scalar constant data object (see Notes 1 and 3)

- *Data Pointer Defined Scalar*:
  - A scalar defined by a data pointer
  - Substring of a character scalar defined by a data pointer (see Notes 1 and 3)

- *Pointer*:
  - Pointer data object that is not an array (see Note 1)
  - Element of an array of pointers (see Notes 1 and 2)
  - Space pointer machine object

- *Array*: An array of scalars or an array of pointers (see Note 1)

- *Variable Scalar*: Same as scalar except constant scalar objects and immediate operand values are excluded.

- *Data Pointer*: A pointer data object that is to be used as a data pointer.
  - If the operand is a source operand, the pointer storage form must contain a data pointer when the instruction is executed.
  - If the operand is a receiver operand, a data pointer is constructed by the instruction in the specified area regardless of its current contents (see Note 4).

- *Space Pointer*: A space pointer data object or a space pointer machine object.

- *Space Pointer Data Object*: A pointer data object that is to be used as a space pointer.
  - If the operand is a source operand, the pointer storage form must contain a space pointer when the instruction is executed.
  - If the operand is a receiver operand, a space pointer is constructed by the instruction in the specified area regardless of its current contents (see Note 4).

- *System Pointer*: A pointer data object that is to be used as a system pointer.
  - If the operand is a source operand, the specified area must contain a system pointer when the instruction is executed.
  - If the operand is a receiver operand, a system pointer is constructed by the instruction in the specified area regardless of its current contents (see Note 4).

- *Relative Instruction Number*: Signed immediate operand.

- *Instruction Number*: Unsigned immediate operand.

- *Instruction Pointer*: A pointer data object that is to be used as an instruction pointer.
  - If the operand is a source operand, the specified area must contain an instruction pointer when the instruction is executed.
  - If the operand is a receiver operand, an instruction pointer is constructed by the instruction in the specified area regardless of its current contents (see Notes 4 and 5).

- *Instruction Definition List Element*: An entry in an instruction definition list that can be used as a branch target. A compound subscript operand form must always be used (see Note 5).

Notes:

1. An instruction operand in which the primary operand is a scalar or a pointer may also have an operand form in which an explicit base pointer is specified.

   See *ODT Object References* earlier in this chapter for more information on compound operands.

2. A compound subscript operand may be used to select a specific element from an array of scalars or from an array of pointers.

   See *ODT Object References* earlier in this chapter for more information on compound operands.

3. A compound substring operand may be used to define a substring of a character scalar, or a character constant scalar object.

   A compound substring operand that disallows a null substring reference (a length value of zero) may, unless precluded by the particular instruction, be specified for any operand syntactically defined as allowing a character scalar. A compound substring operand that allows a null substring reference may be specified for an operand syntactically defined as allowing a character scalar only if the instruction specifies that it is allowed. Whether a compound substring operand does or does not allow a null substring reference is controlled through the specification of the length secondary operand field.

   See *ODT Object References* earlier in this chapter for more information on compound operands.

4. A compound subscript operand form may be used to select an element from an array of pointers to act as the operand for an instruction.

   See *ODT Object References* earlier in this chapter for more information on compound operands.

5. Compound subscript forms are not allowed on branch target operands that are used for conditional branching. Selection of elements of instruction pointer arrays and elements of instruction definition lists may, however, be referenced for branch operands by the Branch instruction.

Alternate choices of operand types and the allowable variations within each choice are indicated in the syntax descriptions as shown in the following example.

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Instruction number, branch point or instruction pointer.

Operand 1 must be variable scalar. Operands 1 and 2 must be numeric. Operand 3 can be an instruction number, branch point or instruction pointer.

When a length is specified in the syntax for the operand, character scalar operands must be at least the size specified. Any excess beyond that required by the instruction is ignored.

Scalar operands that are operated on by instructions requiring 1-byte operands, such as pad values or indicator operands, can be greater than 1 byte in length; however, only the first byte of the character string is used. The remaining bytes are ignored by the instruction.

Chapter 2. Computation and Branching Instructions

This chapter describes all the instructions used for computation and branching. These instructions are arranged in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

ADD LOGICAL CHARACTER (ADDLC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1023	Sum	Addend 1	Addend 2

Operand 1: Character variable scalar (fixed-length).

Operand 2: Character scalar (fixed-length).

Operand 3: Character scalar (fixed-length).

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ADDLCS	1123	Short
ADDLCI	1823	Indicator
ADDLCIS	1923	Indicator, Short
ADDLCB	1C23	Branch
ADDLCBS	1D23	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The unsigned binary value of the addend 1 operand is added to the unsigned binary value of the addend 2 operand and the result is placed in the sum operand.

Operands 1, 2, and 3 must be the same length; otherwise, the Create Program instruction signals an invalid length exception. The length can be a maximum of 256 bytes.

The addition operation is performed according to the rules of algebra. The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a byte value of hex 00.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: The logical sum of the character scalar operands is zero with no carry out of the leftmost bit position, not-zero with no carry, zero with carry, or not-zero with carry.

Events

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

ADD NUMERIC (ADDN)

Op Code (Hex)	**Operand 1**	**Operand 2**	**Operand 3**
1043	Sum	Addend 1	Addend 2

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	**Op Code (Hex)**	**Form Type**
ADDNS	1143	Short
ADDNR	1243	Round
ADDNSR	1343	Short, Round
ADDNI	1843	Indicator
ADDNIS	1943	Indicator, Short
ADDNIR	1A43	Indicator, Round
ADDNISR	1B43	Indicator, Short, Round
ADDNB	1C43	Branch
ADDNBS	1D43	Branch, Short
ADDNBR	1E43	Branch, Round
ADDNBSR	1F43	Branch, Short, Round

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The signed numeric value of the addend 1 operand is added to the numeric value of the addend 2 operand, and the result is placed in the sum operand.

All operands must be numeric with any implicit conversions occurring according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual.*

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

For a decimal operation, alignment of the assumed decimal point takes place by padding with 0's on the right end of the addend with lesser precision.

Floating-point addition uses exponent comparison and significand addition. Alignment of the binary point is performed, if necessary, by shifting the significand of the value with the smaller exponent to the right. The exponent is increased by one for each binary digit shifted until the two exponents agree.

The operation uses the lengths and the precision of the source and receiver operands to calculate accurate results.

The addition operation is performed according to the rules of algebra.

The result of the operation is copied into the sum operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the sum, aligned at the assumed decimal point of the sum operand, or both before being copied. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations outlined in the *Functional Concepts Manual.* If nonzero digits are truncated on the left end of the resultant value, a size exception is signaled.

For the optional round form of the instruction, specification of a floating-point receiver operand is invalid.

For fixed-point operations, if nonzero digits are truncated off the left end of the resultant value, a size exception is signaled.

For floating-point operations involving a fixed-point receiver field, if nonzero digits would be truncated off the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point sum, if the exponent of the resultant value is either too large or too small to be represented in the sum field, the floating-point overflow and floating-point underflow exceptions are signaled, respectively.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: Positive, negative, or zero–The algebraic value of the numeric scalar sum operand is positive, negative, or zero. Unordered–The value assigned a floating-point sum operand is NaN.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operands			Other
	1	2	3	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
03 Decimal point alignment		X	X	
06 Floating-point overflow	X			
07 Floating-point underflow	X			
09 Floating-point invalid operand		X	X	X
0A Size	X			
0C Invalid floating-point conversion	X			
0D Floating-point inexact result	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

AND (AND)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1093	Receiver	Source 1	Source 2

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ANDS	1193	Short
ANDI	1893	Indicator
ANDIS	1993	Indicator, Short
ANDB	1C93	Branch
ANDBS	1D93	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Boolean AND operation is performed on the string values in the source operands. The resulting string is placed in the receiver operand. The operands must be character strings that are interpreted as bit strings.

The length of the operation is equal to the length of the longer of the two source operands. The shorter of the two operands is logically padded on the right with hex 00 values. This assigns hex 00 values to the results for those bytes that correspond to the excess bytes of the longer operand.

The bit values of the result are determined as follows:

Source 1 Bit	**Source 2 Bit**	**Result Bit**
1	1	1
0	1	0
1	0	0
0	0	0

The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a byte value of hex 00.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1, 2, and 3. The effect of specifying a null substring reference for either or both of the source operands is that the result is all zero and instruction's resultant condition is zero. When a null substring reference is specified for the receiver, a result is not set and the instruction's resultant condition is Zero regardless of the values of the source operands.

Resultant Conditions: Zero—The bit value for the bits of the scalar receiver operand is either all zero or a null substring reference is specified for the receiver. Not zero—The bit value for the bits of the scalar receiver operand is not all zero.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

BRANCH (B)

Op Code (Hex)	Operand 1
1011	Branch Target

Operand 1: Instruction number, relative instruction number, branch point, instruction pointer, or instruction definition list element.

Description: Control is unconditionally transferred to the instruction indicated in the branch target operand. The instruction number indicated by the branch target operand must be within the instruction stream containing the branch instruction.

The branch target may be an element of an array of instruction pointers or an element of an instruction definition list. The specific element can be identified by using a compound subscript operand.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
09 Invalid branch target operand	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
2C Program Execution		
04 Branch target invalid	X	

CIPHER (CIPHER)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10EF	Receiver	Controls	Source

Operand 1: Space pointer.

Operand 2: Character(32) variable scalar.

Operand 3: Space pointer.

Description: The cipher operation specified in the controls (operand 2) is performed on the string value addressed by the source (operand 3). The result is placed into the string addressed by the receiver (operand 1).

The first character of the source and receiver strings is addressed by their respective operand pointers. The data length field of the controls operand specifies the length of the input source data. The length of data returned in the receiver is determined from the length of the source. When the data padding option specifies no, the length of data returned in the receiver is equal to the length of the source. When the data padding option specifies yes, the length of data returned in the receiver is not equal to the length of the source and is returned in the data length field of the controls operand. Refer to the discussion of the data padding option for details on the amount of data returned in this case.

The controls operand must be a character variable scalar which specifies information to be used to control the cipher operation. It must be at least 32 bytes long and have the following format:

• Controls operand	Char(32)
– Function identifier	Char(2)
Must be a hex 0001	
– Data length	Char(2)
– Options	Char(1)
— Encrypt or decrypt	Bit 0
0 = Encrypt	
1 = Decrypt	
— Use cipher block chaining	Bit 1
0 = No	
1 = Yes	
— Data padding	Bit 2
0 = No	
1 = Yes	
— Reserved (must be 0)	Bit 3-7
– Cryptographic key	Char(8)
— Key type	Char(1)
Hex 00 = Unencrypted	
Hex 01 = Encrypted with master key	
– Initial chaining value	Char(8)
– Pad character	Char(1)
– Reserved	Char(9)

The function identifier specifies the function number for the cipher operation. It must specify a hexadecimal value of 0001. Any other value causes a template value invalid exception to be signaled.

Hex 0001 Function identifier value of 1 specifies that the ANSI (American National Standard Institute) DEA (data encryption algorithm) is to be used in conjunction with the cryptographic key for an encryption or decryption operation.

The data length field specifies the length of the data addressed by the source operand. The data length value must be nonzero, and when the data padding and use cipher block chaining options specify no, a multiple of 8 bytes. An incorrect data length value results in the signaling of the template value invalid exception. When the data padding option specifies yes, the length of the data placed into the receiver is returned in this field.

The encrypt or decrypt option specifies whether an encryption or decryption operation is to be performed.

The cipher block chaining option specifies whether or not cipher block chaining is to be used during the cipher operation.

When the cipher block chaining option specifies yes for an encryption operation, the first block of data from the source operand is exclusive ORed with the initial chaining value and then encrypted. For subsequent blocks of data, the prior block of encrypted data from the receiver operand is exclusive ORed with the current data block from the source operand and the result is encrypted.

When the cipher block chaining option specifies yes for a decryption operation, the blocks of data from the source operand, starting with the last and then moving from right to left, are decrypted and then exclusive ORed with the prior block of encrypted data from the source operand to return the original data. When the first block of data in the source operand is encountered, it is decrypted and then exclusive ORed with the initial chaining value to return the original data.

The data padding option specifies whether data padding is to be used during the cipher operation. When the data padding option specifies no, padding is not performed and the data length value must be a multiple of 8 bytes. When the data padding option specifies yes, padding is performed. In this case, the length of the data returned in the receiver is different from the source length and is returned in the data length field for both encrypt and decrypt operations.

When the data padding option specifies yes for an encryption operation, the data from the source operand is padded out to the next multiple of 8 bytes; for example, a source length of 20 is padded to 24, 32 is padded to 40, and so forth. When the source length is not a multiple of eight, the final block of source data is padded with zero to six repetitions of the pad character until the block length is 7 bytes in length. The eighth byte is then filled with a 1-byte binary counter containing the number of pad characters used (a value from zero to six) and the block is encrypted. When the source length is a multiple of eight, the final block of source data is encrypted as is and padding produces an extra block of data which is the encryption of the value formed from seven repetitions of the pad character followed by a 1-byte binary value of seven. In this case, the receiver is set with this extra eight byte block of encrypted data even though the source length was a multiple of eight.

When the data padding option specifies yes for a decryption operation, the final block of data is decrypted and the last byte of data, which contains the pad character count (a value from zero to seven), is removed and used to determine the number of additional pad characters to remove from the data. The specified number of pad characters are then removed from the source data prior to placing the remaining decrypted data in the receiver operand.

The cryptographic key field specifies the key to be used for the cipher operation. The cryptographic key may be provided in either an unencrypted or encrypted form through control of the key type.

The key type field specifies whether the cryptographic key is being supplied in an unencrypted or encrypted form. The field must contain a valid key type (one defined in the template). Any other value causes a template value invalid exception to be signaled.

The unencrypted key type specifies the cryptographic key is to be used as is to encrypt or decrypt the source operand. This value is not valid when the use cipher block chaining or data padding options are yes.

The encrypted under master key key type specifies the cryptographic key is to be decrypted using the master key prior to encrypting or decrypting the source operand.

The initial chaining value field specifies the 8-byte value to be used in conjunction with cipher block chaining when the cipher block chaining option specifies yes. In this case, the initial chaining value must not be binary zero or the template value invalid exception is signaled. When the cipher block option specifies no, this field is ignored. Refer to the description of the cipher block chaining option for details on how this value is used in the cipher operation.

The pad character field specifies the value to be used as a pad character when the data padding option specifies yes. When the data padding option specifies no, this field is ignored.

Specific Properties of ANSI DEA

The encrypt or decrypt operation is performed iteratively upon 8-byte blocks of the source operand. Each block is encrypted/decrypted using DEA and the information specified in the controls and the resulting value is placed into the receiver at the same relative location as that from which the source data was accessed from the source operand. The process is repeated until the data in the source is exhausted.

Encryption and decryption use the same key, but in a different key schedule according to the algorithm's rules.

Valid results are produced for the case of the receiver and source operands being coincident with one another. The source data is accessed first, then the result is stored in the receiver.

Partial overlap between the source and receiver operands may produce invalid results.

Events

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
0F Master key not defined or invalid		X		
10 Damage Encountered				
44 Partial system object damage				X
1C Machine Dependent Exception				
03 Machine storage limit exceeded				X
08 Requested function not valid				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2E Resource Control Limit				
02 Process storage limit exceeded				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		

CIPHER KEY (CIPHERKY)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10FF	Receiver	Controls	Source

Operand 1: Character(8) variable scalar.

Operand 2: Character(24) variable scalar.

Operand 3: Character(8) scalar.

Description: The cipher key operations specified in the controls (operand 2) are performed on a source key value either accessed from the source (operand 3) or generated by the machine and the result is placed into the string addressed by the receiver (operand 1).

The source and receiver strings must be at least 8 bytes in length. Any excess bytes are ignored.

The controls operand must be a character variable scalar which specifies information to be used to control the cipher key operation. It must be at least 24 bytes long and has the following format:

- Controls Operand Char(64)
  - Source operation Char(1)
    Hex 00 = No decrypt
    Hex 01 = Decrypt using master key
    Hex 02 = Decrypt using template key
    Hex 03 = Generate random key
    Hex 04 = Decrypt using master key variant 1
    Hex 05 = Decrypt using master key variant 2
    Hex 06 = Decrypt using master key variant 3
  - Receiver operation Char(1)
    Hex 00 = Encrypt using master key
    Hex 01 = Encrypt using master key variant 1
    Hex 02 = Encrypt using master key variant 2
    Hex 03 = Encrypt using master key variant 3
    Hex 04 = Encrypt using template key
    Hex 05 = Verify master key
    Hex 06 = Generate PIN
    Hex 07 = Verify PIN
    Hex 08 = Translate PIN
    Hex 09 = No encrypt
  - Template key type Char(1)
    Hex 00 = Encrypted using master key variant 1
    Hex 01 = Encrypted using master key variant 2
    Hex 02 = Use variant 1 of template key
    Hex 03 = Use variant 2 of template key
    Hex 04 = Use variant 3 of template key
  - Template key Char(8)
  - PIN operation parameters Char(42)
    PIN validation key Char(8)
    PIN decimalization key Char(8)
    PIN protection key Char(8)
    PIN pad character Char(1)
    PIN check length Char(1)
    PIN offset data Char(8)
    Encrypted PIN Char(8)
  - Reserved Char(3)

The source operation specifies how the source key is to be accessed for the cipher key operation to be performed. It must specify a valid source operation (one defined in the template). Any other value causes a template value invalid exception to be signaled.

The no decrypt source operation specifies that the source key is to be accessed directly from the source operand with no decryption.

The decrypt using master key source operation specifies that the source key is to be accessed from the source operand and decrypted using the master key held internally in the machine.

The decrypt using template key source operation specifies that the source key is to be accessed from the source operand and decrypted using the template key specified in the controls operand.

The generate random key source operation specifies that the source key is to be produced by the machine rather than being accessed from the source operand. In this case, the machine generates a random key as the source key for the cipher key operation to be performed. If the receiver operation is encrypt, using master key variant 1, 2, or 3, an 8-byte value must be specified in the source operation to be used as a seed for the random key generator. If the receiver operation is encrypt using master key, values contained in the system are used to generate the random key.

The decrypt using master key variant 1 source operation specifies that the source key is to be accessed from the source operand and decrypted using variant 1 of the master key held internally by the machine.

The decrypt using master key variant 2 source operation specifies that the source key is to be accessed from the source operand and decrypted using variant 2 of the master key held internally by the machine.

The decrypt using master key variant 3 source operation specifies that the source key is to be accessed from the source operand and decrypted using variant 3 of the master key held internally by the machine.

The receiver operation specifies how the receiver key is to be set for the cipher key operation to be performed. It must specify a valid receiver operation (one defined in the template). Any other value causes a template value invalid exception to be signaled.

The encrypt using master key receiver operation specifies that the receiver key is to be set by encrypting the processed source key using the master key held internally in the machine. This operation is not valid when the source operation specifies decrypt using master key or decrypt using template key when template key is encrypted under master key variant 1. Any invalid combination will result in a template value invalid exception being signaled.

The encrypt using master key variant 1 receiver operation specifies that the receiver key is to be set by encrypting the processed source key using variant 1 of the master key held internally in the machine. This operation is not valid when the source operation specifies decrypt using master key or decrypt using template key. Any invalid combination will result in a template value invalid exception being signaled.

The encrypt using master key variant 2 receiver operation specifies that the receiver key is to be set by encrypting the processed source key using variant 2 of the master key held internally in the machine. This operation is not valid when the source operation specifies decrypt using master key or decrypt using template key. Any invalid combination will result in a template value invalid exception being signaled.

The encrypt using master key variant 3 receiver operation specifies that the receiver key is to be set by encrypting the processed source key using variant 3 of the master key held internally in the machine. This operation is not valid when the source operation specifies decrypt using master key or decrypt using template key. Any invalid combination will result in a template value invalid exception being signaled.

The encrypt using template key receiver operation specifies that the receiver key is to be set by encrypting the processed source key using the template key specified in the controls operand. This operation is not valid when the source operation specifies no decrypt, decrypt using template key, or generate random key. The template key type must be encrypted under master key variant 2. Any invalid combination will result in a template value exception being signaled.

The verify master key receiver operation returns the 4-byte verification code for the host master key in the receiver operand. The source operand is not used.

The generate PIN (personal identification number) receiver operation returns a plain text intermediate PIN generated from the data supplied in the PIN operation parameters and source operand. The PIN validation key and PIN decimalization data must be specified in the PIN operations parameters. The validation data must be specified in the source operand and the source operation must be no decrypt. Any invalid combination will result in a template value invalid exception being signaled.

The verify PIN receiver operation verifies the encrypted PIN parameter using the data supplied in the PIN operation parameters and source operand and returns a hex F02 for a valid PIN and a hex F1 for an invalid PIN. All PIN operation parameters must be specified. The validation data must be specified in the source operand and the source operand must be no decrypt. Any invalid combination will result in a template value invalid exception being signaled.

The translate PIN receiver operation translated the encrypted PIN specified in the source operand using the PIN validation key PIN parameter as the input PIN protection key and the PIN protection key PIN parameter as the output PIN protection key. The PIN encrypted under the output PIN protection key is returned. The PIN validation key and PIN protection key PIN parameters must be specified. The encrypted PIN to be translated must be specified in the source operand and the source operation must be no decrypt. Any invalid combination will result in a template value invalid exception being signaled.

The no encrypt receiver operation specifies that the receiver key is to be set without performing an encryption operation. This option is only valid when the source operation is generate random key. Any invalid combination will result in a template value invalid exception being signaled.

The template key type specifies the variant of the master key that was used to encrypt the template key. It must specify a valid template key type (one defined in the template). Any other value causes a template value invalid exception to be signaled. This field is ignored when the template key is not to be used.

The encrypted under master key variant 1 template key type specifies that the template key is encrypted under variant 1 of the master key and must be decrypted prior to use. This type is not valid when the source operation specifies decrypt using template key. Any invalid combination will result in a template value invalid exception being signaled.

The encrypted under master key variant 2 template key type specifies that the template key is encrypted under variant 2 of the master key and must be decrypted prior to use. This type is not valid when the source operation specifies decrypt using template key and the receiver operation specifies encrypt using master key variant 1, encrypt using master key variant 2, or encrypt using template key. Any invalid combination will result in a template value invalid exception being signaled.

The use variant 1 of template key template key type specifies that variant 1 of the template key is to be calculated and used for the required operation.

The use variant 2 of template key template key type specifies that variant 2 of the template key is to be calculated and used for the required operation.

The use variant 3 of template key template key type specifies that variant 3 of the template key is to be calculated and used for the required operation.

The template key field specifies the key to be used for a cipher key operation which specifies usage of the template key for an encrypt or decrypt operation. This field is ignored when the template key is not to be used.

The PIN operation parameters contain the information needed to perform the generate PIN, verify PIN, or translate PIN functions.

The PIN validation key is used when a generate PIN, verify PIN, or translate PIN function is being performed. For generate PIN and verify PIN, this field contains the PIN validation key encrypted under variant 3 of the host master key. For the translate PIN function, this field contains the input PIN protection key encrypted under variant 3 of the host master key.

The PIN decimalization data is used when a generate PIN or verify PIN function is being performed. This data is used to decimalize enciphered validation data from the source operand.

The PIN protection key is used when a verify PIN or translate PIN function is being performed. For verify PIN, this contains the input PIN protection key encrypted under variant 3 of the host master key. For the translate PIN function, this field contains the output PIN protection key encrypted under variant 1 of the host master key.

The PIN pad character is used only when a verify PIN function is being performed. This is the PIN pad character presented by the application.

The PIN check length is used only when a verify PIN function is being performed. This is the PIN check length presented by the application.

The encrypted PIN is used only when a verify PIN function is being performed. This is the encrypted PIN presented by the application.

Valid results are produced for the case of the receiver and source operands being coincident with one another. The source data is accessed first, then the result is stored in the receiver.

Partial overlap between the source and receiver operands may produce invalid results.

All keys processed by this instruction which are encrypted under variant 1, 2, or 3 of the host master key must have odd parity in each byte when decrypted for use. If the parity of any key byte is not odd, a key parity invalid exception will be signaled.

The following table lists the source operation values and the valid receiver operation and template key type values for each source operation.

Source Operation	Valid Receiver Operations	Valid Template Key Types
Hex 00	Hex 00 Hex 01 Hex 02 Hex 03 Hex 04 Hex 05 Hex 06 Hex 07 Hex 08	n/a
Hex 01	Hex 04	Hex 00 Hex 01
Hex 02	Hex 00	Hex 01
Hex 03	Hex 00 Hex 09	
Hex 04	Hex 04	Hex 02 Hex 03 Hex 04
Hex 05	Hex 04	Hex 02 Hex 03 Hex 04
Hex 06	Hex 04	Hex 02 Hex 03 Hex 04

Events

000C Machine resource
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
07 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
0F Master key not defined or invalid		X		
10 Weak key not valid	X	X		
11 Key parity invalid		X		
10 Damage Encountered				
44 Partial system object damage				X
1C Machine Dependent Exception				
03 Machine storage limit exceeded				X
08 Requested function not available				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2E Resource Control Limit				
02 Process storage limit exceeded				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid	X	X		

COMPARE BYTES LEFT-ADJUSTED (CMPBLAB or CMPBLAI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4, 5]
1CC2	Branch options	Compare operand 1	Compare operand 2	Branch target
18C2	Indicator options			Indicator target

Operand 1: Numeric scalar or character scalar.

Operand 2: Numeric scalar or character scalar.

Operand 3 [*4, 5*]:

- *Branch Form*–Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*–Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction compares the logical string values of two left-adjusted compare operands. The logical string value of the first compare operand is compared with the logical string value of the second compare operand (no padding done). Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The compare operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The compare operands are compared byte by byte, from left to right with no numeric conversions performed. The length of the operation is equal to the length of the shorter of the two compare operands. The comparison begins with the leftmost byte of each of the compare operands and proceeds until all bytes of the shorter compare operand have been compared or until the first unequal pair of bytes is encountered.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either or both compare operands is that the instruction's resultant condition is equal.

Resultant Conditions: The scalar first compare operand has a higher, lower, or equal string value than the second compare operand.

Events

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3 [4, 5]**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

COMPARE BYTES LEFT-ADJUSTED WITH PAD (CMPBLAPB or CMPBLAPI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3	Operand 4 [5, 6]
1CC3	Branch options	Compare operand 1	Compare operand 2	Pad	Branch target
18C3	Indicator options				Indicator target

Operand 1: Numeric scalar or character scalar.

Operand 2: Numeric scalar or character scalar.

Operand 3: Numeric scalar or character scalar.

Operand 4 [*5, 6*]:

- *Branch Form*–Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*–Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 4 and optional for operands 5 and 6. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction compares the logical string values of two left-adjusted compare operands (padded if needed). The logical string value of the first compare operand is compared with the logical string value of the second compare operand. Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The compare operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The compare operands are compared byte by byte, from left to right with no numeric conversions being performed.

The length of the operation is equal to the length of the longer of the two compare operands. The shorter of the two compare operands is logically padded on the right with the 1-byte value indicated in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. The comparison begins with the leftmost byte of each of the compare operands and proceeds until all the bytes of the longer of the two compare operands have been compared or until the first unequal pair of bytes is encountered. All excess bytes in the longer of the two compare operands are compared to the pad value.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for one of the compare operands is that the other compare operand is compared with an equal length string of pad character values. When a null substring reference is specified for both compare operands, the resultant condition is equal.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Resultant Conditions: The scalar first compare operand has a higher, lower, or equal string value than the second compare operand.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4 [5, 6]**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op code extender field					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand					X
0A Invalid operand length	X	X			
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2C Program Execution					
04 Branch target invalid					X

COMPARE BYTES RIGHT-ADJUSTED (CMPBRAB or CMPBRAI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4, 5]
1CC6	Branch options	Compare operand 1	Compare operand 2	Branch target
18C6	Indicator options			Indicator target

Operand 1: Numeric scalar or character scalar.

Operand 2: Numeric scalar or character scalar.

Operand 3 [*4, 5*]:

- *Branch Form*—Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*—Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or the indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction compares the logical string values of two right-adjusted compare operands. The logical string value of the first compare operand is compared with the logical string value of the second compare operand (no padding done). Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The compare operands can be either string or numeric. Any numeric operands are interpreted as logical character strings.

The compare operands are compared byte by byte, from left to right with no numeric conversions performed. The length of the operation is equal to the length of the shorter of the two compare operands. The comparison begins with the leftmost byte of each of the compare operands and proceeds until all bytes of the shorter compare operand have been compared or until the first unequal pair of bytes is encountered.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either or both compare operands is that the instruction's resultant condition is equal.

Resultant Conditions: The scalar first compare operand has a higher, lower, or equal string value than the second compare operand.

Events

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3 [4, 5]**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length	X	X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid			X	X

COMPARE BYTES RIGHT-ADJUSTED WITH PAD (CMPBRAPB or CMPBRAPI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3	Operand 4 [5, 6]
1CC7	Branch options	Compare operand 1	Compare operand 2	Pad	Branch target
18C7	Indicator options				Indicator target

Operand 1: Numeric scalar or character scalar.

Operand 2: Numeric scalar or character scalar.

Operand 3: Numeric scalar or character scalar.

Operand 4 [*5, 6*]:

- *Branch Form*–Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*–Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or the indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 4 and optional for operands 5 and 6. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction compares the logical string values of the right-adjusted compare operands (padded if needed). The logical string value of the first compare operand is compared with the logical string value of the second compare operand. Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The compare operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The compare operands are compared byte by byte, from left to right with no numeric conversions performed.

The length of the operation is equal to the length of the longer of the two compare operands. The shorter of the two compare operands is logically padded on the left with the 1-byte value indicated in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. The comparison begins with the leftmost byte of the longer of the compare operands. Any excess bytes (on the left) in the longer compare operand are compared with the pad value. All other bytes are compared with the corresponding bytes in the other compare operand. The operation proceeds until all bytes in the longer operand are compared or until the first unequal pair of bytes is encountered.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for one of the compare operands is that the other compare operand is compared with an equal length string of pad character values. When a null substring reference is specified for both compare operands, the instruction's resultant condition is equal.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Resultant Conditions: The scalar first compare operand has a higher, lower, or equal string value than the second compare operand.

Events

0000C Machine resource

0201 Machine auxiliary storage threshold exceeded

0010 Process

0701 Maximum processor time exceeded

0801 Process storage limit exceeded

0016 Machine observation

0101 Instruction reference

0017 Damage set

0401 System object damage set

0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4 [5, 6]**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op code extender field					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand					X
0A Invalid operand length	X	X			
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2C Program Execution					
04 Branch target invalid			X	X	

COMPARE NUMERIC VALUE (CMPNVB or CMPNVI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4, 5, 6]
1C46	Branch options	Compare operand 1	Compare operand 2	Branch target
1846	Indicator options			Indicator target

Operand 1: Numeric scalar.

Operand 2: Numeric scalar.

Operand 3 [*4, 5, 6*]:

- *Branch Form*—Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*—Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or indicator option is required by the instruction. The extender field is required along with from one to four branch targets (for branch option) or one to four indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The signed numeric value of the first compare operand is compared with the numeric value of the second compare operand. Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

Both the compare operands must be numeric with any implicit conversions occurring according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual*. For a decimal operation, alignment of the assumed decimal point takes place by padding with 0's on the right end of the compare operand with lesser precision.

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

The length of the operation is equal to the length of the longer of the two compare operands. The shorter of the two operands is adjusted to the length of the longer operand according to the rules of arithmetic operations outlined in the *Functional Concepts Manual*.

Floating-point comparisons use exponent comparison and significand comparison. For a denormalized floating-point number, the comparison is performed as if the denormalized number had first been normalized.

For floating-point, two values compare unordered when at least one comparand is NaN. Every NaN compares unordered with everything including another NaN value.

Floating-point comparisons ignore the sign of zero. Positive zero always compares equal with negative zero.

A floating-point invalid operand exception is signaled when two floating-point values compare unordered and no branch or indicator option exists for any of the unordered, negation of unordered equal, or negation of equal resultant conditions.

When a comparison is made between a floating-point compare operand and a fixed-point decimal compare operand that contains fractional digit positions, a floating-point inexact result exception may be signaled because of the implicit conversion from decimal to floating-point.

Resultant Conditions: High, low, or equal–The first compare operand has a higher, lower, or equal numeric value than the second compare operand. Unordered–The first compare operand is unordered compared to the second compare operand.

Events

0000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3 [4, 5]**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data	X	X		
03 Decimal point alignment	X	X		
09 Floating-point invalid operand		X		X
0D Floating-point inexact result				X
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid			X	

COMPUTE ARRAY INDEX (CAI)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
1044	Array index	Subscript A	Subscript B	Dimension

Operand 1: Binary(2) variable scalar.

Operand 2: Binary(2) scalar.

Operand 3: Binary(2) scalar.

Operand 4: Binary(2) constant scalar object or immediate operand.

Description: This instruction provides the ability to reduce multidimensional array subscript values into a single index value which can then be used in referencing the single-dimensional arrays of the system. This index value is computed by performing the following arithmetic operation on the indicated operands.

Array Index = Subscript A + ((Subscript B-1) X Dimension)

The signed numeric value of the subscript B operand is decreased by 1 and multiplied by the numeric value of the dimension operand. The result of this multiplication is added to the subscript A operand and the sum is placed in the array index operand.

All the operands must be binary with any implicit conversions occurring according to the rules of arithmetic operations. The usual rules of algebra are observed concerning the subtraction, addition, and multiplication of operands.

This instruction provides for mapping multidimensional arrays to single-dimensional arrays. The elements of an array with the dimensions (d1, d2, d3, ..., dn) can be defined as a single-dimensional array with d1*d2*d3*...*dn elements. To reference a specific element of the multidimensional array with subscripts (s1,s2,s3,...sn), it is necessary to convert the multiple subscripts to a single subscript for use in the single-dimensional System/38 array. This single subscript can be computed using the following:

```
s1+((s2-1)*d1)+(s3-1)*d1*d2)+...+((sn-1)*d*d2*d3*...*dm),
   where m=n-1
```

The CAI instruction is used to form a single index value from two subscript values. To reduce N subscript values into a single index value, N-1 uses of this instruction are necessary.

Assume that S1, S2, and S3 are three subscript values and that D1 is the size of one dimension, D2 is the size of the second dimension, and the D1D2 is the product of D1 and D2. The following two uses of this instruction reduce the three subscripts to a single subscript.

```
CAI INDEX, S1, S2, D1          Calculates s1+(s2-1)*d1
CAI INDEX, INDEX, S3, D1D2     Calculatess1+(s2-1)
                                *d1+(s3-1)*d2*d1
```

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands				Other
	1	2	3	4	
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X

COMPUTE MATH FUNCTION USING ONE INPUT VALUE (CMF1)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
100B	Receiver	Controls	Source

Operand 1: Numeric variable scalar.

Operand 2: Character(2) scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CMF1I	180B	Indicator
CMF1B	1C0B	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The mathematical function, indicated by the controls operand, is performed on the source operand value and the result is placed in the receiver operand.

The calculation is always done in floating-point.

The source and receiver operands must both be specified as floating-point with the same length (4 bytes for short format or 8 bytes for long format).

The controls operand must be a character scalar that specifies which mathematical function is to be performed. It must be at least 2 bytes in length and has the following format:

- Controls operand Char(2)
  - Hex 0001 = Sine
  - Hex 0002 = Arc sine
  - Hex 0003 = Cosine
  - Hex 0004 = Arc cosine
  - Hex 0005 = Tangent
  - Hex 0006 = Arc tangent
  - Hex 0007 = Cotangent
  - Hex 0010 = Exponential function
  - Hex 0011 = Logarithm based e (natural logarithm)
  - Hex 0012 = Sine hyperbolic
  - Hex 0013 = Cosine hyperbolic
  - Hex 0014 = Tangent hyperbolic
  - Hex 0015 = Arc tangent hyperbolic
  - Hex 0020 = Square root
  - All other values are reserved

The controls operand mathematical functions are as follows:

- Hex 0001–Sine

  The sine of the numeric value of the source operand, whose value is considered to be in radians, is computed and placed in the receiver operand.

  The result is in the range:

  -1 ≤ SIN(x) ≤ 1

- Hex 0002–Arc sine

  The arc sine of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  -pi/2 ≤ ASIN(x) ≤ +pi/2

- Hex 0003–Cosine

  The cosine of the numeric value of the source operand, whose value is considered to be in radians, is computed and placed in the receiver operand.

  The result is in the range:

  -1 ≤ COS(x) ≤ 1

- Hex 0004–Arc cosine

  The arc cosine of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  0 ≤ ACOS(x) ≤ pi

- Hex 0005–Tangent

  The tangent of the source operand, whose value is considered to be in radians, is computed and the result is placed in the receiver operand.

  The result is in the range:

  -infinity ≤ TAN(x) ≤ +infinity

- Hex 0006–Arc tangent

  The arc tangent of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  -pi/2 ≤ ATAN(x) ≤ pi/2

- Hex 0007–Cotangent

  The cotangent of the source operand, whose value is considered to be in radians, is computed and the result is placed in the receiver operand.

  The result is in the range:

  -infinity ≤ COT(x) ≤ +infinity

- Hex 0010–Exponential function

  The computation e power (source operand) is performed and the result is placed in the receiver operand.

  The result is in the range:

  0 ≤ EXP(x) ≤ +infinity

- Hex 0011–Logarithm based e (natural logarithm)

  The natural logarithm of the source operand is computed and the result is placed in the receiver operand.

  The result is in the range:

  -infinity ≤ LN(x) ≤ +infinity

- Hex 0012–Sine hyperbolic

  The sine hyperbolic of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  -infinity ≤ SINH(x) ≤ +infinity

- Hex 0013–Cosine hyperbolic

  The cosine hyperbolic of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  +1 ≤ COSH(x) ≤ +infinity

- Hex 0014–Tangent hyperbolic

  The tangent hyperbolic of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  +1 ≤ TANH(x) ≤ +1

- Hex 0015–Arc tangent hyperbolic

  The inverse of the tangent hyperbolic of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  -infinity ≤ ATANH(x) ≤ +infinity

- Hex 0020–Square root

  The square root of the numeric value of the source operand is computed and placed in the receiver operand.

  The result is in the range:

  0 ≤ SQRT(x) ≤ +infinity

Null substring references (a length value of zero) cannot be specified for this instruction.

The following chart shows some special cases for certain arguments (X) of the different mathematical functions in projective and affine mode.

X Function	Masked NaN	Unmasked NaN	+infinity	-infinity	+0	-0	Maximum Value	Minimum Value	Other
Sine	g	A(e)	A(f)	A(f)	+0	-0	A(1,f)	A(1,f)	B(3)
Arc sine	g	A(e)	A(f)	A(f)	+0	-0	A(6,f)	A(6,f)	–
Cosine	g	A(e)	A(f)	A(f)	+1	+1	A(1,f)	A(1,f)	B(3)
Arc cosine	g	A(e)	A(f)	A(f)	+pi/2	+pi/2	A(6,f)	A(6,f)	–
Tangent	g	A(e)	A(f)	A(f)	+0	-0	A(1,f)	A(1,f)	B(3)
Arc tangent	g	A(e)	+pi/2	-pi/2	+0	-0	–	–	–
Cotangent	g	A(e)	A(f)	A(f)	+inf	-inf	A(1,f)	A(1,f)	B(3)
Exponent	g	A(e)	+inf	+0	+1	+1	C(4,a)	D(5,b)	–
Logarithm	g	A(e)	+inf	A(f)	-inf	-inf	–	–	A(2,f)
Sine hyperbolic	g	A(e)	+inf	-inf	+0	-0	–	–	–
Cosine hyperbolic	g	A(e)	+inf	+inf	+1	+1	–	–	–
Tangent hyperbolic	g	A(e)	+1	-1	+0	-0	–	–	–
Arc tangent hyperbolic	g	A(e)	A(f)	A(f)	+0	-0	A(6,f)	A(6,f)	–
Square root	g	A(e)	+inf	A(f)	+0	-0	–	–	A(2,f)

Capital letters in the chart indicate the exceptions, small letters indicate the returned results, and Arabic numerals indicate the limits of the arguments (X) as defined in the following lists:

A = Floating-point invalid operand (no result stored if unmasked; if masked, occurrence bit is set)
B = Floating-point inexact result (result is stored whether or not exception is masked)
C = Floating-point overflow (no result is stored if unmasked; if masked, occurrence bit is set)
D = Floating-point underflow (no result is stored if unmasked; occurrence bit is always set)

a = Result follows the rules that depend on round mode
b = Result is +0 or a denormalized value
c = Result is +infinity
d = Result is -infinity
e = Result is the masked form of the input NaN
f = Result is the system default masked NaN
g = Result is the input NaN
inf = Result is infinity

1 = | pi * 2**50 | = Hex 432921FB54442D18
2 = Argument is in the range: -inf < x < -0
3 = | pi * 2**26 | = Hex 41A921FB54442D18

4 = 1n(2**1023) = Hex 40862E42FEFA39EF
5 = 1n(2**-1021.4555) = Hex C086200000000000
6 = Argument is in the range: $-1 \le x \le +1$

The following chart provides accuracy data for the mathematical functions that can be invoked by this instruction.

				Accuracy Data			
Function	Sample Selection			Relative Error (e)		Absolute Error (E)	
Name	A	Range of x	D	MAX(e)	SD(e)	MAX(E)	SD(E)
Arc cosine	9	0 <= x <= 3.14	U			8.26 * 10**-14	2.11 * 10**-15
Arc sine	10	-1.57 <= x <= 1.57	U	1.02 * 10**-13	2.66 * 10**-15		
Arc tangent	1	-pi/2 < x < pi/2	1			3.33 * 10**-16	9.57 * 10**-17
Arc tangent hyperbolic	14	-3 <= x <= 3	U			1.06 * 10**-14	1.79 * 10**-15
Cosine	–	(See Sine below)	–	–	–	–	–
Cosine hyperbolic		(See Sine Hyperbolic below)					
Cotangent	11	-10 <= x <= 100	U	4.83 * 10**-16	1.48 * 10**-16		
		.000001 <= x <= .001	U	4.36 * 10**-16	1.49 * 10**-16		
		4000 <= x <= 4000000	U	5.72 * 10**-16	1.46 * 10**-16		
Exponential	2	-100 <= x <= 300	U	5.70 * 10**-14	1.13 * 10**-14		
Natural logarithm	3	0.5 <= x <= 1.5	U			2.77 * 10**-16	8.01 * 10**-17
	4	-100 <= x <= 700	E	2.17 * 10**-16	7.37 * 10**-17		
Sine cosine	5	-10 <= x <= 100	U			2.22 * 10**-16	1.31 * 10**-16
		.000001 <= x <= .001	U			2.22 * 10**-16	1.56 * 10**-16
		4000 <= x <= 4000000	U			2.22 * 10**-16	1.28 * 10**-16
	6	-10 <= x <= 100	U			3.33 * 10**-16	8.39 * 10**-17
		.000001 <= x <= .001	U			4.33 * 10**-19	1.28 * 10**-19
		4000 <= x <= 4000000	U			3.33 * 10**-16	8.17 * 10**-17
Sine/cosine hyperbolic	12	-100 <= x <= 300	U	6.31 * 10**-16	1.97 * 10**-16		
Square root	7	-100 <= x <= 700	E	4.13 * 10**-16	1.27 * 10**-16		
Tangent	8	-10 <= x <= 100	U	4.59 * 10**-16	1.54 * 10**-16		
		.000001 <= x <= .001	U	4.42 * 10**-16	1.44 * 10**-16	3.25 * 10**-19	8.06 * 10**-20
		4000 <= x <= 4000000	U	4.77 * 10**-16	1.43 * 10**-16		
Tangent hyperbolic	13	-100 <= x <= 300	U	2.22 * 10**-15	6.26 * 10**-17	2.22 * 10**-16	3.64 * 10**-17

Algorithm Notes:

1. f(x) = x, and g(x) = ATAN(TAN(x)).
2. f(x) = e**x, and g(x) = e**(1n(e**x)).
3. f(x) = 1n(x), and g(x) = 1n(e**(1n(x))).
4. f(x) = x, and g(x) = 1n(e**x).
5. Sum of squares algorithm. f(x) = 1, and g(x) = SIN(x))**2 + (COS(x))**2.
6. Double angle algorithm. f(x) - SIN(2x), and g(x) = 2*(SIN(x)*COS(x)).
7. f(x) = e(**x, and g(x) = (SQR(e**x))**2.
8. f(x) = TAN(x), and g(x) = SIN(x) / COS(x).
9. f(x) = x, and g(x) = ACOS(COS(x)).
10. f(x) = x, and g(x) = ASIN(SIN(x)).
11. f(x) = COT(x), and g(x) = COS(x) / SIN(x).
12. f(x) = SINH(2x), and g(x) = 2*(SINH(x)*COSH(x)).
13. f(x) = TANH(x), and g(x) = SINH(x) / COSH(x).
14. f(x) = x, and g(x) = ATANH(TANH(x)).

Distribution Note: The sample input arguments were tangents of numbers, x, uniformly distributed between -pi/2 and +pi/2.

The vertical columns in the accuracy data chart have the following meanings:

- *Function Name*: This column identifies the principal mathematical functions evaluated with entries arranged in alphabetical order by function name.

- *Sample Selection*: This column identifies the selection of samples taken for a particular math function through the following subcolumns:
  - *A*: identifies the algorithm used against the argument, x, to gather the accuracy samples. The numbers in this column refer to notes describing the functions, f(x) and g(x), which were calculated to test for the anticipated relation where f(x) should equal g(x). An accuracy sample then, is an evaluation of the degree to which this relation held true. The algorithm used to sample the arctangent function, for example, defines g(x) to first calculate the tangent of x to provide an appropriate distribution of input arguments for the arctangent function. Since f(x) is defined simply as the value of x, the relation to be evaluated is then x=ARCTAN(TAN(x)). This type of algorithm, where a function and its inverse are used in tandem, is the usual type employed to provide the appropriate comparison values for the evaluation.
  - *Range of x*: gives the range of x used to obtain the accuracy samples. The test values for x are uniformly distributed over this range. It should be noted that x is not always the direct input argument to the function being tested; it is sometimes desirable to distribute the input arguments in a nonuniform fashion to provide a more complete test of the function (see column D below). For each function, accuracy data is given for one or more segments within the valid range of x. In each case, the numbers given are the most meaningful to the function and range under consideration.
  - *D*: identifies the distribution of arguments input to the particular function being sampled. The letter E indicates an exponential distribution. The letter U indicates a uniform distribution. A number refers to a note providing detailed information regarding the distribution.

- *Accuracy Data*: The maximum relative error and standard deviation of the relative error are generally useful and revealing statistics; however, they are useless for the range of a function where its value becomes zero. This is because the slightest error in the argument can cause an unpredictable fluctuation in the magnitude of the answer. When a small argument error would have this effect, the maximum absolute error and standard deviation of the absolute error are given for the range.
  - *Relative Error (e)*: The maximum relative error and standard deviation (root mean square) of the relative error are defined:

    MAX(e):

    = MAX(ABS((f(x) - g(x)) / f(x)))

    where: MAX selects the largest of its arguments and ABS takes the absolute value of its argument.

    SD(e):

    = SQR((1/N) SUMSQ((f(x) - g(x)) / f(x)))

    where: SQR takes the square root of its argument and SUMSQ takes the summation of the squares of its arguments over all of the test cases.

  - *Absolute Error (E)*: The maximum absolute error produced during the testing and the standard deviation (root mean square) of the absolute error are:

    MAX(E):

    = MAX(ABS(f(x) - g(x)))

    where: the operators are those defined above.

    SD(E):

    = SQR((1/N) SUMSQ(f(x) - g(x)))

    where: the operators are those defined above.

Resultant Conditions: Positive, negative, or zero–The algebraic value of the receiver operand is positive, negative, or zero. Unordered–The value assigned to the floating-point result is NaN.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0201 Object location reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
06 Floating-point overflow	X			
07 Floating-point underflow	X			
09 Floating-point invalid operand			X	
0D Floating-point inexact result	X			
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op-code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Invalid branch target				X
2E Resource Control Limit				
02 Process storage limit exceeded				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
03 Scalar value invalid		X		

COMPUTE MATH FUNCTION USING TWO INPUT VALUES (CMF2)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
100C	Receiver	Controls	Source 1	Source 2

Operand 1: Numeric variable scalar.

Operand 2: Character(2) scalar.

Operand 3: Numeric scalar.

Operand 4: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CMF2I	180C	Indicator
CMF2B	1C0C	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The mathematical function, indicated by the controls operand, is performed on the source operand values and the result is placed in the receiver operand.

The calculation is always done in floating-point.

The source and receiver operands must both be specified as floating-point with the same length (4 bytes for short format or 8 bytes for long format).

The controls operand must be a character scalar that specifies which mathematical function is to be performed. It must be at least 2 bytes in length and have the following format:

- Controls operand Char(2)
  - Hex 0001= Power (x to the y)
  - All other values are reserved

The computation x power y, where x is the first source operand and y is the second source operand, is performed and the result is placed in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

The following chart shows some special cases for certain arguments of the power function (x**y). Within the chart, the capitalized letters X and Y refer to the absolute value of the arguments x and y; that is, X = |x| and Y = |y|.

y / x	−inf	y<0, y=2n+1	y<0 y=2n	y<0 real	−1	−1/2	+0 or −0	+1/2	+1	y>0 y=2n+1	y>0 y=2n	y>0 real	+inf	Masked NaN	Un-masked NaN
+inf	+0	+0	+0	+0	+0	+0	+1	+inf	+inf	+inf	+inf	+inf	+inf	b	A(c)
x>1	+0	$\frac{+1}{\text{x**Y}}$	$\frac{+1}{\text{x**Y}}$	$\frac{+1}{\text{x**Y}}$	$\frac{+1}{\text{x}}$	$\frac{+1}{\text{SQRT(x)}}$	+1	SQRT(x)	x	x**y	x**y	x**y	+inf	b	A(c)
x=+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	b	A(c)
0<x<1	+inf	$\frac{+1}{\text{x**Y}}$	$\frac{+1}{\text{x**Y}}$	$\frac{+1}{\text{x**Y}}$	$\frac{+1}{\text{x}}$	$\frac{+1}{\text{SQRT(x)}}$	+1	SQRT(x)	x	x**y	x**y	x**y	+0	b	A(c)
x=+0	E(f)	E(f)	E(f)	E(f)	E(f)	E(f)	+1	+0	+0	+0	+0	+0	+0	b	A(c)
x=−0	E(f)	E(g)	E(f)	E(f)	E(g)	E(g)	+1	−0	−0	−0	+0	+0	+0	b	A(c)
0>x>−1	A(a)	$\frac{-1}{\text{X**Y}}$	$\frac{+1}{\text{X**Y}}$	A(a)	$\frac{-1}{\text{X}}$	A(a)	+1	A(a)	x	−X**y	|X**y	A(a)	A(a)	b	A(c)
x=−1	A(a)	−1	+1	A(a)	−1	A(a)	+1	A(a)	−1	−1	+1	A(a)	A(a)	b	A(c)
x<−1	A(a)	$\frac{-1}{\text{X**Y}}$	$\frac{+1}{\text{X**Y}}$	A(a)	$\frac{-1}{\text{X}}$	A(a)	+1	A(a)	x	−|X**y	X**y	A(a)	A(a)	b	A(c)
x=−inf	A(a)	−0	+0	A(a)	−0	A(a)	+1	A(a)	−inf	−inf	+inf	A(a)	A(a)	b	A(c)
Masked NaN	b	b	b	b	b	b	b	b	b	b	b	b	b	d	A(e)
Un-masked NaN	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(e)	A(e)

Capital letters in the chart indicate the exceptions and small letters indicate the returned results as defined in the following list:

A	= Floating-point invalid operand	d	= Result is the larger NaN
E	= Divide by zero	e	= Result is the larger NAN masked
a	= Result is the system default masked NaN	f	= Result is +infinity
b	= Result is the same NaN	g	= Result is −infinity
c	= Result is the same NaN masked		

The following chart provides accuracy data for the mathematical function that can be invoked by this instruction.

Function Name	Sample Selection		Accuracy Data	
	x	y	MAX(e)	SD(e)
Power	1/3	-345 <= y <= 330	4.99 * 10**-16	1.90 * 10**-16
	.75	-1320 <= y <= 1320	2.96 * 10**-16	2.39 * 10**-16
	.9	-3605 <= y <= 3605	1.23 * 10**-16	1.02 * 10**-16
	10	-165 <= y <= 165	7.10 * 10**-16	3.18 * 10**-16
	712	-57 <= y <= 57	1.75 * 10**-15	7.24 * 10**-16

The vertical columns in the accuracy data chart have the following meanings:

- *Function Name*: This column identifies the mathematical function.

- *Sample Selection*: This column identifies the selection of samples taken for the power function. The algorithm used against the arguments, x and y, to gather the accuracy samples was a test for the anticipated relation where f(x) should equal g(x,y):

  where:

  f (x) = x

  g(x,y) = (x**y)**(1/y)

  An accuracy sample then, is an evaluation of the degree to which this relation held true.

  The range of argument values for x and y were selected such that x was held constant at a particular value and y was uniformly varied throughout a range of values which avoided overflowing or underflowing the result field. The particular values selected are indicated in the subcolumns entitled x and y.

- *Accuracy Data*: The maximum relative error and standard deviation (root mean square) of the relative error are generally useful and revealing statistics. These statistics for the relative error, (e), are provided in the following subcolumns:

  MAX(e):

  = MAX(ABS((f(x) - g(x)) / f(x)))

  where: MAX selects the largest of its arguments and ABS takes the absolute value of its argument.

  SD(e):

  = SQR((1/N) SUMSQ((f(x) - g(x)) / f(x)))

  where: SQR takes the square root of its argument and SUMSQ takes the summation of the squares of its arguments over all of the test cases.

Resultant Conditions: Positive, negative, or zero–The algebraic value of the receiver operand is positive, negative, or zero. Unordered–The value assigned to the floating-point result is NaN.

Events

0000C Machine resource
0201 Machine auxiliary storage threshold exceeded

000D Machine status
0101 Machine check

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment violation	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0C Computation					
06 Floating-point overflow	X				
07 Floating-point underflow	X				
09 Floating-point invalid operand			X	X	
0C Invalid floating-point conversion	X				
0D Floating-point inexact result	X				
0E Floating-point zero divide	X				
10 Damage Encountered					
44 Partial system object damage					X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op-code extender field					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand					X
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2C Program Execution					
04 Invalid branch target					X
2E Resource Control Limit					
02 Process storage limit exceeded					X
32 Scalar Specification					
01 Scalar type invalid	X	X	X	X	
03 Scalar value invalid		X			

CONCATENATE (CAT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10F3	Receiver	Source 1	Source 2

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar.

Description: The character string value of the second source operand is joined to the right end of the character string value of the first source operand. The resulting string value is placed (left-adjusted) in the receiver operand.

The length of the operation is equal to the length of the receiver operand with the resulting string truncated or is logically padded on the right end accordingly. The pad value for this instruction is hex 40.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1, 2, and 3. The effect of specifying a null substring reference for one source operand is that the other source operand is used as the result of the concatenation. The effect of specifying a null substring reference for both source operands is that the bytes of the receiver are each set with a value of hex 40. The effect of specifying a null substring reference for the receiver is that a result is not set regardless of the value of the source operands.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation				X
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X

CONVERT BSC TO CHARACTER (CVTBC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10AF	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(3) variable scalar (fixed-length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTBCI	18AF	Indicator
CVTBCB	1CAF	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a string value from the BSC (binary synchronous communications) compressed format to a character string. The operation converts the source (operand 3) from the BSC compressed format to character under control of the controls (operand 2) and places the result into the receiver (operand 1).

The source and receiver operands must both be character strings.

The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 3 bytes in length and have the following format:

• Controls operand	Char(3)
– Source offset	Bin(2)
– Record separator	Char(1)

The source offset specifies the offset where bytes are to be accessed from the source operand. If the offset is equal to or greater than the length specified for the source operand (it identifies a byte beyond the end of the source operand), a template value invalid exception is signaled. As output from the instruction, the source offset is set to specify the offset that indicates how much of the source is processed when the instruction ends.

The record separator, if specified with a value other than hex 01, contains the value used to separate converted records in the source operand. A value of hex 01 specifies that record separators do not occur in the converted records in the source.

Only the first 3 bytes of the controls operand are used. Any excess bytes are ignored.

The operation begins by accessing the bytes of the source operand located at the offset specified in the source offset. This is assumed to be the start of a record. The bytes of the record in the source operand are converted into the receiver record according to the following algorithm.

The strings to be built in the receiver are contained in the source as blank compression entries and strings of consecutive nonblank characters.

The format of the blank compression entries occurring in the source are as follows:

• Blank compression entry	Char(2)
– Interchange group separator	Char(1)
– Count of compressed blanks	Char(1)

The interchange group separator has a fixed value of hex 1D.

The compressed blanks count provides for describing up to 63 compressed blanks. The count of the number of blanks (up to 63) to be decompressed is formed by subtracting hex 40 from the value of the count field. The count field can vary from a value of hex 41 to hex 7F. If the count field contains a value outside of this range, a conversion exception is signaled.

Strings of blanks described by blank compression entries in the source are repeated in the receiver the number of times specified by the blank compression count.

Nonblank strings in the source are copied into the receiver intact with no alteration.

If the receiver record is filled with converted data without encountering the end of the source operand, the instruction ends with a resultant condition of *completed record*. This can occur in two ways. If a record separator was not specified, the instruction ends when enough bytes have been converted from the source to fill the receiver. If a record separator was specified, the instruction ends when a source byte is encountered with that value prior to or just after filling the receiver record. The offset value for the source locates the byte following the last source record (including the record separator) for which conversion was completed. When the record separator value is encountered, any remaining bytes in the receiver are padded with blanks.

If the end of the source operand is encountered (whether or not in conjunction with a record separator or the filling of the receiver), the instruction ends with a resultant condition of *source exhausted*. The offset value for the source locates the byte following the last byte of the source operand. The remaining bytes in the receiver after the converted record are padded with blanks.

If the converted form of a record cannot be completely contained in the receiver, the instruction ends with a resultant condition of *truncated record*. The offset value for the source locates the byte following the last source byte for which conversion was performed, unless a blank compression entry was being processed. In this case, the source offset is set to locate the byte after the blank compression entry. If the source does not contain record separators, this condition can only occur for the case in which a blank compression entry was being converted when the receiver record became full.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Completed record–The receiver record has been completely filled with converted data from a source record. Source exhausted–All of the bytes in the source operand have been converted into the receiver operand. Truncated record–The receiver record cannot contain all of the converted data from the source record.

Events

000C Machine resource
  0201 Machine auxiliary storage exceeded

000D Machine status
  0101 Machine check

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands			Other
	1	2	3	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
01 Conversion			X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Invalid branch target				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		

CONVERT CHARACTER TO BSC (CVTCB)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
108F	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(3) variable scalar (fixed-length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTCBI	188F	Indicator
CVTCBB	1C8F	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operations immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a string value from character to BSC (binary synchronous communications) compressed format. The operation converts the source (operand 3) from character to the BSC compressed format under control of the controls (operand 2) and places the result into the receiver (operand 1).

The source and receiver operands must both be character strings.

The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 3 bytes in length and have the following format:

• Controls operand	Char(3)
– Receiver offset	Bin(2)
– Record separator	Char(1)

The receiver offset specifies the offset where bytes are to be placed into the receiver operand. If the offset is equal to or greater than the length specified for the receiver operand (it identifies a byte beyond the end of the receiver), a template value invalid exception is signaled. As output from the instruction, the source offset is set to specify the offset that indicates how much of the receiver has been filled when the instruction ends.

The record separator, if specified with a value other than hex 01, contains the value used to separate converted records in the receiver operand. A value of hex 01 specifies that record separators are not to be placed into the receiver to separate converted records.

Only the first 3 bytes of the controls operand are used. Any excess bytes are ignored.

The source operand is assumed to be one record. The bytes of the record in the source operand are converted into the receiver operand at the location specified in the receiver offset according to the following algorithm.

The bytes of the source record are interrogated to identify the strings of consecutive blank (hex 40) characters and the strings of consecutive nonblank characters which occur in the source record. Only three or more blank characters are treated as a blank string for purposes of conversion into the receiver.

As the blank and nonblank strings are encountered in the source they are packaged into the receiver.

Blank strings are reflected in the receiver as one or more blank compression entries. The format of the blank compression entries built into the receiver are as follows:

• Blank compression entry	Char(2)
– Interchange group separator	Char(1)
– Count of compressed blanks	Char(1)

The interchange group separator has a fixed value of hex 1D.

The compressed blanks count provides for compressing up to 63 blanks. The value of the count field is formed by adding hex 40 to the actual number of blanks (up to 63) to be compressed. The count field can vary from a value of hex 43 to hex 7F.

Nonblank strings are copied into the receiver intact with no alteration or additional control information.

When the end of the source record is encountered the record separator value if specified is placed into the receiver and the instruction ends with a resultant condition of *source exhausted*. The offset value for the receiver locates the byte following the converted record in the receiver. The value of the remaining bytes in the receiver after the converted record is unpredictable.

If the converted form of a record cannot be completely contained in the receiver (including the record separator if specified), the instruction ends with a resultant condition of *receiver overrun*. The offset value for the receiver remains unchanged. The remaining bytes in the receiver, starting with the byte located by the receiver offset, are unpredictable.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Source exhausted–All of the bytes in the source operand have been converted into the receiver operand. Receiver overrun–An overrun condition in the receiver operand was detected before all of the bytes in the source operand were processed.

Events

000C Machine resources
0201 Machine auxiliary storage exceeded

000D Machine status
0101 Machine check

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Invalid branch target				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		

CONVERT CHARACTER TO HEX (CVTCH)

Op Code (Hex)	Operand 1	Operand 2
1082	Receiver	Source

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Description: Each character (8-bit value) of the string value in the source operand is converted to a hex digit (4-bit value) and placed in the receiver operand. The source operand characters must relate to valid hex digits or a conversion exception is signaled.

Characters		Hex Digits
Hex F0-hex F9	=	Hex 0-hex 9
Hex C1-hex C6	=	Hex A-hex F

The operation begins with the two operands left-adjusted and proceeds left to right until all the hex digits of the receiver operand have been filled. If the source operand is too small, it is logically padded on the right with zero characters (hex F0). If the source operand is too large, a length conformance or an invalid operand length exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with a value of hex 00. The effect of specifying a null substring reference for the receiver is that no result is set.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computation			
01 Conversion		X	
08 Length Conformance	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

CONVERT CHARACTER TO MRJE (CVTCM)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
108B	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(13) variable scalar (fixed-length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTCMI	188B	Indicator
CVTCMB	1C8B	Branch

Extender: Branch options or indicator options.

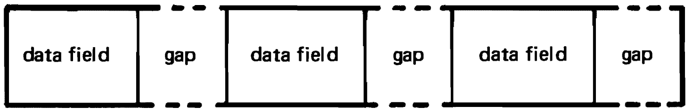
If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a string of characters to MRJE (MULTI-LEAVING remote job entry) compressed format. The operation converts the source (operand 3) from character to the MRJE compressed format under control of the controls (operand 2) and places the results in the receiver (operand 1).

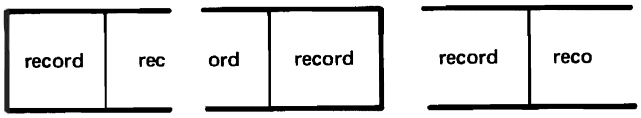
The source and receiver operands must both be character strings. The source operand cannot be specified as either a signed or unsigned immediate value.

The source operand can be described through the controls operand as being composed of one or more fixed length data fields, which may be separated by fixed length gaps of characters to be ignored during the conversion operation. Additionally, the controls operand specifies the amount of data to be processed from the source to produce a converted record in the receiver. This may be a different value than the length of the data fields in the source. The following diagram shows this structure for the source operand.

Actual Source Operand Bytes



Data Process as Source Records



The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 13 bytes in length and have the following format:

• Controls operand	Char(13)
– Offset into the receiver operand	Bin(2)
– Offset into the source operand	Bin(2)
– Algorithm modifier	Char(1)
– Source record length	Char(1)
– Data field length	Bin(2)
– Offset to next gap in source operand	Bin(2)
– Gap length	Bin(2)
– Record control block (RCB) value	Char(1)

As input to the instruction, the source and receiver offset fields specify the offsets where bytes of the source and receiver operands are to be processed. If an offset is equal to or greater than the length specified for the operand it corresponds to (it identifies a byte beyond the end of the operand), a template value invalid exception is signaled.

As output from the instruction, the source and receiver offset fields specify offsets that indicate how much of the operation is complete when the instruction ends.

The algorithm modifier has the following valid values:

- Hex 00 = Perform full compression.
- Hex 01 = Perform only truncation of trailing blanks.

The source record length value specifies the amount of data from the source to be processed. If a source record length of 0 is specified, a template value invalid exception is signaled.

The data field length value specifies the length of the data fields in the source. Data fields occurring in the source may be separated by gaps of characters, which are to be ignored during the conversion operation. Specification of a data field length of 0 indicates that the source operand is one data field. In this case, the gap length and gap offset values have no meaning and are ignored.

The gap offset value specifies the offset to the next gap in the source. This value is both input to and output from the instruction. This is relative to the current byte to be processed in the source as located by the source offset value. No validation is done for this offset. It is assumed to be valid relative to the source operand. The gap offset value is ignored if the data field length is specified with a value of 0.

The gap length value specifies the amount of data occurring between data fields in the source operand which is to be ignored during the conversion operation. The gap length value is ignored if the data field length is specified with a value of 0.

The record control block (RCB) field specifies the RCB value that is to precede the converted form of each record in the receiver. It can have any value.

Only the first 13 bytes of the controls operand are used. Any excess bytes are ignored.

The operation begins by accessing the bytes of the source operand at the location specified by the source offset. This is assumed to be the start of a source record. Only the bytes of the data fields in the source are accessed for conversion purposes. Gaps between data fields are ignored, causing the access of data field bytes to occur as if the data fields were contiguous with one another. Bytes accessed from the source for the source record length are considered a source record for the conversion operation. They are converted into the receiver operand at the location specified by the receiver offset according to the following algorithm.

The RCB value is placed into the first byte of the receiver record.

An SRCB (sub record control byte) value of hex 80 is placed into the second byte of the receiver record.

If the algorithm modifier specifies full compression (a value of hex 00) then:

> The bytes of the source record are interrogated to locate the blank character strings (2 or more consecutive blanks), identical character strings (3 or more consecutive identical characters), and nonidentical character strings occurring in the source. A blank character string occurring at the end of the record is treated as a special case (see following information on trailing blanks).

If the algorithm modifier specifies blank truncation (a value of hex 01) then:

> The bytes of the source record are interrogated to determine if a blank character string exists at the end of the source record. If one exists, it is treated as a string of trailing blanks. All characters prior to it in the record are treated as one string of nonidentical characters.

The strings encountered (blank, identical, or nonidentical) are reflected in the receiver by building one or more SCBs (string control bytes) in the receiver to describe them.

The format of the SCBs built into the receiver is:

- SCB format is o k l jjjjj

The bit meanings are:

Bit	Value	Meaning
o	0	End of record; the EOR SCB is hex 00.
	1	All other SCBs.
k	0	The string is compressed.
	1	The string is not compressed.
1		For k = 0:
	0	Blanks (hex 40s) have been deleted.
	1	Nonblank characters have been deleted. The next character in the data stream is the specimen character.
		For k = 1: This bit is part of the length field for length of uncompressed data.
jjjjj		Number of characters that have been deleted if k = 0. The value can be 2-31.
1jjjjj		Number of characters to the next SCB (no compression) if k = 1. The value can be 1-63. The uncompressed (nonidentical bytes) follow the SCB in the data stream.

When the end of a source record is encountered, an EOR (end of record) SCB (hex 00) is built into the receiver. Trailing blanks in a record including a record of all blanks are represented in the receiver by an EOR character if either full compression or trailing blank truncation is specified.

If the end of the source operand is not encountered, the operation then continues by reapplying the above algorithm to the next record in the source operand.

If the end of the source operand is encountered (whether or not in conjunction with a record boundary), the instruction ends with a resultant condition of *source exhausted*. The offset value for the source locates the byte following the last source record for which conversion was completed. The gap offset value indicates the offset to the next gap relative to the source offset value set for this condition. The gap offset value has no meaning and is not set when the data field length is 0. The offset value for the receiver locates the byte following the last fully converted record in the receiver. The value of the remaining bytes in the receiver after the last converted record is unpredictable.

If the converted form of a record cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. The offset value for the source locates the byte following the last source record for which conversion was completed. The gap offset value indicates the offset to the next gap relative to the source offset value set for this condition. The gap offset value has no meaning and is not set when the data field length is 0. The offset value for the receiver locates the byte following the last fully converted record in the receiver. The value of the remaining bytes in the receiver after the last converted record is unpredictable.

Any form of overlap between the operands of this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions

Source exhausted–All complete records in the source operand have been converted into the receiver operand.
Receiver overrun–An overrun condition in the receiver operand was detected prior to processing all of the bytes in the source operand.

If source exhausted and receiver overrun occur at the same time, the source exhausted condition is recognized first. When source exhausted is the resultant condition, the receiver may also be full. In this case, the offset into the receiver may contain a value equal to the length specified for the receiver, and this condition will cause an exception on the next invocation of the instruction. The processing performed for the source exhausted condition provides for this case when the instruction is invoked multiple times with the same controls operand template. When the receiver overrun condition is the resultant condition, the source always contains data that can be converted.

Events

000C Machine resources
- 0201 Machine auxiliary storage exceeded

000D Machine status
- 0101 Machine check

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference
- 0201 Object location reference

0017 Damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Invalid branch target				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		

CONVERT CHARACTER TO NUMERIC (CVTCN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1083	Receiver	Source	Attributes

Operand 1: Numeric variable scalar or data-pointer-defined numeric scalar.

Operand 2: Character scalar or data-pointer-defined character scalar.

Operand 3: Character(7) scalar or data-pointer-defined character scalar.

Description: The character scalar specified by operand 2 is treated as though it were a numeric scalar with the attributes specified by operand 3. The character string source operand is converted to the numeric forms of the receiver operand and moved to the receiver operand. The value of operand 2, when viewed in this manner, is converted to the type, length, and precision of the numeric receiver, operand 1, following the rules for the Copy Numeric Value instruction.

The length of operand 2 must be large enough to contain the numeric value described by operand 3. If it is not large enough, a scalar value invalid exception is signaled. If it is larger than needed, its leftmost bytes are used as the value, and the rightmost bytes are ignored.

Normal rules of arithmetic conversion apply except for the following. If operand 2 is interpreted as a zoned decimal value, a value of hex 40 in the rightmost byte referenced in the conversion is treated as a positive sign and a zero digit.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

The format of the attribute operand specified by operand 3 is as follows:

• Scalar attributes	Char(7)
– Scalar type	Char(1)
Hex 00 = Binary	
Hex 01 = Floating-point	
Hex 02 = Zoned decimal	
Hex 03 = Packed decimal	
– Scalar length	Bin(2)
If binary:	
Length (L)	Bits 0-15
(where L = 2 or 4)	
If floating-point:	
Length (L)	Bits 0-15
(where L = 4 or 8)	
If zoned decimal or packed decimal:	
Fractional digits (F)	Bits 0-7
Total digits (T) (where 1 ≤ T ≤ 31 and 0 ≤ F ≤ T)	Bits 8-15
– Reserved (binary 0)	Bin(4)

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	**Operands** 1	2	3	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
04 External data object not found	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
06 Floating-point overflow	X			
07 Floating-point underflow	X			
09 Floating-point invalid operand		X		
0A Size	X			
0C Floating-point conversion	X			
0D Floating-point inexact result	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length		X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attribute invalid			X	
03 Scalar value invalid			X	

CONVERT CHARACTER TO SNA (CVTCS)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10CB	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(15) variable scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTCSI	18CB	Indicator
CVTCSB	1CCB	Branch

Extender: Branch options or indicator options.

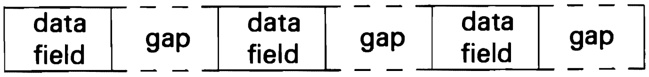
If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts the source (operand 3) from character to SNA (systems network architecture) format under control of the controls (operand 2) and places the result into the receiver (operand 1).

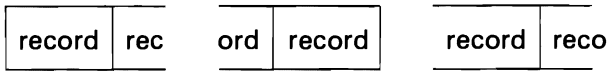
The source and receiver operands must both be character strings. The source operand may not be specified as an immediate operand.

The source operand can be described by the controls operand as being one or more fixed-length data fields that may be separated by fixed-length gaps of characters to be ignored during the conversion operation. Additionally, the controls operand specifies the amount of data to be processed from the source to produce a converted record in the receiver. This may be a different value than the length of the data fields in the source. The following diagram shows this structure for the source operand.

Actual source operand bytes



Data processed as source records



The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. The operand must be at least 15 bytes in length and has the following format:

• Controls operand	Char(15)
– Offset into the receiver operand	Bin(2)
– Offset into the source operand	Bin(2)
– Algorithm modifier	Char(1)
– Source record length	Char(1)
– Data field length	Bin(2)
– Gap offset	Bin(2)
– Gap length	Bin(2)
– Record separator character	Char(1)
– Prime compression character	Char(1)
– Unconverted source record bytes	Char(1)

When the source and receiver operands are input to the instruction, they specify the offsets where the bytes of the source and receiver operands are to be processed. If an offset is equal to or greater than the length specified for the operand, the offset identifies a byte beyond the end of the operand and a template value invalid exception is signaled. When the source and the receiver are output from the instruction, they specify offsets that indicate how much of the operation is complete when the instruction ends.

The algorithm modifier specifies the optional functions to be performed. Any combination of functions can be specified as indicated by the bit meanings in the following chart. At least one of the functions must be specified. If all of the algorithm modifier bits are zero, a template value invalid exception is signaled. The algorithm modifier bit meanings are:

Bits	Meaning	
0	0 =	Do not perform compression.
	1 =	Perform compression.
1-2	00 =	Do not use record separators and no blank truncation. Do not perform data transparency conversion.
	01 =	Reserved.
	10 =	Use record separators and perform blank truncation. Do not perform data transparency conversion.
	11 =	Use record separators and perform blank truncation. Perform data transparency conversion.
3	0 =	Do not perform record spanning.
	1 =	Perform record spanning. (allowed only when bit 1 = 1)
4-7	(Reserved)	

The source record length value specifies the amount of data from the source to be processed to produce a converted record in the receiver. Specification of a source record length of zero results in a template value invalid exception.

The data field length value specifies the length of the data fields in the source. Data fields occurring in the source may be separated by gaps of characters that are to be ignored during the conversion operation. Specification of a data field length of zero indicates that the source operand is one data field. In this case, the gap length and gap offset values have no meaning and are ignored.

The gap offset value specifies the offset to the next gap in the source. This value is both input to and output from the instruction. This is relative to the current byte to be processed in the source as located by the source offset value. No validation is done for this offset. It is assumed to be valid relative to the source operand. The gap offset value is ignored if the data field length is specified with a value of zero.

The gap length value specifies the amount of data that is to be ignored between data fields in the source operand during the conversion operation. The gap length value is ignored if the data field length is zero.

The record separator character value specifies the character that precedes the converted form of each record in the receiver. It also serves as a delimiter when the previous record is truncating trailing blanks. The Convert SNA to Character instruction recognizes any value that is less than hex 40. The record separator value is ignored if do not use record separators is specified in the algorithm modifier.

The prime compression character value specifies the character to be used as the prime compression character when performing compression of the source data to SNA format. It may have any value. The prime compression character value is ignored if the compression function is not specified in the algorithm modifier.

The unconverted source record bytes value specifies the number of bytes remaining in the current source record that are yet to be converted.

When the record spanning function is specified in the algorithm modifier, the unconverted source record bytes value is both input to and output from the instruction. On input, a value of hex 00 means it is the start of a new record and the initial conversion step is yet to be performed. That is, a record separator character has not yet been placed in the receiver. On input, a nonzero value less than or equal to the record length specifies the number of bytes remaining in the current source record that are yet to be converted into the receiver. This value is assumed to be the valid count of unconverted source record bytes relative to the current byte to be processed in the source as located by the source offset value. As such, it is used to determine the location of the next record boundary in the source operand. This value must be less than or equal to the source record length value; otherwise, a template value invalid exception is signaled. On output this field is set with a value as defined above that describes the number of bytes of the current source record that have not yet been converted.

When the record spanning function is not specified in the algorithm modifier, the unconverted source record bytes value is ignored.

Only the first 15 bytes of the controls operand are used. Any excess bytes are ignored.

The description of the conversion process is presented as a series of separately performed steps that may be selected in allowable combinations to accomplish the conversion function. It is presented this way to allow for describing these functions separately. However, in the actual execution of the instruction, these functions may be performed in conjunction with one another or separately depending upon which technique is determined to provide the best implementation.

The operation is performed either on a record-by-record basis, record processing, or on a nonrecord basis, string processing. This is determined by the functions selected in the algorithm modifier. Specifying the use record separators and do blank truncation function indicates record processing is to be performed. If this is not specified, in which case compression must be specified, it indicates that string processing is to be performed.

The operation begins by accessing the bytes of the source operand at the location specified by the source offset.

When record processing is specified, the source offset may locate the start of a full or partial record.

When the record spanning function has not been specified in the algorithm modifier, the source offset is assumed to locate the start of a record.

When the record spanning function has been specified in the algorithm modifier, the source offset is assumed to locate a point at which processing of a possible partially converted record is to be resumed. In this case the unconverted source record bytes value contains the length of the remaining portion of the source record to be converted. The conversion process in this case is started by completing the conversion of the current source record before processing the next full source record.

When string processing is specified, the source offset locates the start of the source string to be converted.

Only the bytes of the data fields in the source are accessed for conversion purposes. Gaps between data fields are ignored causing the access of data field bytes to occur as if the data fields were contiguous. A string of bytes accessed from the source for a length equal to the source record length is considered to be a record for the conversion operation.

When during the conversion process the end of the source operation is encountered, the instruction ends with a resultant condition of source exhausted.

When record processing is specified in the algorithm modifier, this check is performed at the start of conversion for each record. If the source operand does not contain a full record, the source exhausted condition is recognized. The instruction is terminated with status in the controls operand describing the last completely converted record. For source exhausted, partial conversion of a source record is not performed.

When string processing is specified in the algorithm modifier, then compression must be specified and the compression function described below defines the detection of source exhausted.

If the converted form of the source cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. See the description of this condition in the conversion process described below to determine the status of the controls operand values and the converted bytes in the receiver for each case.

When string processing is specified, the bytes accessed from the source are converted on a string basis into the receiver operand at the location specified by the receiver offset. In this case, the compression function must be specified and the conversion process proceeds with the compression function defined below.

When record processing is specified, the bytes accessed from the source are converted one record at a time into the receiver operand at the location specified by the receiver offset performing the functions specified in the algorithm modifier in the sequence defined by the following algorithm.

The first function performed is trailing blank truncation.

A truncated record is built by logically appending the record data to the record separator value specified in the controls operand and removing all blank characters after the last nonblank character in the record. If a record has no trailing blanks, then no actual truncation takes place. A null record, a record consisting entirely of blanks, will be converted as just the record separator character with no other data following it. The truncated record then consists of the record separator character followed by the truncated record data, the full record data, or no data from the record.

If either the data transparency conversion or the compression function is specified in the algorithm modifier, the conversion process continues for this record with the next specified function.

If not, the conversion process for this record is completed by placing the truncated record into the receiver. If the truncated record cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. When the record spanning function is specified in the algorithm modifier, as much of the truncated record as will fit is placed into the receiver and the controls operand is updated to describe how much of the source record was successfully converted into the receiver. When the record spanning function is not specified in the algorithm modifier, the controls operand is updated to describe only the last fully converted record in the receiver and the value of the remaining bytes in the receiver is unpredictable.

The second function performed is data transparency conversion.

Data transparency conversion is performed if the function is specified in the algorithm modifier. This provides for making the data in a record transparent to the Convert SNA to Character instruction in the area of its scanning for record separator values. Transparent data is built by preceding the data with 2 bytes of transparency control information. The first byte has a fixed value of hex 35 and is referred to as the TRN (transparency) control character. The second byte is a 1-byte hexadecimal count, a value ranging from 1 to 255 decimal, of the number of bytes of data that follow and is referred to as the TRN count. This contains the length of the data and does not include the TRN control information length.

Transparency conversion can be specified only in conjunction with record processing and, as such, is performed on the truncated form of the source record. The transparent record is built by preceding the data that follows the record separator in the truncated record with the TRN control information. The TRN count in this case contains the length of just the truncated data for the record and does not include the record separator character. For the special case of a null record, no TRN control information is placed after the record separator character because there is no record data to be made transparent.

If the compression function is specified in the algorithm modifier, the conversion process continues for this record with the compression function.

If not, the conversion process for this record is completed by placing the transparent record into the receiver. If the transparent record cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun.

When the record spanning function is specified in the algorithm modifier, as much of the transparent record as will fit is placed into the receiver and the controls operand is updated to describe how much of the source record was successfully converted into the receiver. The TRN count is also adjusted to describe the length of the data successfully converted into the receiver; thus, the transparent data for the record is not spanned out of the receiver. The remaining bytes of the transparent record, if any, will be processed as a partial source record on the next invocation of the instruction and will be preceded by the appropriate TRN control information. For the special case where only 1 to 3 bytes are available at the end of the receiver, (not enough room for the record separator, the transparency control, and a byte of data) then just the record separator is placed in the receiver for the record being converted. This can cause up to 2 bytes of unused space at the end of the receiver. The value of these unused bytes is unpredictable.

When the record spanning function is not specified in the algorithm modifier, the controls operand is updated to describe only the last fully converted record in the receiver and the value of the remaining bytes in the receiver is unpredictable.

The third function performed is compression.

Compression is performed if the function is specified in the algorithm modifier. This provides for reducing the size of strings of duplicate characters in the source data. The source data to be compressed may have assumed a partially converted form at this point as a result of processing for functions specified in the algorithm modifier. Compressed data is built by concatenating one or more compression strings together to describe the bytes that make up the converted form of the source data prior to the compression step. The bytes of the converted source data are interrogated to locate the prime compression character strings (two or more consecutive prime compression characters), duplicate character strings (three or more duplicate nonprime characters) and nonduplicate character strings occurring in the source.

The character strings encountered (prime, duplicate and nonduplicate) are reflected in the compressed data by building one or more compression strings to describe them. Compression strings are comprised of an SCB (string control byte) possibly followed by one or more bytes of data related to the character string to be described.

The format of an SCB and the description of the data that may follow it are:

- SCB Char(1)
  - Control Bits 0-1

    00 = n nonduplicate characters are between this SCB and the next one; where n is the value of the count field (1-63).

    01 = Reserved

    10 = This SCB represents n deleted prime compression characters; where n is the value of the count field (2-63). The next byte is the next SCB.

    11 = This SCB represents n deleted duplicate characters; where n is the value of the count field (3-63). The next byte contains a specimen of the deleted characters. The byte following the specimen character contains the next SCB.
  - Count Bits 2-7

    This contains the number of characters that have been deleted for a prime or duplicate string, or the number of characters to the next SCB for a nonduplicate string. A count value of zero cannot be produced.

When record processing is specified, the compression is performed as follows.

The compression function is performed on just the converted form of the current source record including the record separator character. The converted form of the source record prior to the compression step may be a truncated record or a transparent record as described above, depending upon the functions selected in the algorithm modifier. The record separator and TRN control information is always converted as a nondupIicate compression entry to provide for length adjustment of the TRN count, if necessary.

The conversion process for this record is completed by placing the compressed record into the receiver. If the compressed record cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun.

When the record spanning function is specified in the algorithm modifier, as much of the compressed record as will fit is placed into the receiver and the controls operand is updated to describe how much of the source record was successfully converted into the receiver. The last compression entry placed into the receiver may be adjusted if necessary to a length that provides for filling out the receiver. This length adjustment applies only to compression entries for nonduplicate strings. Compression entries for duplicate strings are placed in the receiver only if they fit with no adjustment. For the special case where data transparency conversion is specified, the transparent data being described is not spanned out of the receiver. This is provided for by performing length adjustment on the TRN count of a transparent record, which may be included in the compressed data so that it describes only the source data that was successfully converted into the receiver. For the special case where only 2 to 5 bytes are available at the end of the receiver, not enough room for the compression entry for a nonduplicate string containing the record separator and the TRN control, and up to a 2-byte compression entry for some of the transparent data, the nonduplicate compression entry is adjusted to describe only the record separator. By doing this, no more than 3 bytes of unused space will remain in the receiver. The value of these unused bytes is unpredictable. Unconverted source record bytes, if any, will be processed as a partial source record on the next invocation of the instruction and will be preceded by the appropriate TRN control information when performing transparency conversion.

When the record spanning function is not specified in the algorithm modifier, the controls operand is updated to describe only the last fully converted record in the receiver. The value of the remaining bytes in the receiver is unpredictable.

When string processing is specified, the compression is performed as follows.

The compression function is performed on the data for the entire source operand on a compression string basis. In this case, the fields in the controls operand related to record processing are ignored.

The conversion process for the source operand is completed by placing the compressed data into the receiver.

When the compressed data cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. As much of the compressed data as will fit is placed into the receiver and the controls operand is updated to describe how much of the source data was successfully converted into the receiver. The last compression entry placed into the receiver may be adjusted if necessary to a length that provides for filling out the receiver. This length adjustment applies only to compression entries for nonduplicate strings. Compression entries for duplicate strings are placed in the receiver only if they fit with no adjustment. By doing this, no more than 1 byte of unused space will remain in the receiver.

When the compressed data can be completely contained in the receiver, the instruction ends with a resultant condition of source exhausted. The compressed data is placed into the receiver and the controls operand is updated to indicate that all of the source data was successfully converted into the receiver.

At this point, either conversion of a source record has been completed or conversion has been interrupted due to detection of the source exhausted or receiver overrun conditions. For record processing, if neither of the above conditions has been detected either during conversion of or at completion of conversion for the current record, the conversion process continues on the next source record with the blank truncation step described above.

At completion of the instruction, the offset value for the receiver locates the byte following the last converted byte in the receiver. The value of the remaining bytes in the receiver after the last converted byte are unpredictable. The offset value for the source locates the byte following the last source byte for which conversion was completed. When the record spanning function is specified in the algorithm modifier, the unconverted source record bytes value specifies the length of the remaining source record bytes yet to be converted. When the record spanning function is not specified in the algorithm modifier, the unconverted source record bytes value has no meaning and is not set. The gap offset value indicates the offset to the next gap relative to the source offset value set for this condition. The gap offset value has no meaning and is not set when the data field length is zero.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Resultant Conditions: Source exhausted–All bytes in the source operand have been converted into the receiver operand. Receiver overrun–An overrun condition in the receiver operand was detected before all of the bytes in the source operand were processed.

Programming Notes:

If the source operand does not end on a record boundary, in which case the last record is spanned out of the source, this instruction performs conversion only up to the start of that partial record. In this case, the user of the instruction must move this partial record to combine it with the rest of the record in the source operand to provide for its being processed correctly upon the next invocation of the instruction. If full records are provided, the instruction performs its conversions out to the end of the source operand and no special processing is required.

For the special case of a tie between the source exhausted and receiver overrun conditions, the source exhausted condition is recognized first. That is, when source exhausted is the resultant condition, the receiver may also be full. In this case, the offset into the receiver operand may contain a value equal to the length specified for the receiver, which would cause an exception to be detected on the next invocation of the instruction. The processing performed for the source exhausted condition should provide for this case if the instruction is to be invoked multiple times with the same controls operand template. When the receiver overrun condition is the resultant condition, the source will always contain data that can be converted.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

000D Machine status
- 0101 Machine check

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op-code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
2C Program Execution				
04 Invalid branch target				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		

CONVERT DECIMAL FORM TO FLOATING-POINT (CVTDFFP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
107F	Receiver	Decimal exponent	Decimal significand

Operand 1: Floating-point variable scalar.

Operand 2: Packed scalar or zoned scalar.

Operand 3: Packed scalar or zoned scalar.

Description: This instruction converts the decimal form of a floating-point value specified by a decimal exponent and a decimal significand to binary floating-point format, and places the result in the receiver operand. The decimal exponent (operand 2) and decimal significand (operand 3) are considered to specify a decimal form of a floating-point number. The value of this number is considered to be as follows:

Value = S * (10**E)

where: S = The value of the decimal significand operand.
E = The value of the decimal exponent operand.
* Denotes multiplication.
** Denotes exponentiation.

The decimal exponent must be specified as a decimal integer value; no fractional digit positions may be specified in its definition. The decimal exponent is a signed integer value specifying a power of 10 which gives the floating-point value its magnitude. A decimal exponent value too large or too small to be represented in the receiver will result in the detection of the appropriate floating-point overflow or floating-point underflow exception.

The decimal significand must be specified as a decimal value with a single integer digit position and optional fractional digit positions. The decimal significand is a signed decimal value specifying decimal digits which give the floating-point value its precision. The significant digits of the decimal significand are considered to start with the leftmost nonzero decimal digit and continue to the right to the end of the decimal significand value. Significant digits beyond 7 for a short float receiver, and beyond 15 for a long float receiver exceed the precision provided for in the binary floating-point receiver. These excess digits do participate in the conversion to provide for uniqueness of the conversion as well as for proper rounding.

The decimal form floating-point value specified by the decimal exponent and decimal significand operands is converted to a binary floating-point number and rounded to the precision of the result field as follows:

Source values which, in magnitude M, are in the range where (10**31-1) * 10**-31 <= M <= (10**31-1) *10**+31 are converted subject to the normal rounding error defined for the floating-point rounding modes.

Source values which, in magnitude M, are in the range where (10**31-1) * 10**-31 > M > (10**31-1) *10**+31 are converted such that the rounding error incurred on the conversion may exceed that defined above. For round to nearest, this error will not exceed by more than .47 units in the least significant digit position of the result in relation to the error that would be incurred for normal rounding. For the other floating-point rounding modes, this error will not exceed 1.47 units in the least significant digit position of the result.

The converted and rounded value is then assigned to the floating-point receiver.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
06 Floating-point overflow	X			
07 Floating-point underflow	X			
0D Floating-point inexact result	X			
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op-code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	

CONVERT EXTERNAL FORM TO NUMERIC VALUE (CVTEFN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1087	Receiver	Source	Mask

Operand 1: Numeric variable scalar or data-pointer-defined numeric scalar.

Operand 2: Character scalar or data-pointer-defined character scalar.

Operand 3: Character(3) scalar, null, or data-pointer-defined character(3) scalar.

Description: This instruction scans a character string for a valid decimal number in display format, removes the display character, and places the results in the receiver operand. The operation begins by scanning the character string value in the source operand to make sure it is a valid decimal number in display format.

The character string defined by operand 2 consists of the following optional entries:

- Currency symbol–This value is optional and, if present, must precede any sign and digit values. The valid symbol is determined by operand 3. The currency symbol may be preceded in the field by blank (hex 40) characters.

- Sign symbol–This value is optional and, if present, may precede any digit values (a leading sign) or may follow the digit values (a trailing sign). Valid signs are positive (hex 4E) and negative (hex 60). The sign symbol, if it is a leading sign, may be preceded by blank characters. If the sign symbol is a trailing sign, it must be the rightmost character in the field. Only one sign symbol is allowed.

- Decimal digits–Up to 31 decimal digits may be specified. Valid decimal digits are in the range of hex F0 through hex F9 (0-9). The first decimal digit may be preceded by blank characters (hex 40), but hex 40 values located to the right of the leftmost decimal digit are invalid.

The decimal digits may be divided into two parts by the decimal point symbol: an integer part and a fractional part. Digits to the left of the decimal point are interpreted as integer values. Digits to the right are interpreted as a fractional values. If no decimal point symbol is included, the value is interpreted as an integer value. The valid decimal point symbol is determined by operand 3. If the decimal point symbol precedes the leftmost decimal digit, the digit value is interpreted as a fractional value, and the leftmost decimal digit must be adjacent to the decimal point symbol. If the decimal point follows the rightmost decimal digit, the digit value is interpreted as an integer value, and the rightmost decimal digit must be adjacent to the decimal point.

Decimal digits in the integer portion may optionally have comma symbols separating groups of three digits. The leftmost group may contain one, two, or three decimal digits, and each succeeding group must be preceded by the comma symbol and contain three digits. The comma symbol must be adjacent to a decimal digit on either side. The valid comma symbol is determined by operand 3.

Decimal digits in the fractional portion may not be separated by commas and must be adjacent to one another.

Examples of external formats follow. The following symbols are used.

$	–	currency symbol
.	–	decimal point
,	–	comma
D	–	digit (hex F0-hex F9)
ƀ	–	blank (hex 40)
+	–	positive sign
-	–	negative sign

Format	**Comments**
$+DDDD.DD	Currency symbol, leading sign, no comma separators
DD,DDD-	Comma symbol, no fraction, trailing sign
-.DDD	No integer, leading sign
$DDD,DDD-	No fraction, comma symbol, trailing sign
ƀ$ƀ+ƀDD.DD	Embedded blanks before digits

Operand 3 must be a 3-byte character scalar. Byte 1 of the string indicates the byte value that is to be used for the currency symbol. Byte 2 of the string indicates the byte value to be used for the comma symbol. Byte 3 of the string indicates the byte value to be used for the decimal point symbol. If operand 3 is null, the currency symbol (hex 5B), comma (hex 6B), and decimal point (hex 4B) are used.

If the syntax rules are violated, a conversion exception is signaled. If not, a zoned decimal value is formed from the digits of the display format character string. This number is placed in the receiver operand following the rules of a normal arithmetic conversion.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands			Other
	1	2	3	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
01 Conversion		X		
0A Size	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length		X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attribute invalid			X	

CONVERT FLOATING-POINT TO DECIMAL FORM (CVTFPDF)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10BF	Decimal exponent	Decimal significand	Source

Operand 1: Packed variable scalar or zoned variable scalar.

Operand 2: Packed variable scalar or zoned variable scalar.

Operand 3: Floating-point scalar.

Optional Form

Mnemonic	Op Code (Hex)	Form Type
CVTFPDFR	12BF	Round

Description: This instruction converts a binary floating-point value to a decimal form of a floating-point value specified by a decimal exponent and a decimal significand, and places the result in the decimal exponent and decimal significand operands.

The value of this number is considered to be as follows:

Value = S * (10**E)

where: S = The value of the decimal significand operand.
E = The value of the decimal exponent operand.
* Denotes multiplication.
** Denotes exponentiation.

The decimal exponent must be specified as a decimal integer value. No fractional digit positions are allowed. It must be specified with at least five digit positions. The decimal exponent provides for containing a signed integer value specifying a power of 10 which gives the floating-point value its magnitude.

The decimal significand must be specified as a decimal value with a single integer digit position and optional fractional digit positions. The decimal significand provides for containing a signed decimal value specifying decimal digit is which give the floating-point value its precision. The decimal significand is formed as a normalized value, that is, the leftmost digit position is nonzero for a nonzero source value.

When the source contains a representation of a normalized binary floating-point number with decimal significand digits beyond the leftmost 7 digits for a short floating-point source or beyond the leftmost 15 digits for a long floating-point source, the precision allowed for the binary floating-point source is exceeded.

When the source contains a representation of a denormalized binary floating-point number, it may provide less precision than the precision of a normalized binary floating-point number, depending on the particular source value. Decimal significand digits exceeding the precision of the source are set as a result of the conversion to provide for uniqueness of conversion and are correct, except for rounding errors. These digits are only as precise as the floating-point calculations that produced the source value. The floating-point inexact result exception provides a means of detecting loss of precision in floating-point calculations.

The binary floating-point source is converted to a decimal form floating-point value and rounded to the precision of the decimal significand operand as follows:

- The decimal significand is formed as a normalized value and the decimal exponent is set accordingly.

- For the nonround form of the instruction, the value to be assigned to the decimal significand is adjusted to the precision of the decimal significand, if necessary, according to the current float rounding mode in effect for the process. For the optional round form of the instruction, the decimal round algorithm is used for the precision adjustment of the decimal significand. The decimal round algorithm overrides the current floating-point rounding mode that is in effect for the process.

- Source values which, in magnitude M, are in the range where (10**31-1) * 10**-31 <= M <= (10**31-1) * 10**+31 are converted subject to the normal rounding error defined for the floating-point rounding modes and the optional round form of the instruction.

- Source values which, in magnitude M, are in the range where (10**31-1) * 10**-31 > M > (10**31-1) * 10**+31 are converted such that the rounding error incurred on the conversion may exceed that defined above. For round to nearest and the optional round form of the instruction, this error will not exceed by more than .47 units in the least significant digit position of the result, the error that would be incurred for a correctly rounded result. For the other floating-point rounding modes, this error will not exceed 1.47 units in the least significant digit position of the result.

- If necessary, the decimal exponent value is adjusted to compensate for rounding.

- The converted and rounded value is then assigned to the decimal exponent and decimal significand operands.

A size exception cannot occur on the assignment of the decimal exponent or the decimal significand values.

The result of the operation is unpredictable for any type of overlap between the decimal exponent and decimal significand operands.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands			Other
	1	2	3	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
0C Invalid floating-point conversion	X	X		
0D Floating-point inexact result		X		
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op-code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	

CONVERT HEX TO CHARACTER (CVTHC)

Op Code (Hex)	Operand 1	Operand 2
1086	Receiver	Source

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Description: Each hex digit (4-bit value) of the string value in the source operand is converted to a character (8-bit value) and placed in the receiver operand.

Hex Digits		Characters
Hex 0-9	=	Hex F0-F9
Hex A-F	=	Hex C1-C6

The operation begins with the two operands left-adjusted and proceeds left to right until all the characters of the receiver operand have been filled. If the source operand contains fewer hex digits than needed to fill the receiver, the excess characters are assigned a value of hex F0. If the source operand is too large, a length conformance or an invalid operand length exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with a value of hex F0. The effect of specifying a null substring reference for the receiver is that no result is set.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computation			
08 Length conformance		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

CONVERT MRJE TO CHARACTER (CVTMC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10AB	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(6) variable scalar (fixed-length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTMCI	18AB	Indicator
CVTMCB	1CAB	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a character string from the MRJE (MULTI-LEAVING remote job entry) compressed format to character format. The operation converts the source (operand 3) from the MRJE compressed format to character format under control of the controls (operand 2) and places the results in the receiver (operand 1).

The source and receiver operands must both be character strings. The source operand cannot be specified as either a signed or unsigned immediate value.

The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 6 bytes in length and have the following format:

• Controls operand	Char(6)
– Offset into the receiver operand	Bin(2)
– Offset into the source operand	Bin(2)
– Algorithm modifier	Char(1)
– Receiver record length	Char(1)

As input to the instruction, the source and receiver offset fields specify the offsets where bytes of the source and receiver operands are to be processed. If an offset is equal to or greater than the length specified for the operand it corresponds to (it identifies a byte beyond the end of the operand), a template value invalid exception is signaled. As output from the instruction, the source and receiver offset fields specify offsets that indicate how much of the operation is complete when the instruction ends.

The algorithm modifier has the following valid values:

- Hex 00 = Do not move SRCBs (sub record control bytes) from the source into the receiver.

- Hex 01 = Move SRCBs from the source into the receiver.

The receiver record length value specifies the record length to be used to convert source records into the receiver operand. This length applies to only the string portion of the receiver record and does not include the optional SRCB field. If a receiver record length of 0 is specified, a template value invalid exception is signaled.

Only the first 6 bytes of the controls operand are used. Any excess bytes are ignored.

The operation begins by accessing the bytes of the source operand at the location specified by the source offset. This is assumed to be the start of a record. The bytes of the records in the source operand are converted into the receiver operand at the location specified by the receiver offset according to the following algorithm.

The first byte of the source record is considered to be an RCB (record control byte) that is to be ignored during conversion.

The second byte of the source record is considered to be an SRCB. If an algorithm modifier of value hex 00 was specified, the SRCB is ignored. If an algorithm modifier of value hex 01 was specified, the SRCB is copied into the receiver.

The strings to be built in the receiver record are described in the source after the SRCB by one or more SCBs (string control bytes).

The format of the SCBs in the source are as follows:

- SCB format is o k 1 jjjjj

The bit meanings are:

Bit	Value	Meaning
o	0	End of record; the EOR SCB is hex 00.
	1	All other SCBs.
k	0	The string is compressed.
	1	The string is not compressed.
1		For k = 0:
	0	Blanks (hex 40s) have been deleted.
	1	Nonblank characters have been deleted. The next character in the data stream is the specimen character.
		For k = 1: This bit is part of the length field for length of uncompressed data.
jjjjj		Number of characters that have been deleted if k = 0. The value can be 1-31.
1jjjjj		Number of characters to the next SCB (no compression) if k = 1. The value can be 1-63. The uncompressed (nonidentical bytes) follow the SCB in the data stream.

A length of 0 encountered in an SCB results in the signaling of a conversion exception.

Strings of blanks or nonblank identical characters described in the source record are repeated in the receiver the number of times indicated by the SCB count value.

Strings of nonidentical characters described in the source record are moved into the receiver for the length indicated by the SCB count value.

When an EOR (end of record) SCB (hex 00) is encountered in the source, the receiver is padded with blanks out to the end of the current record.

If the converted form of a source record is larger than the receiver record length, the instruction is terminated by signaling a length conformance exception.

If the end of the source operand is not encountered, the operation then continues by reapplying the above algorithm to the next record in the source operand.

If the end of the source operand is encountered (whether or not in conjunction with a record boundary, EOR SCB in the source), the instruction ends with a resultant condition of *source exhausted.* The offset value for the receiver locates the byte following the last fully converted record in the receiver. The offset value for the source locates the byte following the last source record for which conversion is complete. The value of the remaining bytes in the receiver after the last converted record are unpredictable.

If the converted form of a record cannot be completely contained in the receiver, the instruction ends with a resultant condition of *receiver overrun.* The offset value for the receiver locates the byte following the last fully converted record in the receiver. The offset value for the source locates the byte following the last source record for which conversion is complete. The value of the remaining bytes in the receiver after the last converted record is unpredictable.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

If the source exhausted and the receiver overrun conditions occur at the same time, the source exhausted condition is recognized first. In this case, the offset into the receiver operand may contain a value equal to the length specified for the receiver which causes an exception to be signaled on the next invocation of the instruction. The processing performed for the source exhausted condition provides for this case if the instruction is invoked multiple times with the same controls operand template. When the receiver overrun condition is the resultant condition, the source always contains data that can be converted.

Resultant Conditions: Source exhausted–All full records in the source operand have been converted into the receiver operand. Receiver overrun–An overrun condition in the receiver operand was detected prior to processing all of the bytes in the source operand.

Events

000C Machine resources
  0201 Machine auxiliary storage exceeded

000D Machine status
  0101 Machine check

0010 Process
  0701 Maximum processor time exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
01 Conversion			X	
08 Length conformance	X			
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Invalid branch target				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		

CONVERT NUMERIC TO CHARACTER (CVTNC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10A3	Receiver	Source	Attributes

Operand 1: Character variable scalar or data-pointer-defined character scalar.

Operand 2: Numeric scalar or data-pointer-defined numeric scalar.

Operand 3: Character(7) scalar or data-pointer-defined character(7) scalar.

Description: The source numeric value (operand 2) is converted and copied to the receiver character string (operand 1). The receiver operand is treated as though it had the attributes supplied by operand 3.

Operand 1, when viewed in this manner, receives the numeric value of operand 2 following the rules of the Copy Numeric Value instruction.

The format of operand 3 is as follows:

• Scalar attributes	Char(7)
– Scalar type	Char(1)
Hex 00 = Binary	
Hex 01 = Floating-point	
Hex 02 = Zoned decimal	
Hex 03 = Packed decimal	
– Scalar length	Bin(2)
If binary:	
Length (L)	Bits 0-15
(where L = 2 or 4)	
If floating-point:	
Length	Bits 0-15
(where L = 4 or 8)	
If zoned decimal or packed decimal:	
Fractional digits (F)	Bits 0-7
Total digits (T) (where $1 \leq T \leq 31$ and $0 \leq F \leq T$)	Bits 8-15
– Reserved (binary 0)	Bin(4)

The byte length of operand 1 must be large enough to contain the numeric value described by operand 3. If it is not large enough, a scalar value invalid exception is signaled. If it is larger than needed, the numeric value is placed in the leftmost bytes and the unneeded rightmost bytes are unchanged by the instruction.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3 [4, 5]	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
04 External data object not found	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X		
06 Floating-point overflow	X			
07 Floating-point underflow	X			
09 Floating-point invalid operand		X		
0A Size	X			
0C Invalid floating-point conversion	X			
0D Floating-point inexact result	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attribute invalid			X	
03 Scalar value invalid			X	

CONVERT SNA TO CHARACTER (CVTSC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10DB	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(14) variable scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTSCI	18DB	Indicator
CVTSCB	1CDB	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a string value from the SNA (systems network architecture) format to character. The operation converts the source (operand 3) from SNA format to character under control of the controls (operand 2) and places the result into the receiver (operand 1).

The source and receiver operands must both be character strings. The source operand may not be specified as an immediate operand.

The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 14 bytes in length and have the following format:

• Controls operand base template	Char(14)
– Receiver offset	Bin(2)
– Source offset	Bin(2)
– Algorithm modifier	Char(1)
– Receiver record length	Char(1)
– Record separator	Char(1)
– Prime compression	Char(1)
– Unconverted receiver record bytes	Char(1)
– Conversion status	Char(2)
– Unconverted transparency string bytes	Char(1)
– Offset into template to translate table	Bin(2)
• Controls operand optional template extension	Char(64)
– Record separator translate table	Char(64)

Upon input to the instruction, the source and receiver offset fields specify the offsets where bytes of the source and receiver operands are to be processed. If an offset is equal to or greater than the length specified for the operand it corresponds to (it identifies a byte beyond the end of the operand), a template value invalid exception is signaled. As output from the instruction they are set to specify offsets that indicate how much of the operation is complete when the instruction ends.

The algorithm modifier specifies the optional functions to be performed. Any combination of functions not precluded by the bit definitions below is valid except that at least one of the functions must be specified. All algorithm modifier bits cannot be zero. Specification of an invalid algorithm modifier value results in a template value invalid exception. The meaning of the bits in the algorithm modifier is the following:

Bits	**Meaning**		
0	0	=	Do not perform decompression. Interpret a source character value of hex 00 as null.
	1	=	Perform decompression. Interpret a source character value of hex 00 as a record separator.
1-2	00	=	No record separators in source, no blank padding. Do not perform data transparency conversion.
	01	=	Reserved.
	10	=	Record separators in source, perform blank padding. Do not perform data transparency conversion.
	11	=	Record separators in source, perform blank padding. Perform data transparency conversion.
3-4	00	=	Do not put record separators into receiver.
	01	=	Move record separators from source to receiver (allowed only when bit 1 = 1).
	10	=	Translate record separators from source to receiver (allowed only when bit 1 = 1).
	11	=	Move record separator from controls to receiver.
5-7	Reserved		

The receiver record length value specifies the record length to be used to convert source records into the receiver operand. This length applies only to the data portion of the receiver record and does not include the optional record separator. Specification of a receiver record length of zero results in a template value invalid exception. The receiver record length value is ignored if no record separator processing is requested in the algorithm modifier.

The record separator value specifies the character that is to precede the converted form of each record in the receiver. The record separator character specified in the controls operand is used only for the case where the move record separator from controls to receiver function is specified in the algorithm modifier, or where a missing record separator in the source is detected.

The prime compression value specifies the character to be used as the prime compression character when performing decompression of the SNA format source data to character. It may have any value. The prime compression value is ignored if the decompression function is not specified in the algorithm modifier.

The unconverted receiver record bytes value specifies the number of bytes remaining in the current receiver record that are yet to be set with converted bytes from the source.

When record separator processing is specified in the algorithm modifier, this value is both input to and output from the instruction. On input, a value of hex 00 means it is the start of processing for a new record, and the initial conversion step is yet to be performed. This indicates that for the case where a function for putting record separators into the receiver is specified in the algorithm modifier, a record separator character has yet to be placed in the receiver. On input, a nonzero value less than or equal to the record length specifies the number of bytes remaining in the current receiver record that are yet to be set with converted bytes from the source. This value is assumed to be the valid count of unconverted receiver record bytes relative to the current byte to be processed in the receiver as located by the receiver offset value. As such, it is used to determine the location of the next record boundary in the receiver operand. This value must be less than or equal to the receiver record length value; otherwise, a template value invalid exception is signaled. On output, this field is set with a value as defined above which describes the number of bytes of the current receiver record not yet containing converted data.

When record separator processing is not specified in the algorithm modifier, this value is ignored.

The conversion status value specifies status information for the operation to be performed. The meaning of the bits in the conversion status is the following:

Bits	**Meaning**
0	0 = No transparency string active. 1 = Transparency string active. Unconverted transparency string bytes value contains the remaining string length.
1-15	Reserved

This field is both input to and output from the instruction. It provides for checkpointing the conversion status over successive executions of the instruction.

If the conversion status indicates transparency string active, but the algorithm modifier does not specify perform data transparency conversion, a template value invalid exception is signaled.

The unconverted transparency string bytes value specifies the number of bytes remaining to be converted for a partially processed transparency string in the source.

When perform data transparency conversion is specified in the algorithm modifier, the unconverted transparency string bytes value can be both input to and output from the instruction.

On input, when the no transparency string active status is specified in the conversion status, this value is ignored.

On input, when transparency string active status is specified in the conversion status, this value contains a count for the remaining bytes to be converted for a transparency string in the source. A value of hex 00 means the count field for a transparency string is the first byte of data to be processed from the source operand. A value of hex 01 through hex FF specifies the count of the remaining bytes to be converted for a transparency string. This value is assumed to be the valid count of unconverted transparency string bytes relative to the current byte to be processed in the source as located by the source offset value.

On output, this value is set if necessary along with the transparency string active status to describe a partially converted transparency string. A value of hex 00 will be set if the count field is the next byte to be processed for a transparency string. A value of hex 01 through hex FF specifying the number of remaining bytes to be converted for a transparency string, will be set if the count field has already been processed.

When do not perform data transparency conversion is specified in the algorithm modifier, the unconverted transparency string bytes value is ignored.

The offset into template to translate table value specifies the offset from the beginning of the template to the record separator translate table. This value is ignored unless the translate record separators from source to receiver function is specified in the algorithm modifier.

The record separator translate table value specifies the translate table to be used in translating record separators specified in the source to the record separator value to be placed into the receiver. It is assumed to be 64 bytes in length, providing for translation of record separator values of from hex 00 to hex 3F. This translate table is used only when the translate record separators from source to receiver function is specified in the algorithm modifier. See the record separator conversion function under the conversion process described below for more detail on the usage of the translate table.

Only the first 14 bytes of the controls operand base template and the optional 64-byte extension area specified for the record separator translate table are used. Any excess bytes are ignored.

The description of the conversion process is presented as a series of separately performed steps, which may be selected in allowable combinations to accomplish the conversion function. It is presented this way to allow for describing these functions separately. However, in the actual execution of the instruction, these functions may be performed in conjunction with one another or separately, depending upon which technique is determined to provide the best implementation.

The operation is performed either on a record-by-record basis, record processing, or on a nonrecord basis, string processing. This is determined by the functions selected in the algorithm modifier. Specifying the record separators in source, perform blank padding or move record separator from controls to receiver indicates record processing is to be performed. If neither of these functions is specified, in which case decompression must be specified, it indicates that string processing is to be performed.

The operation begins by accessing the bytes of the source operand at the location specified by the source offset.

When record processing is specified, the source offset may locate a point at which processing of a partially converted record is to be resumed or processing for a full record is to be started. The unconverted receiver record bytes value indicates whether conversion processing is to be started with a partial or a full record. Additionally, the transparency string active indicator in the conversion status field indicates whether conversion of a transparency string is active for the case of resumption of processing for a partially converted record. The conversion process is started by completing the conversion of a partial source record if necessary before processing the first full source record.

When string processing is specified, the source offset is assumed to locate the start of a compression entry.

When during the conversion process the end of the receiver operand is encountered, the instruction ends with a resultant condition of receiver overrun.

When record processing is specified in the algorithm modifier, this check is performed at the start of conversion for each record. A source exhausted condition would be detected before a receiver overrun condition if there is no source data to convert. If the receiver operand does not have room for a full record, the receiver overrun condition is recognized. The instruction is terminated with status in the controls operand describing the last completely converted record. For receiver overrun, partial conversion of a source record is not performed.

When string processing is specified in the algorithm modifier, then decompression must be specified and the decompression function described below defines the detection of receiver overrun.

When during the conversion process the end of the source operand is encountered, the instruction ends with a resultant condition of source exhausted. See the description of this condition in the conversion process described below to determine the status of the controls operand values and the converted bytes in the receiver for each case.

When string processing is specified, the bytes accessed from the source are converted on a string basis into the receiver operand at the location specified by the receiver offset. In this case, the decompression function must be specified and the conversion process is accomplished with just the decompression function defined below.

When record processing is specified the bytes accessed from the source are converted one record at a time into the receiver operand at the location specified by the receiver offset performing the functions specified in the algorithm modifier in the sequence defined by the following algorithm.

Record separator conversion is performed as requested in the algorithm modifier during the initial record separator processing performed as each record is being converted. This provides for controlling the setting of the record separator value in the receiver.

When the record separators in source option is specified, the following algorithm is used to locate them. A record separator is recognized in the source when a character value less than hex 40 is encountered. When do not perform decompression is specified, a source character value of hex 00 is recognized as a null value rather than as a record separator. In this case, the processing of the current record continues with the next source byte and the receiver is not updated. When perform data transparency conversion is specified, a character value of hex 35 is recognized as the start of a transparency string rather than as a record separator.

If the do not put record separators into the receiver function is specified, the record separator, if any, from the source record being processed is removed from the converted form of the source record and will not be placed in the receiver.

If the move record separators from the source to the receiver function is specified, the record separator from the source record being processed is left as is in the converted form of the source record and will be placed in the receiver.

If the translate record separators from the source to the receiver function is specified, the record separator from the source record being processed is translated using the specified translate table, replaced with its translated value in the converted form of the source record and, will be placed in the receiver. The translation is performed as in the translate instruction with the record separator value serving as the source byte to be translated. It is used as an index into the specified translate table to select the byte in the translate table that contains the value to which the record separator is to be set. If the selected translate table byte is equal to hex FF, it is recognized as an escape code. The instruction ends with a resultant condition of escape code encountered, and the controls operand is set to describe the conversion status as of the processing completed just prior to the conversion step for the record separator. If the selected translate table byte is not equal to hex FF, the record separator in the converted form of the record is set to its value.

If the move record separator from controls to receiver function is specified, the controls record separator value is used in the converted form of the source record and will be placed into the receiver.

When the record separators in source do blank padding function is requested, an assumed record separator will be used if a record separator is missing in the source data. In this case, the controls record separator character is used as the record separator to precede the converted record if record separators are to be placed in the receiver. The conversion process continues, bypassing the record separator conversion step that would normally be performed. The condition of a missing record separator is detected when during initial processing for a full record, the first byte of data is not a record separator character.

Decompression is performed if the function is specified in the algorithm modifier. This provides for converting strings of duplicate characters in compressed format in the source back to their full size in the receiver. Decompression of the source data is accomplished by concatenating together character strings described by the compression strings occurring in the source. The source offset value is assumed to locate the start of a compression string. Processing of a partial decompressed record is performed if necessary.

The character strings to be built into the receiver are described in the source by one or more compression strings. Compression strings are comprised of an SCB (string control byte) possibly followed by one or more bytes of data related to the character string to be built into the receiver.

The format of an SCB and the description of the data that may follow it is as follows:

- SCB Char(1)
  - Control Bits 0-1

    00 = n nondulicate characters are between this SCB and the next one; where n is the value of the count field (1-63).

    01 = Reserved.

    10 = This SCB represents n deleted prime compression characters; where n is the value of the count field (1-63). The next byte is the next SCB.

    11 = This SCB represents n deleted duplicate characters; where n is the value of the count field (1-63). The next byte contains a specimen of the deleted characters. The byte following the specimen character contains the next SCB.
  - Count Bits 2-7

    This contains the number of characters that have been deleted for a prime or duplicate string, or the number of characters to the next SCB for a nonduplicate string. A count value of zero is invalid and results in the signaling of a conversion exception.

Strings of prime compression characters or duplicate characters described in the source are repeated in the decompressed character string the number of times indicated by the SCB count value.

Strings of nonduplicate characters described in the source record are formed into a decompressed character string for the length indicated by the SCB count value.

If the end of the source is encountered prior to the end of a compression string, a conversion exception is signaled.

When record processing is specified, decompression is performed one record at a time. In this case, a conversion exception is signaled if a compression string describes a character string that would span a record boundary in the receiver. If the source contains record separators, the case of a missing record separator in the source is detected as defined under the initial description of the conversion process. Record separator conversion, as requested in the algorithm modifier, is performed as the initial step in the building of the decompressed record. A record separator to be placed into the receiver is in addition to the data to be converted into receiver for the length specified in the receiver record length field. The decompression of compression strings from the source continues until a record separator character for the next record is recognized when the source contains record separators, or until the decompressed data required to fill the receiver record has been processed or the end of the source is encountered whether record separators are in the source or not. Transparency strings encountered in the decompressed character string are not scanned for a record separator value. If the end of the source is encountered, the data decompressed to that point appended to the optional record separator for this record forms a partial decompressed record. Otherwise, the decompressed character strings appended to the optional record separator for this record form the decompressed record. The conversion process then continues for this record with the next specified function.

When string processing is specified, decompression is performed on a compression string basis with no record oriented processing implied. The conversion process for each compression string from the source is completed by placing the decompressed character string into the receiver. The conversion process continues decompressing compression strings from the source until the end of the source or the receiver is encountered. When the end of the source operand is encountered, the instruction ends with a resultant condition of source exhausted. When a character string cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. For either of the above ending conditions, the controls operand is updated to describe the status of the conversion operation as of the last completely converted compression entry. Partial conversion of a compression entry is not performed.

Data transparency conversion is performed if perform data transparency conversion is specified in the algorithm modifier. This provides for correctly identifying record separators in the source even if the data for a record contains value that could be interpreted as record separator values. Processing of active transparency strings is performed if necessary.

A nontransparent record is built by appending the nontransparent and transparent data converted from the record to the record separator for the record. The nontransparent record may be produced from either a partial record from the source or a full record from the source. This is accomplished by first accessing the record separator for a full record. The case of a missing record separator in the source is detected as defined under the initial description of the conversion process. Record separator conversion as requested in the algorithm modifier is performed if it has not already been performed by a prior step. Then the rest of the source record is scanned for values of less than hex 40.

A value greater than or equal to hex 40 is considered nontransparent data and is concatenated onto the record being built as is.

A value equal to hex 35 identifies the start of a transparency string. A transparency string is comprised of 2 bytes of transparency control information followed by the data to be made transparent to scanning for record separators. The first byte has a fixed value of hex 35 and is referred to as the TRN (transparency) control character. The second byte is a 1-byte hexadecimal count, a value remaining from 1 to 255 decimal, of the number of bytes of data that follow and is referred to as the TRN count. A TRN count of zero is invalid and causes a conversion exception. This contains the length of the transparent data and does not include the TRN control information length. The transparent data is concatenated to the nontransparent record being built and is not scanned for record separator characters.

A value equal to hex 00 is recognized as the record separator for the next record only when perform decompression is specified in the algorithm modifier. In this case, the nontransparent record is complete. When do not perform decompression is specified in the algorithm modifier, a value equal to hex 00 is ignored and is not included as part of the nontransparent data built for the current record.

A value less than hex 40 but not equal to hex 35 is considered to be the record separator for the next record, and the forming of the nontransparent record is complete.

The building of the nontransparent record is completed when the length of the data converted into the receiver equals the receiver record length if the record separator for the next record is not encountered prior to that point.

If the end of the source is encountered prior to completion of building the nontransparent record, the nontransparent record built up to this point is placed in the receiver and the instruction ends with a resultant condition of source exhausted. The controls operand is updated to describe the status for the partially converted record. This includes describing a partially converted transparency string, if necessary, by setting the active transparency string status and the unconverted transparency string bytes value.

If the building of the nontransparent record is completed prior to encountering the end of the source, the conversion process continues with the blank padding function described below.

Blank padding is performed if the function is specified in the algorithm modifier. This provides for expanding out to the size specified by the receiver record length the source records for which trailing blanks have been truncated. The padded record may be produced from either a partial record from the source or a full record from the source.

The record separator for this record is accessed. The case of a missing record separator in the source is detected as defined under the initial description of the conversion process. Record separator conversion, as requested in the algorithm modifier, is performed if it has not already been performed by a prior step.

The nontruncated data, if any, for the record is appended to the optional record separator for the record. The nontruncated data is determined by scanning the source record for the record separator for the next record. This scan is concluded after processing enough data to completely fill the receiver record or upon encountering the record separator for the next record. The data processed prior to concluding the scan is considered the nontruncated data for the record.

The blanks, if any, required to pad the record out to the nontruncated data for the record, concluding the forming of the padded record.

If the end of the source is encountered during the forming of the padded record, the data processed up to that point, appended to the optional record separator for the record, is placed into the receiver and the instruction ends with a resultant condition of source exhausted. The controls operand is updated to describe the status of the partially converted record.

If the forming of the padded record is concluded prior to encountering the end of the source, the conversion of the record is completed by placing the converted form of the record into the receiver.

At this point, either conversion of a source record has been completed or conversion has been interrupted due to detection of the source exhausted or receiver overrun condition. For record processing, if neither of the above conditions has been detected either during conversion of or at completion of conversion for the current record, the conversion process continues on the next source record with the decompression function described above.

At completion of the instruction, the offset value for the receiver locates the byte following the last converted byte in the receiver. The value of the remaining bytes in the receiver after the last converted byte are unpredictable. The offset value for the source locates the byte following the last source byte for which conversion was completed. When record processing is specified, the unconverted receiver record bytes value specifies the length of the receiver record bytes not yet containing converted data. When perform data transparency conversion is specified in the algorithm modifier, the conversion status indicates whether conversion of a transparency string was active and the unconverted transparency string bytes value specifies the length of the remaining bytes to be processed for an active transparency string.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

This instruction does not provide support for compression entries in the source describing data that would span records in the receiver. SNA data from some systems may violate this restriction and as such be incompatible with the instruction. A provision can be made to avoid this incompatibility by performing the conversion of the SNA data through two invocations of this instruction. The first invocation would specify decompression with no record separator processing. The second invocation would specify record separator processing with no decompression. This technique provides for separating the decompression step from record separator processing; thus, the incompatibility is avoided.

This instruction can end with the escape code encountered condition. In this case, it is expected that the user of the instruction will want to do some special processing for the record separator causing the condition. In order to resume execution of the instruction, the user will have to set the appropriate value for the record separator into the receiver and update the controls operand offset values correctly to provide for restarting processing at the right points in the receiver and source operands.

For the special case of a tie between the source exhausted and receiver overrun conditions, the source exhausted condition is recognized first. That is, when source exhausted is the resultant condition, the receiver may also be full. In this case, the offset into the receiver operand may contain a value equal to the length specified for the receiver, which would cause an exception to be detected on the next invocation of the instruction. The processing performed for the source exhausted condition should provide for this case if the instruction is to be invoked multiple times with the same controls operand template. When the receiver overrun condition is the resultant condition, the source will always contain data that can be converted.

This instruction will, in certain cases, ignore what would normally have been interpreted as a record separator value of hex 00. This applies (hex 00 is ignored) for the special case when do not perform decompression and record separators in source are specified in the algorithm modifier. Note that this does not apply when perform decompression is specified, or when do not perform decompression and no record separators in source and move record separator from controls to receiver are specified in the algorithm modifier.

Resultant Conditions

Source exhausted–The end of the source operand is encountered and no more bytes from the source can be converted. Receiver overrun–An overrun condition in the receiver operand is detected before all of the bytes in the source operand have been processed. Escape code encountered–A record separator character is encountered in the source operand that is to be treated as an escape code.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
01 Conversion			X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op-code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
2C Program Execution				
04 Invalid branch target				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		

COPY BITS WITH LEFT LOGICAL SHIFT (CPYBTLLS)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
102F	Receiver	Source	Shift control

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character(2) scalar.

Description: This instruction copies the bit string value of the source operand to the bit string defined by the receiver operand with a left logical shift of the source bit string value under control of the shift control operand.

The operation results in copying the shifted bit string value of the source to the bit string of the receiver while padding the receiver with bit values of 0 and truncating bit values of the source as is appropriate for the specific operation.

No indication is given of truncation of bit values from the shifted source value. This is true whether the values truncated are 0 or 1.

The operation is performed such that the bit string of the source is considered to be extended on the left and right by an unlimited number of bit string positions of value 0. Additionally, a receiver bit string view (window) with the attributes of the receiver is considered to overlay this conceptual bit string value of the source starting at the leftmost bit position of the original source value. A left logical shift of the conceptual bit string value of the source is then performed relative to the receiver bit string view according to the shift criteria specified in the shift control operand. After the shift, the bit string value then contained within the receiver bit string view is copied to the receiver.

The source and the receiver can be either character or numeric. Any numeric operands are interpreted as logical character strings. Due to the operation being treated as a character string operation, the source operand may not be specified as a signed immediate operand. Additionally, for a source operand specified as an unsigned immediate value, only a 1-byte immediate value may be specified.

The shift control operand may be specified as an immediate operand or as a character(2) scalar. It provides an unsigned binary value indicating the number of bit positions for which the left logical shift of the source bit string value is to be performed. A zero value specifies no shift.

Operands 1 and 2 may be specified as variable length substring compound operands.

Operand 3 may not be specified as a variable length substring compound operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands			Other
	1	2	3	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	

COPY BITS WITH RIGHT LOGICAL SHIFT (CPYBTRLS)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
103F	Receiver	Source	Shift control

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character(2) scalar.

Description: This instruction copies the bit string value of the source operand to the bit string defined by the receiver operand with a right logical shift of the source bit string value under control of the shift control operand.

The operation results in copying the shifted bit string value of the source to the bit string of the receiver while padding the receiver with bit values of 0 and truncating bit values of the source as is appropriate for the specific operation.

No indication is given of truncation of bit values from the shifted source value. This is true whether the values truncated are 0 or 1.

The operation is performed such that the bit string of the source is considered to be extended on the left and right by an unlimited number of bit string positions of value 0. Additionally, a receiver bit string view (window) with the attributes of the receiver is considered to overlay this conceptual bit string value of the source starting at the leftmost bit position of the original source value. A right logical shift of the conceptual bit string value of the source is then performed relative to the receiver bit string view according to the shift criteria specified in the shift control operand. After the shift, the bit string value then contained within the receiver bit string view is copied to the receiver.

The source and the receiver can be either character or numeric. Any numeric operands are interpreted as logical character strings. Due to the operation being treated as a character string operation, the source operand may not be specified as a signed immediate operand. Additionally, for a source operand specified as an unsigned immediate value, only a 1-byte immediate value may be specified.

The shift control operand may be specified as an immediate operand or as a character(2) scalar. It provides an unsigned binary value indicating the number of bit positions for which the right logical shift of the source bit string value is to be performed. A zero value specifies no shift.

Operands 1 and 2 may be specified as variable length substring compound operands.

Operand 3 may not be specified as a variable length substring compound operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	

COPY BYTES LEFT-ADJUSTED (CPYBLA)

Op Code (Hex)	Operand 1	Operand 2
10B2	Receiver	Source

Operand 1: Character variable scalar, numeric variable scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 2: Character scalar, numeric scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (no padding done).

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the shorter of the two operands. The copying begins with the two operands left-adjusted and proceeds until the shorter operand has been copied.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either operand is that no result is set.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
04 External data object not found	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	

COPY BYTES LEFT-ADJUSTED WITH PAD (CPYBLAP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10B3	Receiver	Source	Pad

Operand 1: Character variable scalar or numeric variable scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 2: Character scalar, numeric scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 3: Character scalar or numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (padded if needed).

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the receiver operand. If the source operand is shorter than the receiver operand, the source operand is copied to the leftmost bytes of the receiver operand, and each excess byte of the receiver operand is assigned the single byte value in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. If the source operand is longer than the receiver operand, the leftmost bytes of the source operand (equal in length to the receiver operand) are copied to the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with the single byte value of the pad operand. The effect of specifying a null substring reference for the receiver is that no result is set.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Space addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	04 External data object not found	X	X		
	06 Optimized addressability invalid	X	X	X	
08	Argument/Parameter				
	01 Parameter reference violation	X	X	X	
10	Damage Encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-Dependent Exception				
	03 Machine storage limit exceeded				X
20	Machine Support				
	02 Machine check				X
	03 Function check				X
22	Object Access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer Specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program Creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X		
	0C Invalid operand ODT reference	X	X	X	X
	0D Reserved bits are not zero	X	X	X	X
32	Scalar Specification				
	01 Scalar type invalid	X	X		

COPY BYTES OVERLAP LEFT-ADJUSTED (CPYBOLA)

Op Code (Hex)	Operand 1	Operand 2
10BA	Receiver	Source

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (no padding done).

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the shorter of the two operands. The copying begins with the two operands left-adjusted and proceeds until the shorter operand has been copied. The excess bytes in the longer operand are not included in the operation.

Predictable results occur even if two operands overlap because the source operand is, in effect, first copied to an intermediate result.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either operand is that no result is set.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	

COPY BYTES OVERLAP LEFT-ADJUSTED WITH PAD (CPYBOLAP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10BB	Receiver	Source	Pad

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character scalar or numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand.

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the receiver operand. If the source operand is shorter than the receiver operand, the source operand is copied to the leftmost bytes of the receiver operand and each excess byte of the receiver operand is assigned the single byte value in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. If the source operand is longer than the receiver operand, the leftmost bytes of the source operand (equal in length to the receiver operand) are copied to the receiver operand.

Predictable results occur even if two operands overlap because the source operand is, in effect, first copied to an intermediate result.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with the single byte value of the pad operand. The effect of specifying a null substring reference for the receiver is that no result is set.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X

COPY BYTES REPEATEDLY (CPYBREP)

Op Code (Hex)	Operand 1	Operand 2
10BE	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The logical string value of the source operand is repeatedly copied to the receiver operand until the receiver is filled.

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The operation begins with the two operands left-adjusted and continues until the receiver operand is completely filled. If the source operand is shorter than the receiver, it is repeatedly copied from left to right (all or in part) until the receiver operand is completely filled. If the source operand is longer than the receive operand, the leftmost bytes of the source operand (equal in length to the receiver operand) are copied to the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either operand is that no result is set.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

COPY BYTES RIGHT-ADJUSTED (CPYBRA)

Op Code (Hex)	Operand 1	Operand 2
10B6	Receiver	Source

Operand 1: Character variable scalar, numeric variable scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 2: Character scalar, numeric scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (no padding done).

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the shorter of the two operands. The rightmost bytes (equal to the length of the shorter of the two operands) of the source operand are copied to the rightmost bytes of the receiver operand. The excess bytes in the longer operand are not included in the operation.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either operand is that no result is set.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
04 External data object not found	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	

COPY BYTES RIGHT-ADJUSTED WITH PAD (CPYBRAP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10B7	Receiver	Source	Pad

Operand 1: Character variable scalar, numeric variable scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 2: Character scalar, numeric scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 3: Character scalar or numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (padded if needed).

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the receiver operand. If the source operand is shorter than the receiver operand, the source operand is copied to the rightmost bytes of receiver operand, and each excess byte is assigned the single byte value in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. If the source operand is longer than the receiver operand, the rightmost bytes of the source operand (equal in length to the receiver operand) are copied to the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with the single byte value of the pad operand. The effect of specifying a null substring reference for the receiver is that no result is set.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
04 External data object not found	X	X		
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X		

COPY HEX DIGIT NUMERIC TO NUMERIC (CPYHEXNN)

Op Code (Hex)	Operand 1	Operand 2
1092	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the source operand is copied to the numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the receiver operand.

The operands can be either character strings or numeric. Any numeric operands are interpreted as logical character strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

COPY HEX DIGIT NUMERIC TO ZONE (CPYHEXNZ)

Op Code (Hex)	Operand 1	Operand 2
1096	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the source operand is copied to the zone hex digit value (leftmost 4 bits) of the leftmost byte in the receiver operand.

The operands can be either character strings or numeric. Any numeric operands are interpreted as logical character strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands		
Exception	1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

COPY HEX DIGIT ZONE TO NUMERIC (CPYHEXZN)

Op Code (Hex)	Operand 1	Operand 2
109A	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The zone hex digit value (leftmost 4 bits) of the leftmost byte referred to by the source operand is copied to the numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the receiver operand.

The operands can be either character strings or numeric. Any numeric operands are interpreted as logical character strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

COPY HEX DIGIT ZONE TO ZONE (CPYHEXZZ)

Op Code (Hex)	Operand 1	Operand 2
109E	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The zone hex digit value (leftmost 4 bits) of the leftmost byte referred to by the source operand is copied to the zone hex digit value (leftmost 4 bits) of the leftmost byte referred to by the receiver operand.

The operands can be either character strings or numeric. Any numeric operands are interpreted as logical character strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

COPY NUMERIC VALUE (CPYNV)

Op Code (Hex)	Operand 1	Operand 2
1042	Receiver	Source

Operand 1: Numeric variable scalar or data-pointer-defined numeric scalar.

Operand 2: Numeric scalar or data pointer-defined-numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CPYNVR	1242	Round
CPYNVI	1842	Indicator
CPYNVIR	1A42	Indicator, Round
CPYNVB	1C42	Branch
CPYNVBR	1E42	Branch, Round

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The signed numeric value of the source operand is copied to the numeric receiver operand.

Both operands must be numeric. If necessary, the source operand is converted to the same type as the receiver operand before being copied to the receiver operand. The source value is adjusted to the length of the receiver operand, aligned at the assumed decimal point of the receiver operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations outlined in the *Functional Concepts Manual*. If significant digits are truncated on the left end of the source value, a size exception is signaled.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Conversions between floating-point integers and integer formats (binary or decimal with no fractional digits) is exact, except when an exception is signaled.

An invalid floating-point conversion exception is signaled when an attempt is made to convert from floating-point to binary or decimal and the result would represent infinity or NaN, or nonzero digits would be truncated from the left end of the resultant value.

For the optional round form of the instruction, a floating-point receiver operand is invalid.

For a fixed-point operation, if significant digits are truncated from the left end of the source value, a size exception is signaled.

For a floating-point receiver, if the exponent of the resultant value is too large or too small to be represented in the receiver field, the floating-point overflow and floating-point underflow exceptions are signaled, respectively.

Resultant Conditions: Positive, negative, or zero–The algebraic value of the numeric scalar receiver operand is either positive, negative, or zero. Unordered–The value assigned a floating-point receiver operand is NaN.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
04 External data object not found	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computation			
02 Decimal data		X	
06 Floating-point overflow	X		
07 Floating-point underflow	X		
09 Floating-point invalid operand		X	X
0A Size	X		
0C Invalid floating-point conversion	X		
0D Floating-point inexact result	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	X
03 Object suspended	X	X	X
24 Pointer Specification			
01 Pointer does not exist	X	X	X
02 Pointer type invalid	X	X	X
2A Program Creation			
05 Invalid op code extender field			X
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
09 Invalid branch target operand			X
0C Invalid operand ODT reference	X	X	X
0D Reserved bits are not zero	X	X	X
2C Program Execution			
04 Invalid branch target			X
32 Scalar Specification			
01 Scalar type invalid	X	X	

DIVIDE (DIV)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
104F	Quotient	Dividend	Divisor

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
DIVS	114F	Short
DIVR	124F	Round
DIVSR	134F	Short, Round
DIVI	184F	Indicator
DIVIS	194F	Indicator, Short
DIVIR	1A4F	Indicator, Round
DIVISR	1B4F	Indicator, Short, Round
DIVB	1C4F	Branch
DIVBS	1D4F	Branch, Short
DIVBR	1E4F	Branch, Round
DIVBSR	1F4F	Branch, Short, Round

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands will immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The signed numeric value of the dividend operand is divided by the numeric value of the divisor operand, and the result is placed in the quotient operand.

All of the operands must be numeric with any implicit conversions occurring according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual*.

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

If the divisor has a numeric value of zero, a zero divide or floating-point zero divide exception is signaled respectively for fixed-point versus floating-point operations. If the dividend has a value of zero, the result of the division is a zero quotient value.

If the divisor has a numeric value of 0, a zero divide exception is signaled. If the dividend has a value of 0, the result of the division is a zero value quotient.

For a decimal operation, the precision of the result of the divide operation is determined by the number of fractional digit positions specified for the quotient. In other words, the divide operation will be performed so as to calculate a resultant quotient of the same precision as that specified for the quotient operand. If necessary, internal alignment of the assumed decimal point for the dividend and divisor operands is performed to ensure the correct precision for the resultant quotient value. These internal alignments are not subject to detection of the decimal point alignment exception. An internal quotient value will be calculated for any combination of decimal attributes which may be specified for the instruction's operands. However, the assignment of the result to the quotient operand is subject to detection of the size exception thereby limiting the assignment to, at most, the rightmost 31 digits of the calculated result.

Floating-point division uses exponent subtraction and significand division.

If the dividend operand is shorter than the divisor operand, it is logically adjusted to the length of the divisor operand.

For fixed-point computations and for the significand division of a floating-point computation, the division operation is performed according to the rules of algebra.

For a floating-point computation, .the operation is performed as if to infinite precision.

The result of the operation is copied into the quotient operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the quotient operand, aligned at the assumed decimal point of the quotient operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules for arithmetic operations as outlined in the *Functional Concepts Manual*. If significant digits are truncated on the left end of the resultant value, a size exception is signaled. A decimal point alignment exception is also signaled when a division operation is performed in decimal and one of the following conditions occurs:

- The dividend operand is aligned, and the number of fractional digits specified in the divisor operand plus the number of fractional digits specified for the quotient operand plus the number of significant integer digits in the dividend operand exceeds 31.
- The divisor operand is aligned, and the number of fractional digits specified for the dividend operand minus the number of fractional digits specified for the quotient operand plus the number of significant integer digits in the divisor operand exceeds 31.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

For the optional round form of the instruction, specification of a floating-point receiver operand is invalid.

For fixed-point operations, if nonzero digits are truncated from the left end of the resultant value, a size exception is signaled.

For floating-point operations that involve a fixed-point receiver field, if nonzero digits would be truncated from the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point quotient operand, if the exponent of the resultant value is either too large or too small to be represented in the quotient field, the floating-point overflow and floating-point underflow exceptions are signaled, respectively.

Resultant Conditions: Positive, negative, or zero–The algebraic value of the numeric scalar quotient is positive, negative, or zero. Unordered–The value assigned a floating-point quotient operand is NaN.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
06 Floating-point overflow	X			
07 Floating-point underflow	X			
09 Floating-point invalid operand		X	X	X
0A Size	X			
0B Zero divide		X		
0C Invalid floating-point conversion	X			
0D Floating-point inexact result	X			
0E Floating-point divide by zero			X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Invalid branch target				X

DIVIDE WITH REMAINDER (DIVREM)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
1074	Quotient	Dividend	Divisor	Remainder

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Operand 4: Numeric variable scalar.

Optional Forms

(The optional forms apply to the quotient only.)

Mnemonic	Op Code (Hex)	Form Type
DIVREMS	1174	Short
DIVREMR	1274	Round
DIVREMSR	1374	Short, Round
DIVREMI	1874	Indicator
DIVREMIS	1974	Indicator, Short
DIVREMIR	1A74	Indicator, Round
DIVREMISR	1B74	Indicator, Short, Round
DIVREMB	1C74	Branch
DIVREMBS	1D74	Branch, Short
DIVREMBR	1E74	Branch, Round
DIVREMBSR	1F74	Branch, Short, Round

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The signed numeric value of the dividend operand is divided by the numeric value of the divisor operand; the quotient is placed in the quotient operand; the remainder is placed in the remainder operand.

The operands must be numeric with any implicit conversions occurring according to the rules for arithmetic operations as outlined in the *Functional Concepts Manual.*

If the divisor operand has a numeric value of 0, a zero divide exception is signaled. If the dividend operand has a value of 0, the result of the division is a zero value quotient and remainder.

For a decimal operation, the precision of the result of the divide operation is determined by the number of fractional digit positions specified for the quotient. In other words, the divide operation will be performed so as to calculate a resultant quotient of the same precision as that specified for the quotient operand. If necessary, internal alignment of the assumed decimal point for the dividend and divisor operands is performed to ensure the correct precision for the resultant quotient value. These internal alignments are not subject to detection of the decimal point alignment exception. An internal quotient value will be calculated for any combination of decimal attributes which may be specified for the instruction's operands. However, the assignment of the result to the quotient operand is subject to detection of the size exception thereby limiting the assignment to, at most, the rightmost 31 digits of the calculated result.

If the dividend operand is shorter than the divisor operand, it is logically adjusted to the length of the divisor operand.

The division operation is performed according to the rules of algebra. The quotient result of the operation is copied into the quotient operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the quotient operand, aligned at the assumed decimal point of the quotient operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual.* If significant digits are truncated on the left end of the resultant value, a size exception is signaled. A decimal point alignment exception is also signaled when a division operation is performed in decimal and one of the following conditions occurs:

- The dividend operand is aligned, and the number of fractional digits specified in the divisor operand plus the number of fractional digits specified for the quotient operand plus the number of significant integer digits in the dividend operand exceeds 31.

- The divisor operand is aligned, and the number of fractional digits specified for the dividend operand minus the number of fractional digits specified for the quotient operand plus the number of significant integer digits in the division operand exceeds 31.

After the quotient numeric value has been determined, the numeric value of the remainder operand is calculated as follows:

Remainder = Dividend - (Quotient*Divisor)

If the optional round form of this instruction is being used, the rounding applies to the quotient but not the remainder. The quotient value used to calculate the remainder is the resultant value of the division. The resultant value of the calculation is copied into the remainder operand. The sign of the remainder is the same as that of the dividend operand unless the remainder has a value of 0, in which case its sign is positive. If the remainder operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the remainder operand, aligned at the assumed decimal point of the remainder operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual.* If significant digits are truncated off the left end of the resultant value, a size exception is signaled.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: The algebraic value of the numeric scalar quotient is positive, negative, or 0.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	1	2	3	4	Other
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0C Computation					
02 Decimal data		X	X		
0A Size	X			X	
0B Zero divide			X		
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op code extender field					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand					X
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2C Program Execution					
04 Branch target invalid					X

EDIT (EDIT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10E3	Receiver	Source	Edit mask

Operand 1: Character variable scalar or data-pointer-defined character scalar.

Operand 2: Numeric scalar or data-pointer-defined numeric scalar.

Operand 3: Character scalar or data-pointer-defined character scalar.

Description: The value of a numeric scalar is transformed from its internal form to character form suitable for display at a source/sink device. The following general editing functions can be performed during transforming of the source operand to the receiver operand:

- Unconditional insertion of a source value digit with a zone as a function of the source value's algebraic sign
- Unconditional insertion of a mask operand character string
- Conditional insertion of one of two possible mask operand character strings as a function of the source value's algebraic sign
- Conditional insertion of a source value digit or a mask operand replacement character as a function of source value leading zero suppression
- Conditional insertion of either a mask operand character string or a series of replacement characters as a function of source value leading zero suppression
- Conditional floating insertion of one of two possible mask operand character strings as a function of both the algebraic sign of the source value and leading zero suppression

The operation is performed by transforming the source (operand 2) under control of the edit mask (operand 3) and placing the result in the receiver (operand 1).

The mask operand (operand 3) is limited to no more than 256 bytes.

Mask Syntax: The source field is converted to packed decimal format. The edit mask contains both control character and data character strings. Both the edit mask and the source fields are processed left to right, and the edited result is placed in the result field from left to right. If the number of digits in the source field is even, the four high-order bits of the source field are ignored and not checked for validity. All other source digits as well as the sign are checked for validity, and a decimal data exception is signaled when one is invalid. Overlapping of any of these fields gives unpredictable results.

Ten types of control characters can be in the edit mask, hex AA through hex AD and hex AF through hex B3. Four of these control characters specify strings of characters to be inserted into the result field under certain conditions; and the other five indicate that a digit from the source field should be checked and the appropriate action taken. There is one variable value control character (end-of-string character) that is in the edit mask. This control character indicates the end of a string of characters. The value of the end-of-string character can vary with each execution of the instruction and is determined by the value of the first character in the edit mask. If the first character of the edit mask is a value less than hex 40, then that value is used as the end-of-string character. If the first character of the edit mask is a value equal to or greater than hex 40, then hex AE is used as the end-of-string character.

A significance indicator is set to the off state at the start of the execution of this instruction. It remains in this state until a nonzero source digit is encountered in the source field or until one of the four unconditional digits (hex AA through hex AD) or an unconditional string (hex B3) is encountered in the edit mask.

When significance is detected, the selected floating string is overlaid into the result field immediately before (to the left of) the first significant result character.

When the significance indicator is set to the on state, the first significant result character has been reached. The state of the significance indicator determines whether the fill character or a digit from the source field is to be inserted into the result field for conditional digits and characters in conditional strings specified in the edit mask field. The fill character is a hex 40 until it is replaced by the first character following the floating string specification control character (hex B1).

When the significance indicator is in the off state:

- A conditional digit control character in the edit mask causes the fill character to be moved to the result field.
- A character in a conditional string in the edit mask causes the fill character to be moved to the result field.

When the significance indicator is in the on state:

- A conditional digit control character in the edit mask causes a source digit to be moved to the result field.
- A character in a conditional string in the edit mask is moved to the result field.

The following control characters are found in the edit mask field.

End-of-String Character

One of these control characters (a value less than hex 40 or hex AE) indicates the end of a character string and must be present even if the string is null.

Static Field Character

Hex AF This control character indicates the start of a static field. A static field is used to indicate that one of two mask character strings immediately following this character is to be inserted into the result field, depending upon the algebraic sign of the source field. If the sign is positive, the first string is to be inserted into the result field; if the sign is negative, the second string is to be inserted.

Static field format:

Hex AF positive string. . .less than hex 40 or hex AE negative string. . .hex AE

Floating String Specification Field Character

Hex B1 This control character indicates the start of a floating string specification field. The first character of the field is used as the fill character; following the fill character are two strings delimited by the end-of-string control character. If the algebraic sign of the source field is positive, the first string is to be overlaid into the result field; if the sign is negative, the second string is to be overlaid.

The string selected to be overlaid into the result field, called a floating string, appears immediately to the left of the first significant result character. If significance is never set, neither string is placed in the result field.

Conditional source digit positions (hex B2 control characters) must be provided in the edit mask immediately following the hex B1 field to accommodate the longer of the two floating strings; otherwise, a length conformance exception is signaled. For each of these B2 strings, the fill character is inserted into the result field, and source digits are not consumed. This ensures that the floating string never overlays bytes preceding the receiver operand.

Floating string specification field format:

Hex B1 fill character positive string. . .

end-of-string character negative string. . .end-of-string character

hex B2. . .

Conditional String Character

Hex B0 This control character indicates the start of a conditional string, which consists of any characters delimited by the end-of-string control character. Depending on the state of the significance indicator, this string or fill characters replacing it is inserted into the result field. If the significance indicator is off, a fill character for every character in the conditional string is placed in the result field. If the indicator is on, the characters in the conditional string are placed in the result field.

Conditional string format:

Hex B0 conditional string. . .end-of-string character

Unconditional String Character

Hex B3 This control character turns on the significance indicator and indicates the start of an unconditional string that consists of any characters delimited by the end-of-string control character. This string is unconditionally inserted into the result field regardless of the state of the significance indicator. If the indicator is off when a B3 control character is encountered, the appropriate floating string is overlaid into the result field before (to the left of) the B3 unconditional string (or to the left of where the unconditional string would have been if it were not null).

Unconditional string format:

Hex B3 unconditional string. . .end-of-string character

Control Characters That Correspond to Digits in the Source Field

Hex B2 This control character specifies that either the corresponding source field digit or the floating string (hex B1) fill character is inserted into the result field, depending on the state of the significance indicator. If the significance indicator is off, the fill character is placed in the result field; if the indicator is on, the source digit is placed. When a source digit is moved to the result field, the zone supplied is hex F. When significance (that is, a nonzero source digit) is detected, the floating string is overlaid to the left of the first significant character.

Control characters hex AA, hex AB, hex AC, and hex AD turn on the significance indicator. If the indicator is off when one of these control characters is encountered, the appropriate floating string is overlaid into the result field before (to the left of) the result digit.

Hex AA This control character specifies that the corresponding source field digit is unconditionally placed in the 4 low-order bits of the result field with the zone set to a hex F.

Hex AB This control character specifies that the corresponding source field digit is unconditionally placed in the result field. If the sign of the source field is positive, the zoned portion of the digit is set to hex F (the preferred positive sign); if the sign is negative, the zone portion is set to hex D (the preferred negative sign).

Hex AC This control character specifies that the corresponding source field digit is unconditionally placed in the result field. If the algebraic sign of the source field is positive, the zone portion of the result is set to hex F (the preferred positive sign); otherwise, the source sign field is moved to the result zone field.

Hex AD This control character specifies that the corresponding source field digit is unconditionally placed in the result field. If the algebraic sign of the source field is negative, the zone is set to hex D (the preferred negative sign); otherwise, the source field sign is moved to the zone position of the result byte.

The following table provides an overview of the results obtained with the valid edit conditions and sequences.

Mask Character	Previous Significance Indicator	Source Digit	Source Sign	Result Character(s)	Resulting Significance Indicator
AF	Off/On	Any	Positive	Positive string inserted	No Change
	Off/On	Any	Negative	Negative string inserted	No Change
AA	Off	0–9	Positive	Positive floating string overlaid; hex F, source digit	On
	Off	0–9	Negative	Negative floating string overlaid; hex F, source digit	On
	On	0–9	Any	Hex F, source digit	On
AB	Off	0–9	Positive	Positive floating string overlaid; hex F, source digit	On
	Off	0–9	Negative	Negative floating string overlaid; hex D, source digit	On
	On	0–9	Positive	Hex F, source digit	On
	On	0–9	Negative	Hex D, source digit	On
AC	Off	0–9	Positive	Positive floating string overlaid; hex F, source digit	
	Off	0–9	Negative	Negative floating string overlaid; source sign and digit	On
	On	0–9	Positive	Hex F, source digit	On
	On	0–9	Negative	Source sign and digit	On
AD	Off	0–9	Positive	Positive floating string overlaid; source sign and digit	On
	Off	0–9	Negative	Negative floating string overlaid; hex D, source digit	On
	On	0–9	Positive	Source sign and digit	On
	On	0–9	Negative	Hex D, source digit	On

Figure 2-1 (Part 1 of 2). Valid Edit Conditions and Results

Mask Character	Previous Significance Indicator	Source Digit	Source Sign	Result Character(s)	Resulting Significance Indicator
B0	Off	Any	Any	Insert fill character for each B0 string character	Off
	On	Any	Any	Insert B0 character string	On
B1 (including necessary B2s)	Off	Any	Any	Insert the fill character for each B2 character that corresponds to a character in the longer of the two floating strings	No Change
B2 (not for a B1 field)	Off	0	Any	Insert fill character	Off
	Off	1–9	Positive	Overlay positive floating string and insert hex F, source digit	On
	Off	1–9	Negative	Overlay negative floating string and insert hex F, source digit	On
	On	0–9	Any	Hex F, source digit	
B3	Off	Any	Positive	Overlay positive floating string and insert B3 character string	On
	Off	Any	Negative	Overlay negative floating string and insert B3 character string	On
	On	Any	Any	Insert B3 character string	On

Notes:

1. Any character is a valid fill character, including the end-of-string character.
2. Hex AF, hex B1, hex B0, and hex B3 strings must be terminated by the end-of-string character even if they are null strings.
3. If a hex B1 field has not been encountered (specified) when the significance indicator is turned on, the floating string is considered to be a null string and is therefore not used to overlay into the result field.
4. If the positive and negative strings of a static field are of unequal length, additional static fields are necessary to ensure that the sum of the lengths of the positive strings equal the sum of the lengths of the negative strings; otherwise, a length conformance exception is signaled because the receiver length does not correspond to the length implied by the edit mask and source field sign.

Figure 2-1 (Part 2 of 2). Valid Edit Conditions and Results

The following figure indicates the valid ordering of control characters in an edit mask field.

			Control Character Y				
		AA, AB, AC, AD	AF	B0	B1	B2	B3
	AA, AB, AC, AD	0	0	2	2	2	0
	AF	0	0	0	0	0	0
	B0	1	0	0	2	0	1
Control Character X	B1	1	0	1	3	1	1
	B2	1	0	0	2	0	1
	B3	0	0	2	2	2	0

Explanation:

Condition	Definition
0	Both X and Y can appear in the edit mask field in either order.
1	Y cannot precede X.
2	X cannot precede Y.
3	Both control characters (two B1's) cannot appear in an edit mask field.

Violation of any of the above rules will result in an edit mask syntax exception.

Figure 2-2. Edit Mask Field Control Characters

The following steps are performed when the editing is done:

- Convert Source Value to Packed Decimal
  - The numeric value in the source operand is converted to a packed decimal intermediate value before the editing is done. If the source operand is binary, then the attributes of the intermediate packed field before the edit are calculated as follows:

    Binary(2) = packed (5,0) or
    Binary(4) = packed (10,0)

- Edit
  - The editing of the source digits and mask insertion characters into the receiver operand is done from left to right.

- Insert Floating String into Receiver Field
  - If a floating string is to be inserted into the receiver field, this is done after the other editing.

Edit Digit Count Exception

An edit digit count exception is signaled when:

- The end of the source field is reached and there are more control characters that correspond to digits in the edit mask field.

- The end of the edit mask field is reached and there are more digit positions in the source field.

Edit Mask Syntax Exception

An edit mask syntax exception is signaled when an invalid edit mask control character is encountered or when a sequence rule is violated.

Length Conformance Exception

A length conformance exception is signaled when:

- The end of the edit mask field is reached and there are more character positions in the result field.

- The end of the result field is reached and more positions remain in the edit mask field.

- The number of B2s following a B1 field cannot accommodate the longer of the two floating strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
04 External data object not found	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X		
04 Edit digit count		X		
05 Edit mask syntax			X	
08 Length conformance	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	

EXCHANGE BYTES (EXCHBY)

Op Code (Hex)	**Operand 1**	**Operand 2**
10CE	Source 1	Source 2

Operand 1: Character variable scalar (fixed-length) or numeric variable scalar.

Operand 2: Character variable scalar (fixed-length) or numeric variable scalar.

Description: The logical character string values of the two source operands are exchanged. The value of the second source operand is placed in the first source operand and the value of the first source operand is placed in the second operand.

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings. Both operands must have the same length.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

0OOC Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

EXCLUSIVE OR (XOR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
109B	Receiver	Source 1	Source 2

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
XORS	119B	Short
XORI	189B	Indicator
XORIS	199B	Indicator, Short
XORB	1C9B	Branch
XORBS	1D9B	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Boolean EXCLUSIVE OR operation is performed on the string values in the source operands. The resulting string is placed in the receiver operand.

The operands must be character strings and are interpreted as bit strings.

The length of the operation is equal to the length of the longer of the two source operands. The shorter of the two operands is padded on the right. The operation begins with the two source operands left-adjusted and continues bit by bit until they are completed.

The bit values of the result are determined as follows:

Source 1 Bit	Source 2 Bit	Result Bit
1	1	0
0	0	0
1	0	1
0	1	1

The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right.

The pad value used in this instruction is a hex 00.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1, 2, and 3. The effect of specifying a null substring reference for one source operand is that the other source operand is EXCLUSIVE ORed with an equal length string of all hex 00s. When a null substring reference is specified for both source operands, the result is all zero and the instruction's resultant condition is zero. When a null substring reference is specified for the receiver, a result is not set and the instruction's resultant condition is zero regardless of the values of the source operands.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Resultant Conditions: Zero—The bit value for the bits of the scalar receiver operand is either all zero or a null substring reference is specified for the receiver. Not zero—The bit value for the bits of the scalar receiver operand is not all zero.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

EXTENDED CHARACTER SCAN (ECSCAN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
10D4	Receiver	Base	Compare operand	Mode operand

Operand 1: Binary variable scalar or binary array.

Operand 2: Character scalar.

Operand 3: Character scalar.

Operand 4: Character(1) scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ESCANI	18D4	Indicator
ESCANB	1CD4	Branch

Extender: Branch or indicator options.

Either the branch option or indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction scans the string value of the base operand for occurrences of the string value of the compare operand and indicates the relative locations of these occurrences in the receiver operand. The character string value of the base operand is scanned for occurrences of the character string value of the compare operand under control of the mode operand and mode control characters embedded in the base string.

The base and compare operands must both be character strings. The length of the compare operand must not be greater than that of the base string. The base and compare operand are interpreted as containing a mixture of 1-byte (simple) and 2-byte (extended) character codes. The mode, simple or extended, with which the string is to be interpreted, is controlled initially by the mode operand and thereafter by mode control characters embedded in the strings. The mode control characters are as follows:

- Hex 0E= Shift out of simple character mode to extended mode.

- Hex 0F= Shift into simple character mode from extended mode. This is recognized only if it occurs in the first byte position of an extended character code.

The format of the mode operand is as follows:

• Mode operand	Char(1)
– Operand 2 initial mode indicator	Bit 0
0 = Operand starts in simple character mode.	
1 = Operand starts in extended character mode.	
– Operand 3 initial mode indicator	Bit 1
0 = Operand starts in simple character mode.	
1 = Operand starts in extended character mode.	
– Reserved (binary 0)	Bits 2-7

The operation begins at the left end of the base string and continues character by character, left to right. When the base string is interpreted in simple character mode, the operation moves through the base string 1 byte at a time. When the base string is interpreted in extended character mode, the operation moves through the base string 2 bytes at a time.

The compare operand value is the entire byte string specified for the compare operand. The mode operand determines the initial mode of the compare operand. The first character of the compare operand value is assumed to be a valid character for the initial mode of the compare operand and not a mode control character. Mode control characters in the compare operand value participate in comparisons performed during the scan function except that a mode control character as the first character of the compare operand causes unpredictable results.

The base string is scanned until the mode of the characters being processed is the same as the initial mode of the compare operand value. The operation continues comparing the characters of the base string with those of the compare operand value. The starting character of the characters being compared in the base string is always a valid character for the initial mode of the compare operand value. A mode control character encountered in the base string that changed the base string mode to match the initial mode of the compare operand value does not participate in the comparison. The length of the comparison is equal to the length of the compare operand value and the comparison is performed the same as performed by the Compare Bytes Left Adjusted instruction.

If a set of bytes that matches the compare operand value is found, the binary value for the relative location of the leftmost base string character of the set of bytes is placed in the receiver operand.

If the receiver operand is a scalar, only the first occurrence of the compare operand is noted. If the receiver operand is an array, as many occurrences as there are elements in the array are noted.

If a mode change is encountered in the base string, the base string is again scanned until the mode of the characters being processed is the same as the initial mode of the compare operand value, and then the comparisons are resumed.

The operation continues until no more occurrences of the compare operand value can be noted in the receiver operand or until the number of bytes remaining to be scanned in the base string is less than the length of the compare operand value. When the second condition occurs, the receiver value is set to zero. If the receiver operand is an array, all its remaining elements are also set to zero.

If the escape code encountered result condition is specified (through a branch or indicator option), verifications are performed on the base string as it is scanned. Each byte of the base string is checked for a value less than hex 40. When a value less than hex 40 is encountered, it is then determined if it is a valid mode control character.

If a byte value of less than hex 40 is not a valid mode control character, it is considered to be an escape code. The binary value for the relative location of the character (simple or extended) being interrogated is placed in the receiver operand, and the appropriate action (indicator or branch) is performed according to the specification of the escape code encountered result condition. If the receiver operand is an array, the next array element after any elements set with locations or prior occurrences of the compare operand, is set with the location of the character containing the escape code and all the remaining array elements are set to zero.

If the escape encountered result condition is not specified, verifications of the character codes in the base string are not performed.

Resultant Conditions: Positive or zero–The numeric value(s) of the receiver operand is either positive or zero. In the case where the receiver operand is an array, the resultant condition is zero if all elements are zero. Escape code encountered–An escape character code value was encountered during the scanning of the base string.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment violation	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0C Computation					
08 Length conformance		X	X	X	
10 Damage Encountered					
44 Partial system object damage					X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op-code extender field					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand					X
0A Invalid operand length		X	X		
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2C Program Execution					
04 Invalid branch target					X
32 Scalar Specification					
01 Scalar type invalid	X	X	X	X	
03 Scalar value invalid				X	

EXTRACT EXPONENT (EXTREXP)

Op Code (Hex)	Operand 1	Operand 2
1072	Receiver	Source

Operand 1: Binary variable scalar.

Operand 2: Floating-point scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
EXTREXPI	1872	Indicator
EXTREXPB	1C72	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operations immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction extracts the exponent portion of a floating-point scalar source operand and places it into the receiver operand as a binary variable scalar.

The operands must be the numeric types indicated because no conversions are performed.

The source floating-point field is interrogated to determine the binary floating-point value represented and either a signed exponent, for number values, or a special identifier, for infinity and NaN values, is placed in the binary variable scalar receiver operand.

The value to be assigned to the receiver, is dependent upon the floating-point value represented in the source operand as described below. It is a signed binary integer value and a numeric assignment of the value is made to the receiver.

When the source represents a normalized number, the biased exponent contained in the exponent field of the source is converted to the corresponding signed exponent by subtracting the bias of 127 for short or 1023 for long to determine the value to be returned. The resulting value ranges from -126 to +127 for short format, -1022 to +1023 for long format.

When the source represents a denormalized number, the value to be returned is determined by adjusting the signed exponent of the denormalized number. The signed exponent of a denormalized number is a fixed value of -126 for the short format and -1022 for the long format. It is adjusted to the value the signed exponent would be if the source value was adjusted to a normalized number. The resulting value ranges from -127 to -149 for short format, -1023 to -1074 for long format.

When the source represents a value of zero, the value returned is zero.

When the source represents infinity, the value returned is +32767.

When the source represents a not-a-number, the value returned is -32768.

Resultant Conditions: Normalized–The source operand value represents a normalized binary floating-point number. The signed exponent is stored in the receiver. Denormalized–The source operand value represents a denormalized binary floating-point number. An adjusted signed exponent is stored in the receiver. Infinity–The source operand value represents infinity. The receiver is set with a value of +32767. NaN–The source operand value represents a not-a-number. The receiver is set with a value of -32768.

Events

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

000D Machine status
0101 Machine check

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment violation	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
44 Partial system object damage			X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
05 Invalid op-code extender field			X
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
09 Invalid branch target operand			X
0C Invalid operand ODT reference	X	X	X
0D Reserved bits are not zero	X	X	X
2C Program Execution			
04 Invalid branch target			X
32 Scalar Specification			
01 Scalar type invalid	X	X	

EXTRACT MAGNITUDE (EXTRMAG)

Op Code (Hex)	Operand 1	Operand 2
1052	Receiver	Source

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
EXTRMAGS	1152	Short
EXTRMAGI	1852	Indicator
EXTRMAGIS	1952	Indicator, Short
EXTRMAGB	1C52	Branch
EXTRMAGBS	1D52	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The numeric value of the source operand is converted to its absolute value and placed in the numeric variable scalar receiver operand.

The absolute value is formed from the source operand as follows:

- Binary
  - Extract the numeric value and form twos complement if the source operand is negative.
- Packed/Zoned
  - Extract the numeric value and force the source operand's sign to positive.
- Floating-point
  - Extract the numeric value and force the significand sign to positive.

The result of the operation is copied into the receiver operand according to the rules of the Copy Numeric Value instruction. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the receiver operand, or aligned at the assumed decimal point of the receiver operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations outlined in the *Functional Concepts Manual.* If significant digits are truncated on the left end of the resultant value, a size exception is signaled. An attempt to extract the magnitude of a maximum negative binary value to a binary scalar of the same size also results in a size exception.

When the source floating-point operand represents not-a-number, the sign field of the source is not forced to positive and this value is not altered in the receiver.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

For a fixed-point operation, if significant digits are truncated from the left end of the resultant value, a size exception is signaled. An attempt to extract the absolute value of a maximum negative binary value into a binary scalar of the same size also results in a size exception.

For floating-point operations that involve a fixed-point receiver field, if nonzero digits would be truncated from the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point receiver operand, if the exponent of the resultant value is either too large or too small to be represented in the receiver field, the floating-point overflow or the floating-point underflow exception is signaled.

Resultant Conditions: Positive or zero—The algebraic value of the receiver operand is either positive or zero. Unordered—The value assigned a floating-point receiver operand is NaN.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computation			
02 Decimal data		X	
06 Floating-point overflow	X		
07 Floating-point underflow	X		
09 Floating-point invalid operand		X	
0A Size	X		
0D Floating-point inexact result	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
05 Invalid op code extender field			X
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
09 Invalid branch target operand			X
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
04 Branch target invalid			X

MULTIPLY (MULT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
104B	Product	Multiplicand	Multiplier

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
MULTS	114B	Short
MULTR	124B	Round
MULTSR	134B	Short, Round
MULTI	184B	Indicator
MULTIS	194B	Indicator, Short
MULTIR	1A4B	Indicator, Round
MULTISR	1B4B	Indicator, Short, Round
MULTB	1C4B	Branch
MULTBS	1D4B	Branch, Short
MULTBR	1E4B	Branch, Round
MULTBSR	1F4B	Branch, Short, Round

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The signed numeric value of the multiplicand operand is multiplied by the numeric value of the multiplier operand and the result is placed in the product operand.

The operands must be numeric with any implicit conversions occurring according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual.*

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

If the multiplicand operand or the multiplier operand has a value of 0, the result of the multiplication is a zero product.

For a decimal operation, no alignment of the assumed decimal point is performed for the multiplier and multiplicand operands.

The operation occurs using the specified lengths of the multiplicand and multiplier operands with no logical zero padding on the left necessary.

Floating-point multiplication uses exponent addition and significand multiplication.

For nonfloating-point computations and for significand multiplication for floating-point operations, the multiplication operation is performed according to the rules of algebra.

The result of the operation is copied into the product operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the product operand, aligned at the assumed decimal point of the product operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations outlined in the *Functional Concepts Manual.*

For the optional round form of the instruction, specification of a floating-point receiver operand is invalid.

For fixed-point operations, if nonzero digits are truncated from the left end of the resultant value, a size exception is signaled.

For floating-point operations involving a fixed-point receiver field (if nonzero digits would be truncated from the left end of the resultant value), an invalid floating-point conversion exception is signaled.

For a floating-point product operand, if the exponent of the resultant value is either too large or too small to be represented in the product field, the floating-point overflow or the floating-point underflow exception is signaled.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: Positive, negative, or zero–The algebraic value of the numeric scalar product is positive, negative, or zero. Unordered–The value assigned a floating-point product operand is NaN.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3 [4, 5]**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
06 Floating-point overflow	X			
07 Floating-point underflow	X			
09 Floating-point invalid operand		X	X	X
0A Size	X			
0C Invalid floating-point conversion	X			
0D Floating-point inexact result	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

NEGATE (NEG)

Op Code (Hex)	Operand 1	Operand 2
1056	Receiver	Source

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
NEGS	1156	Short
NEGI	1856	Indicator
NEGIS	1956	Indicator, Short
NEGB	1C56	Branch
NEGBS	1D56	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The sign of the numeric value in the source operand is changed as if it had been multiplied by a negative one (-1). The result is placed in the receiver operand.

The sign changing of the source operand value (positive to negative and negative to positive) is performed as follows:

- Binary
  - Extract the numeric value and form the twos complement of it.
- Packed/Zoned
  - Extract the numeric value and force its sign to positive if it is negative or to negative if it is positive.
- Floating-point
  - Extract the numeric value and force the significand sign to positive if it is negative or to negative if it is positive.

The result of the operation is copied into the receiver operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the receiver operand, aligned at the assumed decimal point of the receiver operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations outlined in the *Functional Concepts Manual.* If significant digits are truncated on the left end of the resultant value, a size exception is signaled. An attempt to negate a maximum negative binary value to a binary scalar of the same size also results in a size exception. If a packed or zoned 0 is negated, the result is always positive 0.

When the source floating-point operand represents not-a-number, the sign field of the source is not forced to positive and this value is not altered in the receiver.

For a fixed-point operation, if significant digits are truncated from the left end of the resultant value, a size exception is signaled. An attempt to negate a maximum negative binary value into a binary scalar of the same size also results in a size exception.

For floating-point operations that involve a fixed-point receiver field, if nonzero digits would be truncated from the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point receiver operand, if the exponent of the resultant value is either too large or too small to be represented in the receiver field, the floating-point overflow and the floating-point underflow exceptions are signaled.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: Positive, negative, or zero–The algebraic value of the receiver operand is either positive, negative, or zero. Unordered–The value assigned a floating-point receiver operand is NaN.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	**Operands**		**Other**
	1	**2**	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computation			
02 Decimal data		X	
06 Floating-point overflow	X		
07 Floating-point underflow	X		
09 Floating-point invalid operand		X	X
0A Size	X		
0C Invalid floating-point conversion	X		
0C Floating-point inexact result	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
05 Invalid op code extender field			X
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
09 Invalid branch target operand			X
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
04 Branch target invalid			X

NO OPERATION (NOOP)

Op Code (Hex)
0000

Description: No function is performed. The instruction consists of an operation code and no operands. The instruction may not be branched to and is not counted as an instruction in the instruction stream.

The instruction may be used for inserting gaps in the instruction stream. These gaps allow instructions with adjacent instruction addresses to be physically separated.

The instruction may precede or follow any machine instruction except the End instruction, and any number of No Operation instructions may exist in succession.

NO OPERATION AND SKIP (NOOPS)

Op Code (Hex)	Operand 1
0001	Skip count

Operand 1: Unsigned immediate value.

Description: This instruction performs no function other than to indicate a specific number of bytes within the instruction stream that are to be skipped during encapsulation. It consists of an operation code and 1 operand. Operand 1 is an unsigned immediate value that contains the number of bytes between this instruction and the next instruction to be processed. These bytes are skipped during the encapsulation of this program. A value of zero for operand 1 indicates that no bytes are to be skipped between this instruction and the next instruction to be processed.

If the operand 1 skip count indicates that the next instruction to process is beyond the end of the instruction stream, an invalid operand value range exception is signaled.

This instruction may be used to insert gaps in the instruction stream in such a manner that allows instructions with adjacent instruction addresses to not be physically adjacent.

This instruction may not be branched to, and is not counted as an instruction in the instruction stream.

The instruction may precede or follow any machine instruction except the End instruction, and any number of No Operation and Skip instructions may exist in succession.

Note: When this instruction is used in an existing program template, the following items within the template may be adversely affected:

- The actual count of instructions may be altered to be different than the count of instructions that is specified in the program template header.
- Object definitions that reference instructions may now be out of range or may not reference the intended instruction.

The actual number of bytes skipped includes the bytes containing the instruction plus the number of bytes specified by the skip count value. The number of bytes skipped per template version is as follows:

- Version 0 = 4 plus the skip count value.
- Version 1 = 5 plus the skip count value.

Exceptions

Exception	Operand 1	Other
2A Program Creation		
06 Invalid operand type	X	
08 Invalid operand value range	X	
0D Reserved bits are not zero	X	X

NOT (NOT)

Op Code (Hex)	Operand 1	Operand 2
108A	Receiver	Source

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
NOTS	118A	Short
NOTI	188A	Indicator
NOTIS	198A	Indicator, Short
NOTB	1C8A	Branch
NOTBS	1D8A	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Boolean NOT operation is performed on the string value in the source operand. The resulting string is placed in the receiver operand.

The operands must be character strings; they are interpreted as bit strings.

The length of the operation is equal to the length of the source operand.

The bit values of the result are determined as follows:

Source Bit	**Result Bit**
1	0
0	1

The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a hex 00 byte.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source operand is that the result is all zero and the instruction's resultant condition is zero. When a null substring reference is specified for the receiver, a result is not set and the instruction's resultant condition is zero regardless of the value of the source operand.

Resultant Conditions: Zero—The bit value for the bits of the scalar receiver operand is either all zero or a null substring reference is specified for the receiver. Not zero—The bit value for the bits of the scalar receiver operand is not all zero.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	**Operands 1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
05 Invalid op code extender			X
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
09 Invalid branch target operand			X
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
04 Branch target invalid			X

OR (OR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1097	Receiver	Source 1	Source 2

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ORS	1197	Short
ORI	1897	Indicator
ORIS	1997	Indicator, Short
ORB	1C97	Branch
ORBS	1D97	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or Indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Boolean OR operation is performed on the string values in the source operands. The resulting string is placed in the receiver operand.

The operands must be character strings; they are interpreted as bit strings.

The length of the operation is equal to the length of the longer of the two source operands. The shorter of the two operands is logically padded on the right with hex 00. The excess bytes in the longer operand are assigned to the results.

The bit values of the result are determined as follows:

Source 1 Bit	Source 2 Bit	Result Bit
1	1	1
0	1	1
1	0	1
0	0	0

The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a hex 00.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1, 2, and 3. The effect of specifying a null substring reference for one source operand is that the other source operand is ORed with an equal length string of all hex 00s. This causes the value of the other operand to be assigned to the result. When a null substring reference is specified for both source operands, the result is all zero and the instruction's resultant condition is zero. When a null substring reference is specified for the receiver, a result is not set and the instruction's resultant condition is zero regardless of the values of the source operands.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Resultant Conditions: Zero–The bit value for the bits of the scalar receiver operand is either all zero or a null substring reference is specified for the receiver. Not zero–The bit value for the bits of the scalar receiver operand is not all zero.

Events

0000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

REMAINDER (REM)

Op Code (Hex)	**Operand 1**	**Operand 2**	**Operand 3**
1073	Remainder	Dividend	Divisor

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	**Op Code (Hex)**	**Form Type**
REMS	1173	Short
REMI	1873	Indicator
REMIS	1973	Indicator, Short
REMB	1C73	Branch
REMBS	1D73	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The signed numeric value of the dividend operand is divided by the numeric value of the divisor operand, and the remainder is placed in the remainder operand.

The operands must be numeric with any implicit conversions occurring according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual.*

If the divisor has a numeric value of 0, a zero divide exception is signaled. If the dividend has a value of 0, the result of the division is a zero value remainder.

For a decimal operation, the internal quotient value produced by the divide operation is always calculated with a precision of zero fractional digit positions. If necessary, internal alignment of the assumed decimal point for the dividend and divisor operands is performed to insure the correct precision for the resultant quotient value. These internal alignments are not subject to detection of the decimal point alignment exception. An internal quotient and the corresponding remainder value will be calculated for any combination of decimal attributes which may be specified for the instruction's operands. However, as described below, the assignment of the remainder value is limited to that portion of the remainder value which fits in the remainder operand.

If the dividend is shorter than the divisor, it is logically adjusted to the length of the divisor.

The division operation is performed according to the rules of algebra. Before the remainder is calculated, an intermediate quotient is calculated. The attributes of this quotient are derived from the attributes of the dividend and divisor operands as follows:

Dividend	**Divisor**	**Intermediate Quotient**
IM,SIM, or BIN(2)	IM,SIM, or BIN(2)	BIN(2)
IM,SIM, or BIN(2)	BIN(4)	BIN(4)
IM,SIM, or BIN(2)	DECIMAL(P2,Q2)	DECIMAL(5+Q2,0)
BIN(4)	IM,SIM, or BIN(2)	BIN(4)
BIN(4)	DECIMAL(P2,Q2)	DECIMAL(10+Q2,0)
DECIMAL(P1,Q1)	IM,SIM, or BIN(2)	DECIMAL(P1,0)
DECIMAL(P1,Q1)	BIN(4)	DECIMAL(P1,0)
DECIMAL(P1,Q1)	DECIMAL(P2,Q2)	DECIMAL(P1-Q1+Q,0 Where Q = Larger of Q1 or Q2

IM = IMMEDIATE
SIM = SIGNED IMMEDIATE
DECIMAL = PACKED OR ZONED

After the intermediate quotient numeric value has been determined, the numeric value of the remainder operand is calculated as follows:

Remainder = Dividend - (Quotient*Divisor)

The sign of the remainder is the same as that of the dividend unless the remainder has a value of 0. When the remainder has a value of 0, the sign of the remainder is positive.

The resultant value of the calculation is copied into the remainder operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the remainder operand, aligned at the assumed decimal point of the remainder operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual.* If significant digits are truncated on the left end of the resultant value, a size exception is signaled.

An exception is also signaled when a decimal division operation is performed and one of the following conditions occurs:

- The dividend is aligned, and the number of fractional digits specified in the divisor plus the number of fractional digits specified for the quotient plus the number of significant integer digits in the dividend exceeds 31.
- The divisor is aligned, and the number of fractional digits specified for the dividend minus the number of fractional digits specified for the quotient plus the number of significant integer digits in the divisor exceeds 31.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: The algebraic value of the numeric scalar remainder is positive, negative, or 0.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage object
- 0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
0A Size	X			
0B Zero divide			X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	X
02 Pointer type invalid	X	X	X	X
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target				X
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

SCALE (SCALE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1063	Receiver	Source	Scale factor

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Binary(2) scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SCALES	1163	Short
SCALEI	1863	Indicator
SCALEIS	1963	Indicator, Short
SCALEB	1C63	Branch
SCALEBS	1D63	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The scale instruction performs numeric scaling of the source operand based on the scale factor and places the results in the receiver operand. The numeric operation is as follows:

Operand 1 = Operand 2 *(B**N)

where:

N is the binary integer value of the scale operand. It can be positive, negative, or 0. If N is 0, then the operation simply copies the source operand value into the receiver operand.

B is the arithmetic base for the type of numeric value in the source operand.

Base Type	B
Binary	2
Packed/Zoned	10
Floating-point	2

The operands must be of the numeric types indicated with any implicit conversions occurring according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual.* The scale operation is a shift of N binary, packed, or zoned digits. The shift is to the left if N is positive, to the right if N is negative. For a floating-point source operand, the scale operation is performed as if the source operand is multiplied by a floating-point value of 2**N.

If the source and receiver operands have different attributes, the scaling operation is done in an intermediate field with the same attributes as the source operand. If a fixed-point scaling operation causes nonzero digits to be truncated on the left end of the intermediate field, a size exception is signaled. For a floating-point scaling operation, the floating-point overflow and the floating-point underflow exceptions can be signaled during the calculation of the intermediate result.

The resultant value of the calculation is copied into the receiver operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the receiver operand, aligned at the assumed decimal point of the receiver operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations outlined in the *Functional Concepts Manual*. For fixed-point operations, if nonzero digits are truncated off the left end of the resultant value, a size exception is signaled.

For floating-point operations involving fixed-point receiver fields, if nonzero digits would be truncated from the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For floating-point receiver fields, if the exponent of the resultant value is either too large or too small to be represented in the receiver field, the floating-point overflow or floating-point underflow exception is signaled.

A scalar value invalid exception is signaled if the value of N is beyond the range of the particular type of the source operand as specified in the following table.

Source Operand Type	**Maximum Value of N**
Binary(2)	$-14 \le N \le 14$
Binary(4)	$-30 \le N \le 30$
Decimal(P,Q)	$-31 \le N \le 31$

For a scale operation in floating-point, no limitations are placed on the values allowed for N other than the implicit limits imposed due to the floating-point overflow and underflow exceptions.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Condition: Positive, negative, or zero–The algebraic value of the receiver operand is positive, negative, or zero. Unordered–The value assigned a floating-point receiver operand is NaN.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X		
06 Floating-point overflow	X			
07 Floating-point underflow	X			
09 Floating-point invalid operand		X		X
0A Size	X			
0C Invalid floating-point conversion	X			
0D Floating-point inexact result	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target				X
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X
32 Scalar Specification				
03 Scalar value invalid			X	

SCAN (SCAN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10D3	Receiver	Base	Compare operand

Operand 1: Binary variable scalar or binary array.

Operand 2: Character scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SCANI	18D3	Indicator
SCANB	1CD3	Branch

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator targets (for indicator options). The branch or indicator targets immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The character string value of the base operand is scanned for occurrences of the character string value of the compare operand.

The base and compare operands must both be character strings. The length of the compare operand must not be greater than that of the base string.

The operation begins at the left end of the base string and continues character by character, from left to right, comparing the characters of the base string with those of the compare operand. The length of the comparisons are equal to the length of the compare operand value and function as if they were being compared in the Compare Bytes Left-Adjusted instruction.

If a set of bytes that match the compare operand is found, the binary value for the relative location of its leftmost base string character is placed in the receiver operand.

If the receiver operand is a scalar, only the first occurrence of the compare operand is noted. If it is an array, as many occurrences as there are elements in the array are noted.

The operation continues until no more occurrences of the compare operand can be noted in the receiver operand or until the number of characters (bytes) remaining to be scanned in the base string is less than the length of the compare operand.

When the second condition occurs, the receiver value is set to 0. If the receiver operand is an array, all its remaining elements are also set to 0.

The base operand and the compare operand can be variable length substring compound operands.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 2 and 3. The effect of specifying a null substring reference for the compare operand or both operands is that the receiver is set to zero (no match found) and the instruction's·resultant condition is null compare operand. Specifying a null substring reference for just the base operand is not allowed due to the requirement that the length of the compare operand must not be greater than that of the base string.

Resultant Conditions: Zero or positive–The numeric value(s) of the receiver operand is either zero or positive. When the receiver operand is an array, the resultant condition is zero if all elements are zero. One of these two conditions will result when the compare operand is not a null substring reference. Null compare operand–The compare operand is a null substring reference; therefore, the receiver has been set to zero which indicates that no occurrences were found.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process control limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operands 1	 2	 3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
08 Length conformance		X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target				X
0A Invalid operand length		X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

SCAN WITH CONTROL (SCANWC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
10E4	Base locator	Controls	Options	Escape target or null

Operand 1: Space pointer.

Operand 2: Character(8) variable scalar.

Operand 3: Character(4) scalar.

Operand 4: Instruction number, relative instruction number, branch point, instruction pointer, instruction definition list element, or null.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SCANWC	10E4	Short
SCANWCI	18E4	Indicator
SCANWCB	1C84	Branch

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The base string to be scanned is specified by the base locator and controls operands. The base locator addresses first character of the base string. The controls specifies the length of the base string in the base length field.

The scan operation begins at the left end of the base string and continues character by character, left-to-right. The scan operation can be performed on a base string which contains all simple (1-byte) or all extended (2-byte) character codes or a mixture of the two. When the base string is being interpreted in simple character mode, the operation moves through the base string one byte at a time. When the base string is being interpreted in extended character mode, the operation moves through the base string 2 bytes at a time. The character string value of the base operand is scanned for occurrences of a character value satisfying the criteria specified in the control and options operands.

The scan is completed by updating the base locator and controls operands with scan status when a character value being scanned for is found, the end of the base string is encountered, or an escape code is encountered when the escape target operand is specified. The base locator is set with addressability to the character (simple or extended) which caused the instruction to complete execution. The controls operand is set with information which identifies the mode (simple or extended) of the base string character addressed by the base locator and which provides for resumption of the scan operation with minimal overhead.

The controls and options operands specify the modes to be used in interpreting characters during the scan operation. Characters can be interpreted in one of two character modes: simple (1-byte) and extended (2-byte). Additionally, the base string can be scanned in one of two scan modes, mixed (base string may contain a mixture of both character modes) and nonmixed (base string contains one mode of characters).

When the mixed scan mode is specified in the options operand, the base string is interpreted as containing a mixture of simple and extended character codes. The mode, simple or extended, with which the string is to be interpreted, is controlled initially by the base mode indicator in the controls operand and thereafter by mode control characters imbedded in the base string. The mode control characters are as follows:

- Hex 0E = Shift out (SO) of simple character mode to extended mode.

- Hex 0F = Shift in (SI) to simple character mode from extended mode. This is only recognized if it occurs in the first byte position of an extended character code.

When the nonmixed scan mode is specified in the options operand, the base string is interpreted using only the character mode specified by the base mode indicator in the controls operand. Character mode shifting can not occur because no mode control characters are recognized when scanning in nonmixed mode.

The base locator operand is a space pointer which is both input to and output from the instruction. On input, it locates the first character of the base string to be processed. On output, it locates the character of the base string which caused the instruction to complete.

The controls operand must be a character scalar which specifies additional information to be used to control the scan operation. It must be at least 8 bytes long and have the following format:

• Controls operand	Char(8)
– Control indicators	Char(1)
– Reserved	Char(1)
– Comparison characters	Char(2)
– Reserved	Char(1)
– Base end	Char(3)
— Instruction work area	Char(1)
— Base length	Char(2)

Only the first 8 bytes of the controls operand are used. Any excess bytes are ignored. Reserved fields must contain binary zeros.

The control indicators field has the following format:

• Control indicators	Char(1)
– Base mode 0 = Simple 1 = Extended	Bit 0
– Comparison character mode 0 = Simple 1 = Extended	Bit 1
– Reserved (must be 0)	Bit 2-6
– Scan state 0 = Resume scan 1 = Start scan	Bit 7

The base mode is both input to and output from the instruction. In either case, it specifies the mode of the character in the base string currently addressed by the base locator.

The comparison character mode is not changed by the instruction. It specifies the mode of the comparison character contained in the controls operand.

The scan state is both input to and output from the instruction. As input, it indicates whether the scan operation for the base string is being started or resumed. If it is being started, the instruction assumes that the base length value in the base end field of the controls operand specifies the length of the base string, and the instruction work area value is ignored. If it is being resumed, the instruction assumes the base end field has been set by a prior start scan execution of the instruction with an internal machine value identifying the end of the base string.

For a start scan execution of the instruction, the scan state indicator is reset to indicate resume scan to provide for subsequent resumption of the scan operation. Additionally, for a start scan execution of the instruction, the base end field is set with an internally optimized value which identifies the end of the base string being scanned. This value then overlays the values which were in the instruction work area and base length fields on input to the instruction. Predictable operation of the instruction on a resume scan execution depends upon this base end field being left intact with the value set by the start scan execution.

For a resume scan execution of the instruction, the scan state and base end fields are unchanged.

The comparison character is input to the instruction. It specifies a character code to be used in the comparisons performed during the scanning of the base string. The comparison character mode in the control indicators specifies the mode (simple or extended) of the comparison character. If it is a simple character, the first byte of the comparison character field is ignored and the comparison character is assumed to be specified in the second byte. If it is an extended character, the comparison character is specified as a 2-byte value in the comparison character field.

The base end field is both input to and output from the instruction. It contains data which identifies the end of the base string. Initially, for a start scan execution of the instruction, it contains the length of the base string in the base length field. Additionally, the base end field is used to retain information over multiple instruction executions which provides for minimizing the overhead required to resume the scan operation for a particular base string. This information is set on the initial start scan execution of the instruction and is used during subsequent resume scan executions of the instruction to determine the end of the base string to be scanned. If the end of the base string being scanned must be altered during iterative usage of this instruction, a start scan execution of the instruction must be performed to provide for correctly resetting the internally optimized value to be stored in the base end from the values specified in the base locator operand and base length field.

For the special case of a start scan execution where a length value of zero (no characters to scan) is specified in the base length field, the instruction results in a not found resultant condition. In this case, the base locator is not verified and the scan state indicator, the base end field, and the base locator are not changed.

The options operand must be a character scalar which specifies the options to be used to control the scan operation. It must be at least 4 bytes in length and has the following format:

• Options operand	Char(4)
– Options indicators	Char(1)
– Reserved	Char(3)

The options operand must be specified as a constant character scalar.

Only the first 4 bytes of the options operand are used. Any excess bytes are ignored. Reserved fields must contain binary zeros.

The option indicators field has the following format:

• Option indicators	Char(1)
– Reserved	Bit 0
– Scan mode 0 = Mixed 1 = Nonmixed	Bit 1
– Reserved	Bit 2-3
– Comparison relation	Bit 4-6
— Equal, (=) relation	Bit 4
— Less than, (<) relation	Bit 5
— Greater than, (>) relation 0 = No match on relation 1 = Match on relation	Bit 6
– Reserved	Bit 7

The scan mode specifies whether the base string contains a mixture of character modes, or contains all one mode of characters; that is, whether or not mode control characters should be recognized in the base string. Mixed specifies that there is a mixture of character modes and, therefore, mode control characters should be recognized. Nonmixed specifies that there is not a mixture of character modes and, therefore, mode control characters should not be recognized. Note that the base mode indicator in the controls operand specifies the character mode of the base string character addressed by the base locator.

The comparison relation specifies the relation or relations of the comparison character to characters of the base string which will satisfy the scan operation and cause completion of the instruction with one of the height, low, or equal resultant conditions. Multiple relations may be specified in conjunction with one another. Specifying all relations insures a match against any character in the base string which is of the same mode as the comparison character. Specifying no relation insures a not found resultant condition, in the absence of an escape due to verification, regardless of the values of the characters in the base string which match the mode of the comparison character.

An example of comparison scanning is a scan of simple mode characters for a value less than hex 40. This could be done by specifying a comparison character of hex 40 and a comparison relation of greater than in conjunction with a branch option for the resultant condition of high. This could also be done by specifying a comparison character of hex 3F and comparison relations of equal and greater than in conjunction with branch options for equal and high. The target of the branch options in either case would be the instructions to process the character less than hex 40 in value.

The escape target operand controls the verification of bytes of the base string for values less than hex 40. Verification, if requested, is always performed in conjunction with whatever comparison processing has been requested. That is, verification is performed even if no comparison relation is specified. This operand is discussed in more detail in the following material.

During the scan operation, the characters of the base string which are not of the same mode as the comparison character are skipped over until the mode of the characters being processed is the same as the mode of the comparison character. The operation then proceeds by comparing the comparison character with each of the characters of the base string. These comparisons behave as if the characters were being compared in the Compare Bytes Left Adjusted instruction.

If a base string character satisfying the criteria specified in the controls and options operands is found, the base locator is set to address the first byte of it, the base mode indicator is set to indicate the mode of the base string as of that character, and the instruction is completed with the appropriate resultant condition based on the relation (high, low, or equal) of the comparison character to the base string character.

If a matching base string character is not found prior to encountering a mode change, the characters of the base string are again skipped over until the mode of the characters being processed is the same as the mode of the comparison character before comparisons are resumed.

If a matching base string character is not found prior to encountering the end of the base string, the base location is set to address the first byte of the character encountered at the end of the base string, the base mode indicator is set to indicate the mode of the base string as of that character, and the instruction is completed with the not found resultant condition. A mode control string results in the changing of the base string mode, but the base locator is left addressing the mode control character.

If the escape target operand is specified (operand 4 is not null), verifications are performed on the characters of the base string prior to their being skipped or compared with the comparison character. Each byte of the base string is checked for a value less than hex 40. Additionally, for a mixed scan mode, when such a value is encountered, it is then determined if it is a valid mode control character.

- Hex 0E (SO) when the base string is being interpreted in simple character mode.

- Hex 0F (SI) in the left byte of the character code when the base string is being interpreted in extended character mode.

If a byte value of less than hex 40 is not a valid mode control character, it is considered to be an escape code. The base locator is set to address the first byte of the base string character (simple or extended) which contains the escape code, the base mode indicator is set to indicate the mode of the base string as of that character, and a branch is taken to the target specified by the escape target operand. When the escape target branch is performed, the value of any optional indicator operands is meaningless.

If the escape target operand is not specified (operand 4 is null), verifications of the character codes in the base string are not performed. However, for a mixed scan mode, mode control values are always processed as described previously under the discussion of the mixed scan mode.

Substring operand references which allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Variable length substring compound operands may not be specified for operands two and three.

If possible, use a Space Pointer Machine Object for the base locator, operand 1. Appreciably less overhead is incurred in accessing and storing the value of the base locator if this is done.

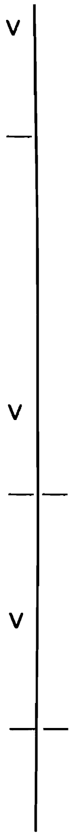
If possible, specify through its ODT definition, the controls operand on an 8-byte multiple (doubleword) boundary relative to the start of the space containing it. Appreciably less overhead is incurred in accessing and storing the value of the controls if this is done.

For the case where a base string is to be just scanned for byte values less than hex 40, two techniques can be used.

- A direct simple mode scan for a value less than hex 40, without usage of the escape target verification feature.
- A scan for any character with usage of the escape target verification feature.

  The direct scan approach, the former, is the more efficient.

The following diagram defines the various conditions which can be encountered at the end of the base string and what the base locator addressability is in each case. The solid vertical line represents the end of the base string. The dashes represent the bytes before and after the base string end. The V is positioned over the byte addressed by the base locator in each case. These are the conditions which can be encountered when the base locator input to the instruction addresses a byte prior to the base string end. When the base length field specifies a value of zero for a start scan execution of the instruction, or the input base locator addresses a point beyond the end of the instruction, no processing is performed and the instruction is immediately completed with the not found resultant condition.



Addressability	Ending Condition	Instruction Response
	(One byte code at string end)	
	• Simple character	• Appropriate resultant condition indicating found or not found
	• Shift in/out encountered	• Mode shift performed, and not found resultant condition
	• Escape code in simple character	• Branch taken
	(Extended character split across string end)	
	• Extended character	• Not found resultant condition
	• Escape code in extended character	• Branch taken
	(Extended character at string end)	
	• Extended character	• Appropriate resultant condition indicating found or not found
	• Escape code in extended character	• Branch taken

An analysis of the diagram shows that normally, after appropriate processing for the particular found, not found, or escape condition, the scan can be restarted at the byte of data which would follow the base string end in the data stream being scanned. Any mode shift required by an ending mode control character will have been performed.

However, one ending condition may require subsequent resumption of the scan at the character encountered at the end of the base string. This is the case where the instruction completes with the not found resultant condition and the base string ends with an extended character split across string end. That is, the base mode indicator specifies extended mode, the base locator addresses the last byte of the base string, and that byte value is not a shift out, hex 0E character. In this case, complete verification of the extended character and relation comparison could not be performed. If this extended character is to be processed, it must be done through another execution of the Scan instruction where both bytes of the character can be input to the instruction within the confines of the base string.

Resultant Conditions

- High, Low, Equal: A character value was found in the base string which satisfies the criteria specified in the controls and options operands in that the comparison character is of higher, lower, or equal string value to the base string character.
- Not found: A character value was not found in the base string which satisfied the criteria specified in the controls and options operands.

Events

000C Machine resource
0201 Machine auxiliary storage exceeded

000D Machine status
0101 Machine check

0010 Process
0601 Exception signaled to process
0701 Maximum processor time exceeded

0016 Machine observation
0101 Instruction reference
0201 Object location reference

0017 Damage set
0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment violation	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0C Computation					
08 Length conformance		X	X		
10 Damage Encountered					
44 Partial system object damage					X
1C Machine Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op-code extender field					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand				X	X
0A Invalid operand length		X	X		
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2C Program Execution					
04 Branch target invalid					X
32 Scalar Specification					
01 Scalar type invalid	X	X	X	X	
03 Scalar value invalid		X	X		

SEARCH (SEARCH)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
1084	Receiver	Array	Find	Location

Operand 1: Binary variable scalar or binary variable array.

Operand 2: Character array or numeric array.

Operand 3: Character scalar or numeric scalar.

Operand 4: Binary scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SEARCHI	1884	Indicator
SEARCHB	1C84	Branch

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator targets (for indicator options). The branch or indicator targets immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The portions of the array operand indicated by the location operand are searched for occurrences of the value indicated in the find operand.

The operation begins with the first element of the array operand and continues element by element, comparing those characters of each element (beginning with the character indicated in the location operand) with the characters of the find operand. The location operand contains an integer value representing the relative location of the first character in each element to be used to begin the compare.

The integer value of the location operand must range from 1 to L, where L is the length of the array operand elements. A value of 1 indicates the leftmost character of each element.

The array and find operands can be either character or numeric. Any numeric operands are interpreted as logical character strings. The compares between these operands are performed at the length of the find operand and function as if they were being compared in the Compare Bytes Left-Adjusted instruction.

The length of the find operand must not be so large that it exceeds the length of the array operand elements when used with the location operand value. The array element length used is the length of the array scalar elements and not the length of the entire array element, which can be larger in noncontiguous arrays.

As each occurrence of the find value is encountered, the integer value of the index for this array element is placed in the receiver operand. If the receiver operand is a scalar, only the first element containing the find value is noted. If the receiver operand is an array, as many occurrences as there are elements within the receiver array are noted.

The operation continues until no more occurrences of elements containing the find value can be noted in the receiver operand or until the array operand has been completely searched. When the second condition occurs, the receiver value is set to 0. If the receiver operand is an array, all its remaining elements are also set to 0. The find operand can be a variable length substring compound operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: The numeric value(s) of the receiver operand is either 0 or positive. When the receiver operand is an array, the resultant condition is 0 if all elements are 0.

Events

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

Exception	Operands				Other
	1	**2**	**3**	**4**	
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0C Computation					
08 Length conformance			X	X	
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op code extender field					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand					X
0A Invalid branch length			X		
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2C Program Execution					
04 Branch target invalid					X
32 Scalar Specification					
01 Scalar type invalid	X	X	X	X	
0A Invalid operand length	X	X	X	X	

SET INSTRUCTION POINTER (SETIP)

Op Code (Hex)	Operand 1	Operand 2
1022	Receiver	Branch target

Operand 1: Instruction pointer.

Operand 2: Instruction number, relative instruction number, or branch point.

Description: The value of the branch target (operand 2) is used to set the value of the instruction pointer specified by operand 1. The instruction number indicated by the branch target must provide the address of an instruction within the program containing the Set Instruction Pointer instruction.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X		
02 Boundary alignment	X		
03 Range	X		
06 Optimized addressability invalid	X		
08 Argument/Parameter			
01 Parameter reference violation	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X		
02 Object destroyed	X		
03 Object suspended	X		
24 Pointer Specification			
01 Pointer does not exist	X		
02 Pointer type invalid	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X		
08 Invalid operand value range	X		
09 Invalid branch target operand		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
04 Branch target invalid		X	

STORE AND SET COMPUTATIONAL ATTRIBUTES (SSCA)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
107B	Receiver	Source	Controls

Operand 1: Character(5) variable scalar.

Operand 2: Character(5) scalar or null.

Operand 3: Character(5) scalar or null.

Description: This instruction stores and optionally sets the attributes for controlling computational operations for the process this instruction is executed in.

The receiver is assigned the values that each of the computational attributes had at the start of execution of the instruction. It has the same format and bit assignment as the source.

The source specifies new values for the computational attributes for the process. The particular computational attributes that are selected for modification are determined by the controls operand. The source operand has the following format:

• Floating-point exception masks	Char(2)
0 = Disabled (exception is masked)	
1 = Enabled (exception is unmasked)	
– Reserved (binary 0)	Bits 0-9
– Floating-point overflow	Bit 10
– Floating-point underflow	Bit 11
– Floating-point zero divide	Bit 12
– Floating-point inexact result	Bit 13
– Floating-point invalid operand	Bit 14
– Reserved (binary 0)	Bit 15
• Floating-point exception occurrence flags	Char(2)
0 = Exception has not occurred	
1 = Exception has occurred	
– Reserved (binary 0)	Bits 0-9
– Floating-point overflow	Bit 10
– Floating-point underflow	Bit 11
– Floating-point zero divide	Bit 12
– Floating-point inexact result	Bit 13
– Floating-point invalid operand	Bit 14
– Reserved (binary 0)	Bit 15

- Modes Char(1)
  - Reserved Bit 0
  - Floating-point rounding mode Bits 1-2
    - 00 = Round toward positive infinity
    - 01 = Round toward negative infinity
    - 10 = Round toward zero
    - 11 = Round to nearest (default)
  - Reserved Bits 3-7

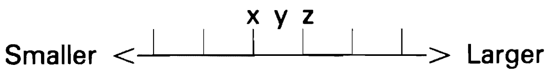
The controls operand is used to select those attributes that are to be set from the bit values of the source operand. The format of the controls is the same as that for the source. A value of one for a bit in controls indicates that the corresponding computational attribute for the process is to be set from the value of that bit of the source. A value of zero for a bit in controls indicates that the corresponding computational attribute for the process is not to be changed, and will retain the value it had prior to this instruction. For an attribute controlled by a multiple-bit field, such as the rounding modes, all of the bits in the field must be ones or all must be zeros. A mixture of ones and zeros in such a field results in a scalar value invalid exception.

If the source and controls operands are both null, the instruction will just return the current computational attributes. If the source is specified, the computational attributes of the process are modified under control of the controls operand. If the source operand is specified and the controls operand is null, the instruction will change all of the computational attributes to the values specified in the source. If the source operand is null and the controls operand is specified, an invalid operand type exception is signaled.

With the floating-point exception masks field, it is possible to unmask/mask the exception processing and handling for each of the five floating-point exceptions. If an exception that is unmasked occurs, then the corresponding exception occurrence bit is set, and the exception is signaled. If an exception that is masked occurs, the exception is not signaled, but the exception occurrence flag is still set to indicate the occurrence of the exception.

The floating-point exception occurrence flag for each exception may be set or cleared by this instruction from the source operand under control of the controls operand.

Unless specified otherwise by a particular instruction, or precluded due to implicit conversions, all floating-point operations are performed as if correct to infinite precision, and then rounded to fit in a destinations format while potentially signaling an exception that the result is inexact. To allow control of the floating-point rounding operations performed within a process, four floating-point rounding modes are supported. Assume y is the infinitely precise number that is to be rounded, bracketed most closely by x and z, where x is the largest representable value less than y and z is the smallest representable value greater than y. Note that x or z may be infinity. The following diagram shows this relationship of x, y, and z on a scale of numerically progressing values where the vertical bars denote values representable in a floating-point format.



Given the above, if y is not exactly representable in the receiving field format, the rounding modes change y as follows:

Round to nearest with round to even in case of a tie is the default rounding mode in effect upon the initiation of a process. For this rounding mode, y is rounded to the closer of x or z. If they are equally close, the even one (the one whose least significant bit is a zero) is chosen. For the purposes of this mode of rounding, infinity is treated as if it was even. Except for the case of y being rounded to a value of infinity, the rounded result will differ from the infinitely precise result by at most half of the least significant digit position of the chosen value. This rounding mode differs slightly from the decimal round algorithm performed for the optional round form of an instruction. This rounding mode would round a value of 0.5 to 0, where the decimal round algorithm would round that value to 1.

Round toward positive infinity indicates directed rounding upward is to occur. For this mode, y is rounded to z.

Round toward negative infinity indicates directed rounding downward is to occur. For this mode, y is rounded to x.

Round toward zero indicates truncation is to occur. For this mode, y is rounded to the smaller (in magnitude) of x or z.

Arithmetic operations upon infinity are exact. Negative infinity is less than every finite value, which is less than positive infinity.

The computational attributes are set with a default value upon process initiation. The default attributes are as follows:

- The floating-point inexact result exception is masked. The other floating-point exceptions are unmasked.
- All occurrence bits have a zero value.
- Round to the nearest rounding mode.

These attributes can be modified by a program executing this instruction. The new attributes are then in effect for the program executing this instruction and for programs invoked subsequent to it unless changed through another execution of this instruction. External exception handlers and invocation exit routines are invoked with the same attributes as were last in effect for the program invocation they are related to. Event handlers do not really relate to another invocation in the process. As such, they are invoked with the attributes that were in effect at the point the process was interrupted to handle the event.

Upon return to the invocation of a program from subsequent program invocations, the computational attributes, other than exception occurrence attributes, are restored to those that were in effect when the program gave up control. The exception occurrence attributes are left intact reflecting the occurrence of any floating-point exceptions during the execution of subsequent invocations.

Internal exception handlers execute under the invocation of the program containing them. As such, the above discussion of how computational attributes are restored upon returning from an external exception handler does not apply. The execution of an internal exception handler occurs in a manner similar to the execution of an internal subroutine invoked through the Call Internal instruction. If the internal exception handler modifies the attributes, the modification remains in effect for that invocation when the exception handler completes the exception.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

000D Machine status
0101 Machine check

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
03 Scalar value invalid		X	X	

SUBTRACT LOGICAL CHARACTER (SUBLC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1027	Difference	Minuend	Subtrahend

Operand 1: Character variable scalar (fixed-length).

Operand 2: Character scalar (fixed-length).

Operand 3: Character scalar (fixed-length).

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SUBLCS	1127	Short
SUBLCI	1827	Indicator
SUBLCIS	1927	Indicator, Short
SUBLCB	1C27	Branch
SUBLCBS	1D27	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The unsigned binary value of the subtrahend operand is subtracted from the unsigned binary value of the minuend operand, and the result is placed in the difference operand.

Operands 1, 2, and 3 must be the same length; otherwise, the Create Program instruction signals an invalid length exception.

The subtraction operation is performed as though the ones complement of the second operand and a low-order 1-bit were added to the first operand.

The result value is then placed (left-adjusted) into the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a byte value of hex 00.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Resultant Conditions: The logical difference of the character scalar operands is zero with carry out of the high-order bit position, not-zero with carry, or not-zero with no carry.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target				X
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid	X	X	X	

SUBTRACT NUMERIC (SUBN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1047	Difference	Minuend	Subtrahend

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SUBNS	1147	Short
SUBNR	1247	Round
SUBNSR	1347	Short, Round
SUBNB	1C47	Branch
SUBNBS	1D47	Branch, Short
SUBNBR	1E47	Branch, Round
SUBNBSR	1F47	Branch, Short, Round
SUBNI	1847	Indicator
SUBNIS	1947	Indicator, Short
SUBNIR	1A47	Indicator, Round
SUBNISR	1B47	Indicator, Short, Round

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The signed numeric value of the subtrahend operand is subtracted from the numeric value of the minuend operand, and the result is placed in the difference operand.

The operands must be numeric with any implicit conversions occurring according to the rules of arithmetic operations as outlined in the *Functional Concepts Manual.*

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

For a decimal operation, alignment of the assumed decimal point takes place by padding with 0's on the right end of the source operand with lesser precision.

Floating-point subtraction uses exponent comparison and significand subtraction. Alignment of the binary point is performed, if necessary, by shifting the significand of the value with the smaller exponent to the right. The exponent is increased by one for each binary digit shifted until the two exponents agree.

The operation uses the length and the precision of the source and receiver operands to calculate accurate results.

The subtract operation is performed according to the rules of algebra.

The result of the operation is copied into the difference operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the difference operand, aligned at the assumed decimal point of the difference operand, or both before being copied to it. Length adjustment and decimal point alignment are performed according to the rules of arithmetic operations outlined in the *Functional Concepts Manual*. For fixed-point operation, if significant digits are truncated on the left end of the resultant value, a size exception is signaled.

For the optional round form of the instruction, specification of a floating-point receiver operand is invalid.

For floating-point operations involving a fixed-point receiver field, if nonzero digits would be truncated off the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point difference operand, if the exponent of the resultant value is either too large or too small to be represented in the difference field, the floating-point overflow or the floating-point underflow exception is signaled.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: Positive, negative, or zero–The algebraic value of the numeric scalar difference is positive, negative, or zero. Unordered–The value assigned a floating-point difference operand is NaN.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3 [4, 5]**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
03 Decimal point alignment		X	X	
06 Floating-point overflow	X			
07 Floating-point underflow	X			
09 Floating-point invalid operand		X	X	X
0A Size	X			
0C Invalid floating-point conversion	X			
0D Floating-point inexact result	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target				X
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

TEST AND REPLACE CHARACTERS (TSTRPLC)

Op Code (Hex)	Operand 1	Operand 2
10A2	Receiver	Replacement

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Description: The character string value represented by operand 1 is tested byte by byte from left to right. Any byte to the left of the leftmost byte which has a value in the range of hex F1 to hex F9 is assigned a byte value equal to the leftmost byte of operand 2.

Both operands must be character strings. Only the first character of the replacement string is used in the operation.

The operation stops when the first nonzero zoned decimal digit is found or when all characters of the receiver operand have been replaced.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

TEST BITS UNDER MASK (TSTBUMB or TSTBUMI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4, 5]
1C2A	Branch options	Source	Mask	Branch target
182A	Indicator options			Indicator target

Operand 1: Character scalar or numeric scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3 [*4, 5*]:

- *Branch Form*—Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*—Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: Selected bits from the leftmost byte of the source operand are tested to determine their bit values.

Based on the test, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The source and the mask operands can be character or numeric. The leftmost byte of each of the operands is used in the operands. The operands are interpreted as bit strings.

The testing is performed bit by bit with only those bits indicated by the mask operand being tested. A 1-bit in the mask operand specifies that the corresponding bit in the source value is to be tested. A 0-bit in the mask operand specifies that the corresponding bit in the source value is to be ignored.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: The selected bits of the bit string source operand are all zeros, all ones, or mixed ones and zeros. A mask operand of all zeros causes a zero resultant condition.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3 [4, 5]**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target				X
0A Invalid operand length	X	X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid			X	

TRANSLATE (XLATE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
1094	Receiver	Source	Position	Replacement

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar or null.

Operand 4: Character scalar.

Description: Selected characters in the string value of the source operand are translated into a different encoding and placed in the receiver operand. The characters selected for translation and the character values they are translated to are indicated by entries in the position and replacement strings.

All the operands must be character strings. The source and receiver values must be of the same length. The position and replacement operands can differ in length. If operand 3 is null, a 256-character string is used, ranging in value from hex 00 to hex FF (EBCDIC collating sequence).

The operation begins with all the operands left-adjusted and proceeds character by character, from left to right until the character string value of the receiver operand is completed.

Each character of the source operand value is compared with the individual characters in the position operand. If a character of equal value does not exist in the position string, the source character is placed unchanged in the receiver operand. If a character of equal value is found in the position string, the corresponding character in the same relative location within the replacement string is placed in the receiver operand as the source character translated value. If the replacement string is shorter than the position string and a match of a source to position string character occurs for which there is no corresponding replacement character, the source character is placed unchanged in the receiver operand. If the replacement string is longer than the position string, the rightmost excess characters of the replacement string are not used in the translation operation because they have no corresponding position string characters. If a character in the position string is duplicated, the first occurrence (leftmost) is used.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

The receiver, source, position, and replacement operands can be variable length substring compound operands.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for all of the operands on this instruction. The effect of specifying a null substring reference for either the position or replacement operands is that the source operand is copied to the receiver with no change in value. The effect of specifying a null substring reference for both the receiver and the source operands (they must have the same length) is that no result is set.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0C Computation					
08 Length conformance	X	X			
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
0A Invalid operand length	X	X	X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X

TRANSLATE WITH TABLE (XLATEWT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
109F	Receiver	Source	Table

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar.

Description: The source characters are translated under control of the translate table and placed in the receiver. The operation begins with the leftmost character of operand 2 and proceeds character-by-character, left-to-right.

Characters are translated as follows:

- The source character is used as an offset and added to the location of operand 3.

- The character contained in the offset location is the translated character. This character is copied to the receiver in the same relative position as the original character in the source string.

If operand 3 is less than 256 bytes long, the character in the source may specify an offset beyond the end of operand 3.

If operand 2 is longer than operand 1, then only the leftmost portion of operand 2, equal to the length of operand 1, is translated. If operand 2 is shorter than operand 1, then only the leftmost portion of operand 1, equal to the length of operand 2, is changed. The remaining portion of operand 1 is unchanged.

If operand 1 overlaps with operand 2 and/or 3, the overlapped operands are updated for every character translated. The operation proceeds from left to right, one character at a time. The following example shows the results of an overlapped operands translate operation. Operands 1, 2, and 3 have the same coincident character string with a value of hex 050403020103.

Hex 050403020103–Initial value
Hex 030403020103–After the 1st character is
translated
Hex 030103020103–After the 2nd character is
translated
Hex 030102020103–After the 3rd character is
translated
Hex 030102020103–After the 4th character is
translated
Hex 030102020103–After the 4th character is
translated
Hex 030102020102–After the 5th character, the final
result

Note that the instruction does not use the length specifi table operand to constrain access of the bytes addresse table operand.

If operand 3 is less than 256 characters long, and a sour character specifies an offset beyond
the end of operand 3, the result characters are obtained from byte locations in the space following
operand 3. If that portion of the space is not currently allocated, a space addressing exception
is signaled. If operand 3 is a constant with a length less than 256, source characters
specifying offsets greater than or equal to the length of the constant are translated into unpredictable characters.

All of the operands support variable length substring co

Substring operand references that allow for a null substr reference (a length value of zero) may be specified for al operands on this instruction. Specifying a null substring for the table operand does not affect the operation of th instruction. In this case, the bytes addressed by the tabl are still accessed as described above. This is due to the of the function of this instruction which does not use th specified for the table operand to constrain access of th addressed by the table operand. The effect of specifyin substring reference for either or both of the receiver and source operands is that no result is set.

Events

000C Machine resources
0201 Machine auxiliary storage exceeded

000D Machine status
0101 Machine check

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	

TRIM LENGTH (TRIML)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10A7	Receiver length	Source string	Trim character

Operand 1: Numeric variable scalar.

Operand 2: Character scalar.

Operand 3: Character(1) scalar.

Description: The operation determines the resultant length of operand 2 after the character specified by operand 3 has been trimmed from the end of operand 2. The resulting length is stored in operand 1.

Operand 2 is trimmed from the end as follows: if the rightmost (last) character of operand 2 is equal to the character specified by operand 3, the length of the trimmed operand 2 string is reduced by 1. This operation continues until the rightmost character is no longer equal to operand 3 or the trimmed length is zero. If operand 3 is longer than one character, only the first (leftmost) character is used as the trim character.

Operands 2 and 3 are not changed by this instruction. Operand 2 or 3 may be variable length substring compound scalars.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
0A Size	X			
10 Damage Encountered				
44 Partial system object damage				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	

VERIFY (VERIFY)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10D7	Receiver	Source	Class

Operand 1: Binary variable scalar or binary array.

Operand 2: Character scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
VERIFYI	18D7	Indicator
VERIFYB	1CD7	Branch

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See *Chapter 1. Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: Each character of the source operand character string value is checked to verify that it is among the valid characters indicated in the class operand.

The operation begins at the left end of the source string and continues character by character, from left to right. Each character of the source value is compared with the characters of the class operand. If a match for the source character exists in the class string, the next source character is verified. If a match for the source character does not exist in the class string, the binary value for the relative location of the character within the source string is placed in the receiver operand.

If the receiver operand is a scalar, only the first occurrence of an invalid character is noted. If the receiver operand is an array, as many occurrences as there are elements in the array are noted.

The operation continues until no more occurrences of invalid characters can be noted or until the end of the source string is encountered. When the second condition occurs, the current receiver value is set to 0. If the receiver operand is an array, all its remaining entries are set to 0's.

The source and class operands can be variable length substring compound operands.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 2 and 3. The effect of specifying a null substring reference for the class operand when a nonnull string reference is specified for the source is that all of the characters of the source are considered invalid. In this case, the receiver is accordingly set with the offset(s) to the bytes of the source, and the instruction's resultant condition is positive. The effect of specifying a null substring reference for the source operand (no characters to verify) is that the receiver is set to zero and the instruction's resultant condition is zero regardless of what is specified for the class operand.

Resultant Conditions: The numeric value(s) of the receiver is either 0 or positive. When the receiver operand is an array, the resultant condition is 0 if all elements are 0.

Events

000C Machine resource
 0201 Machine auxiliary storage threshold exceeded

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

	Operands			
Exception	1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand				X
0A Invalid operand length		X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Branch target invalid				X

Chapter 3. Pointer/Name Resolution Addressing Instructions

This chapter describes the instructions used for pointer and name resolution functions. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

COMPARE POINTER FOR OBJECT ADDRESSABILITY (CMPPTRAB or CMPPTRAI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4]
1CD2	Branch options	Compare operand 1	Compare operand 2	Branch target
18D2	Indicator options			Indicator target

Operand 1: Data pointer, space pointer, system pointer, or instruction pointer.

Operand 2: Data pointer, space pointer, system pointer, or instruction pointer.

Operand 3 [*4*]:

- *Branch Form*—Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*—Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction.

The extender field is required along with one or two branch targets (for branch option) or one or two indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operand 4. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The object addressed by operand 1 is compared with the object addressed by operand 2 to determine if both operands are addressing the same object. Based on the comparison, the resulting condition is used with the extender to transfer control (branch form) or to assign a value to each of the indicator operands (indicator form).

If operand 1 is a data pointer, a space pointer, or a system pointer, operand 2 may be any pointer type except for instruction pointer in any combination. An equal condition occurs if the pointers are addressing the same object. For space pointers and data pointers, only the space they are addressing is considered in the comparison. That is, the space offset portion of the pointer is ignored.

For system pointer compare operands, an equal condition occurs if the system pointer is compared with a space pointer or data pointer that addresses the space that is associated with the object that is addressed by the system pointer. For example, a space pointer that addresses a byte in a space associated with a system object compares equal with a system pointer that addresses the system object.

For instruction pointer comparisons, both operands must be instruction pointers; otherwise, an invalid pointer type exception is signaled. An equal condition occurs when both instruction pointers are addressing the same instruction in the same program. A not equal condition occurs if the instruction pointers are not addressing the same instruction in the same program.

A pointer does not exist exception is signaled if a pointer does not exist in either of the operands.

Resultant Conditions: Equal, not equal.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0A Authorization					
01 Unauthorized for operation	X	X			
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state	X	X			
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op code extender field					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand			X	X	
0A Invalid operand length			X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X

COMPARE POINTER TYPE (CMPPTRTB or CMPPTRTI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4]
1CE2	Branch options	Compare operand 1	Compare operand 2	Branch target
18E2	Indicator options			Indicator target

Operand 1: Data pointer, space pointer, system pointer, or instruction pointer.

Operand 2: Character(1) scalar or null.

Operand 3 [*4*]:

- *Branch Form*–Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*–Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction.

The extender field is required along with one or two branch targets (for branch option) or one or two indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operand 4. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The instruction compares the pointer type currently in operand 1 with the character scalar identified by operand 2. Based on the comparison, the resulting condition is used with the extender to transfer control (branch form) or to assign a value to each of the indicator operands (indicator form).

Operand 1 can specify a space pointer machine object only when operand 2 is null.

If operand 2 is null or if operand 2 specifies a comparison value of hex 00, an equal condition occurs if a pointer does not exist in the storage area identified by operand 1.

Following are the allowable values for operand 2:

Hex 00 – A pointer does not exist at this location
Hex 01 – System pointer
Hex 02 – Space pointer
Hex 03 – Data pointer
Hex 04 – Instruction pointer

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Equal, not equal.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0A Authorization					
01 Unauthorized for operation	X				
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state	X				
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
05 Invalid op code extender operand					X
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
09 Invalid branch target operand			X	X	
0A Invalid operand length		X	X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
03 Scalar value invalid			X		

COPY BYTES WITH POINTERS (CPYBWP)

Op Code (Hex)	Operand 1	Operand 2
0132	Receiver	Source

Operand 1: Character variable scalar, space pointer, data pointer, system pointer, or instruction pointer.

Operand 2: Character variable scalar, space pointer data object, data pointer, system pointer, instruction pointer, or null.

Description: This instruction copies either the pointer value or the byte string specified for the source operand into the receiver operand depending upon whether or not a space pointer machine object is specified as one of the operands.

Operations involving space pointer machine objects perform a pointer value copy operation for only space pointer values or the pointer does not exist state. Due to this, a space pointer machine object may only be specified as an operand in conjunction with another pointer or a null second operand. The pointer does not exist state is copied from the source to the receiver pointer without signaling the pointer does not exist exception. Source pointer data objects must either be not set or contain a space pointer value when being copied into a receiver space pointer machine object. Receiver pointer data objects will be set with either the system default pointer does not exist value or the space pointer value from a source space pointer machine object.

Normal pointer alignment checking is performed on a pointer data object specified as an operand in conjunction with a space pointer machine object.

Operations not involving space pointer machine objects, those involving just data objects as operands, perform a byte string copy of the data for the specified operands.

The value of the byte string specified by operand 2 is copied to the byte string specified by operand 1 (no padding done).

The byte string identified by operand 2 can contain the storage forms of both scalars and pointers. Normal pointer alignment checking is not done. The only alignment requirement is that the space addressability alignment of the two operands must be to the same position relative to a 16-byte multiple boundary. A boundary alignment exception is signaled if the alignment is incorrect. The pointer attributes of any complete pointers in the source are preserved if they can be completely copied into the receiver. Partial pointer storage forms are copied into the receiver as scalar data. Scalars in the source are copied to the receiver as scalars. The length of the operation is equal to the length of the shorter of the two operands. The copying begins with the two operands left-adjusted and proceeds until completion of the shorter operand.

Operand 1 can specify a space pointer machine object only when operand 2 is null.

If operand 2 is null, operand 1 must define a pointer reference; otherwise, an exception is signaled. When operand 2 is null, the byte string identified by operand 1 is set to the system default pointer does not exist value.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute		X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

CREATE CONTEXT (CRTCTX)

Op Code (Hex)	Operand 1	Operand 2
0112	Pointer for addressability to created context	Context template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: The instruction creates a context with the attributes of the context template specified by operand 2 and returns addressability to the created context in a system pointer stored in the storage area specified by operand 1.

The format of the context template is:

- Template size specification — Char(8)
  - Number of bytes provided — Bin(4)*
  - Number of bytes available for materialization — Bin(4)*

- Object identification — Char(32)
  - Object type — Char(1)*
  - Object subtype — Char(1)
  - Object name — Char(30)

- Object creation options — Char(4)
  - Existence attributes — Bit 0
    - 0 = Temporary
    - 1 = Permanent
  - Space attribute — Bit 1
    - 0 = Fixed-length
    - 1 = Variable-length
  - Reserved (binary 0) — Bit 2
  - Access group — Bit 3
    - 0 = Do not create as member of access group
    - 1 = Create as member of access group
  - Reserved (binary 0) — Bits 4-31

- Recovery options — Char(4)
  - Automatic damaged context rebuild option — Bit 0
    - 0 = Do not rebuild at IMPL
    - 1 = Rebuild at IMPL
  - Reserved (binary 0) — Bits 1-31

- Size of space — Bin(4)

- Initial value of space — Char(1)

- Performance class — Char(4)
  - Space alignment — Bit 0
    - 0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    - 1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.
  - Reserved (binary 0) — Bits 1-4
  - Main storage pool selection — Bit 5
    - 0 = Process default main storage pool is used for object.
    - 1 = Machine default main storage pool is used for object.
  - Reserved (binary 0) — Bit 6
  - Block transfer on implicit access state modification — Bit 7
    - 0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
    - 1 = Transfer the machine default storage transfer size. This value is 8 storage units.
  - Reserved (binary 0) — Bits 8-31

- Reserved (binary 0) — Char(23)

- Access group — System pointer

Note: The values of the template entries annotated by an asterisk are ignored by the instruction.

The template identified by operand 2 must be 16-byte aligned.

If the created context is permanent, it is owned by the user profile governing process execution. The owning user profile is implicitly assigned all private authority states for the context. The storage occupied by the created context is charged to this owning user profile. If the created context is temporary, there is no owning user profile, and all authority states are assigned as public. Storage occupied by the created context is charged to the creating process.

The object identification specifies the symbolic name that identifies the context within the machine. A type code of hex 04 is implicitly supplied by the machine. The object identification is used to identify the object on materialize instructions as well as to locate the object in the machine.

The existence attribute specifies whether the object is to be created as a permanent or a temporary object. A temporary context, if not explicitly destroyed by the user, is implicitly destroyed when machine processing is terminated. Permanent contexts have addressability inserted in the machine context. Temporary contexts' addressability may not be inserted in any context.

A space may be associated with the created object. The space may be fixed or variable in size. The initial allocation is as specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated depends on an algorithm defined by a specific implementation. A fixed size space of zero length causes no space to be allocated. Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this byte value is also used to initialize the new allocation. This entry is ignored if no space is to be allocated.

If the access group creation attribute entry indicates that the context is to be created in an access group, the access group entry must be a system pointer that identifies an access group in which the context is to be created. The existence attribute of the context must be identical to the existence attribute of the access group. If the context is not to be created in an access group, the access group entry is ignored.

The recovery options field indicates the rebuild option. A binary 1 indicates the context is to be rebuilt if damaged. This option is not available for temporary objects. The Materialize Context instruction may be used to materialize the rebuild recovery option for a context.

Note: If the machine context becomes damaged or destroyed, it is implicitly rebuilt and/or recreated at IPL time. If a permanent context becomes damaged, and the context was created with the rebuild recovery option, the context is implicitly rebuilt at IPL time.

The performance class parameter provides information allowing the machine to more effectively manage a context considering overall performance objectives of operations involving the context.

Authorization Required

- Insert
  - User profile of creating process

- Object Control
  - Operand 1 if being replaced

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - User profile of creating process
  - Access group identified by operand 2

- Object Control
  - Operand 1 if being replaced

Events

0002 Authorization

- 0101 Object authorization violation

000C Machine resource

- 0201 Machine auxiliary storage threshold exceeded
- 0501 Machine address threshold exceeded

0010 Process

- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation

- 0101 Instruction reference

0017 Damage set

- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access group			
01 Object ineligible for access group		X	
02 Object exceeds available space		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
02 Machine context damage state			X
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
38 Template Specification			
01 Template value invalid		X	

DESTROY CONTEXT (DESCTX)

Op Code (Hex)	**Operand 1**
0121	Context

Operand 1: System pointer.

Description: The context addressed by the system pointer specified by operand 1 is destroyed. If the context contains addressability to any system object, no exception is signaled. The context is destroyed and the objects are, therefore, not addressed by any context. If the context is a permanent object, the context is deleted from the machine context. The system pointer identified by operand 1 is not modified by the instruction, and a subsequent reference to the context through the pointer results in the object destroyed exception.

Authorization Required

- Object control
  - Operand 1

Lock Enforcement

- Modify
  - Access group
  - User profile of object owner

- Object control
  - Operand 1

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
02 Machine context damage state		X
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

MATERIALIZE CONTEXT (MATCTX)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0133	Receiver	Permanent context, temporary context, or machine context	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer or null.

Operand 3: Character scalar (fixed-length).

Description: Based on the contents of the materialization options specified by operand 3, the symbolic identification and/or system pointers to all or a selected set of the objects addressed by the context specified by operand 2 are materialized into the receiver specified by operand 1. If operand 2 is null, then the machine context is materialized.

The materialization control information requirements field in the materialization options operand specifies the information to be materialized for each selected entry. Symbolic identification and system pointers identifying objects addressed by the context can be materialized based on the bit setting of this parameter. The materialization control selection criteria field specifies the context entries from which information is to be presented. The type code, subtype code, and name fields contain the selection criteria when a selective materialization is specified.

When type code or type/subtype codes are part of the selection criteria, only entries that have the specified codes are considered. When a name is specified as part of the selection criteria, the N characters in the search criteria are compared against the N characters of the context entry, where N is defined by the name length field in the materialization options. The remaining characters (if any) in the context entry are not used in the comparison.

The materialization options operand has the following format:

• Materialization control	Char(2)
– Information requirements (1 = materialize)	Char(1)
Reserved (binary 0)	Bits 0-4
Validation 0 = Validate system pointers 1 = No validation	Bit 5
System pointers	Bit 6
Symbolic identification	Bit 7
– Selection criteria Hex 00–All context entries Hex 01–Type code selection Hex 02–Type code/subtype code selection Hex 04–Name selection Hex 05–Type code/name selection Hex 06–Type code/subtype code/name selection	Char(1)
• Length of name to be used for search argument	Bin(2)
• Type code	Char(1)
• Subtype code	Char(1)
• Name	Char(30)

If the information requirements parameter is binary 0, the context attributes are materialized with no context entries.

If the validation attribute indicates no validation is to be performed, no object validation occurs and a significant performance improvement results. When no validation occurs, some of the following pointers may be erroneous:

- Pointers to destroyed objects

- Pointers to objects that are no longer in the context

- Multiple pointers to the same object

The first 4 bytes of the materialization output identify the total number of bytes available for use by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled. The instruction materializes as many bytes and pointers as can be contained in the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception signaled above.

The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Context identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Context options	Char(4)
– Existence attributes 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Reserved (binary 0)	Bit 2
– Access group 0 = Not a member of access group 1 = Member of access group	Bit 3
– Reserved (binary 0)	Bit 4-31
• Recovery options	Char(4)
– Automatic damaged context rebuild option 0 = Do not rebuild at IMPL 1 = Rebuild at IMPL	Bit 0
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Reserved (binary 0)	Char(16)
• Access group	System pointer
• Context entry (repeated for each selected entry)	Char(16-48)
– Object identification (if requested)	Char(32)
Type code	Char(1)
Subtype code	Char(1)
Name	Char(30)
– Object pointer (if requested)	System pointer

The context entry object identification information, if requested by the materialization options parameter, is present for each entry in the context that satisfies the search criteria. If both system pointers and symbolic identification are requested by the materialization options operand, the system pointer immediately follows the object identification for each entry.

The order of the materialization of a context is by object type code, object subtype code, and object name, all in ascending sequence.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Operand 2

Lock Enforcement

- Materialization
  - Operand 2

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded		X		X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X		X	
08 Invalid operand value range	X		X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MODIFY ADDRESSABILITY (MODADR)

Op Code (Hex)	Operand 1	Operand 2
0192	Receiving context	System object

Operand 1: System pointer or null.

Operand 2: System pointer.

Description: The system object referenced by operand 2 has its addressability inserted into a context, deleted from a context, or transferred from one context to another. If operand 1 addresses a temporary or permanent context, addressability to the object is inserted into the specified context. If the object is currently addressed by another context, this addressability is removed. If the object is currently addressed by the context referenced by operand 1, no operation takes place.

If operand 1 is null, addressability is removed from the context that addresses the system object defined in operand 2. If the object referenced by operand 2 is not currently addressed by a context, no operation takes place.

If operand 2 refers to an object that may only be addressed by the machine context, an object ineligible for context exception is signaled.

Authorization Required

- Insert
  - Operand 1

- Delete
  - Context currently addressing object

- Object management
  - Operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Modify
  - Operand 1
  - Operand 2
  - Context currently addressing object
- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
0E Context Operation			
01 Duplicate object identification		X	
02 Object ineligible for context		X	
10 Damage Encountered			
02 Machine context damage state			X
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X	X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded	X		
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded	X		

RENAME OBJECT (RENAME)

Op Code (Hex)	Operand 1	Operand 2
0162	Object to be renamed	New symbolic identification

Operand 1: System pointer.

Operand 2: Character scalar (fixed-length).

Description: The permanent or temporary system object addressed by the system pointer specified by operand 1 is assigned the symbolic identification (name and/or subtype code) specified by operand 2. All objects that can be addressed by a system pointer can be renamed. System pointers currently addressing the object are not affected by the instruction. The symbolic identification is changed in the context (machine, temporary, or permanent), if any, that addresses the object.

If the new symbolic identification is not unique in the context currently addressing the object, a duplicate object identification exception is signaled, and the object is not renamed.

The format of operand 2 is:

• Rename option (1 = rename)	Char(1)
– Subtype code	Bit 0
– Name	Bit 1
– Reserved (binary 0)	Bits 2-7
• Reserved (binary 0)	Char(1)
• Subtype code	Char(1)
• Name	Char(30)

Note: If either the subtype or the name is not to be changed by the instruction, the corresponding entry on the template is ignored.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
- Object management
  - Operand 1
- Update
  - Context that addresses operand 1

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Context that addresses operand 1
- Object Control
  - Operand 1

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
0A Authorization			
01 Unauthorized for operation	X		
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
02 Machine context damage state			X
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	

RESOLVE DATA POINTER (RSLVDP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0163	Pointer for addressability to data object	Data object identification	Program

Operand 1: Data pointer.

Operand 2: Character(32) scalar (fixed-length) or null.

Operand 3: System pointer or null.

Description: A data pointer with addressability to and the attributes of an external scalar data element is returned in the storage area identified by operand 1.

The following describes the instruction's function when operand 2 is null:

- If operand 1 does not contain a data pointer, an exception is signaled.

- If the data pointer specified by operand 1 is not resolved and has an initial value declaration, the instruction resolves the data pointer to the external scalar that the initial value describes. The initial value defines the external scalar to be located and, optionally, defines the program in which it is to be located. If the program name is specified in the initial value, only that program's activation entry is searched for the external scalar. If no program is specified, programs associated with the activation entries in the process static storage area are searched in reverse order of the activation entries, and operand 3 is ignored.

- If the data pointer is currently resolved and defines an existing scalar, the instruction causes no operation, and no exception is signaled.

The following describes the instruction's function when operand 2 is not null:

- A data pointer that is resolved to the external scalar identified by operand 2 is returned in operand 1. Operand 2 is a 32-byte value that provides the name of the external scalar to be located.

- Operand 3 specifies a system pointer that identifies the program whose activation is to be searched for the external scalar definition. If operand 3 is null, the instruction searches all activations in the process, starting with the most recent activation and continuing to the oldest. The activation under which the instruction is issued also participates in the search. If operand 3 is not null, the instruction searches the activation of the program addressed by the system pointer.

If the external scalar is not located, the object not found exception is signaled. If an unresolved system pointer is encountered when the program searches the activation entries, the pointer not resolved exception is signaled. If the PSSA chain being modified bit is on when this instruction is executed, a stack control invalid exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
04 External data object not found	X			
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation	X		X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state			X	
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
04 Pointer not resolved				X
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
04 Pointer not resolved				X
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand ODT reference		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
03 Stack control invalid				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid		X		
03 Scalar value invalid		X		

RESOLVE SYSTEM POINTER (RSLVSP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0164	Pointer for addressability to object	Object identification and required authorization	Context through which object is to be located	Authority to be set

Operand 1: System pointer.

Operand 2: Character(34) scalar (fixed-length) or null.

Operand 3: System pointer or null.

Operand 4: Character(2) scalar (fixed-length) or null.

Description: This instruction locates an object identified by a symbolic address and stores the object's addressability and authority in a system pointer. A resolved system pointer is returned in operand 1 with addressability to a system object and the requested authority currently available to the process for the object.

Note: The ownership flag is never set in the system pointer.

Operand 2 specifies the symbolic identification of the object to be located. Operand 3 identifies the context to be searched in order to locate the object. Operand 4 identifies the authority states to be set in the pointer. First, the instruction locates an object based on operands 2 and 3. Then, the instruction sets the appropriate authority states in the system pointer.

The following describes the instruction's function when operand 2 is null:

- If operand 1 does not contain a system pointer, an exception is signaled.

- If the system pointer specified by operand 1 is not resolved but has an initial value declaration, the instruction resolves the system pointer to the object that the initial value describes. The initial value defines the following:
  - Object to be located (by type, subtype, and name)
  - Context to be searched to locate the object (optional)
  - Minimum authority required for the object

  If a context is specified, only that context is referenced to locate the object, and operand 3 is ignored. If no context is specified, the context(s) located by the process name resolution list is used to locate the object, and operand 3 is ignored. If the object is of a type that can only be addressed through the machine context, then only the machine context is searched, and the context (if any) identified in the initial value or identified in operand 3 is ignored.

  If the minimum required authority in the initial value is not set (binary 0), the instruction resolves the operand 1 system pointer to the first object encountered with the designated type code, subtype code, and object name without regard to the authorization available to the process for the object. If one or more authorization (or ownership) states are required (signified by binary 1's), the context(s) is searched until an object is encountered with the designated type, subtype, and name and for which the process currently has all required authorization states.

- If the system pointer specified by operand 1 is currently resolved to address an existing object, the instruction does not modify the addressability contained in the pointer and causes only the authority attribute in the pointer to be modified based on operand 4.

If operand 2 is not null, the operand 1 system pointer is resolved to the object identified by operand 2 in the context(s) specified by operand 3. The format of operand 2 is as follows:

• Object specification	Char(32)
– Type code	Char(1)
– Subtype code	Char(1)
– Object name	Char(30)
• Required authorization (1 = required)	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Ownership	Bit 8
– Reserved (binary 0)	Bits 9-15

The allowed type codes are as follows:

Hex 01 = Access group
Hex 02 = Program
Hex 04 = Context
Hex 07 = Journal space
Hex 08 = User profile
Hex 09 = Journal port
Hex 0A = Queue
Hex 0B = Data space
Hex 0C = Data space index
Hex 0D = Cursor
Hex 0E = Index
Hex 0F = Commit block
Hex 10 = Logical unit description
Hex 11 = Network description
Hex 12 = Controller description
Hex 13 = Dump space
Hex 19 = Space
Hex 1A = Process control space

All other codes are reserved. If other codes are specified, they cause a scalar value invalid exception to be signaled.

Operand 3 identifies the context in which to locate the object identified by operand 2. If operand 3 is null, then the contexts identified in the process name resolution list are searched in the order in which they appear in the list. If operand 3 is not null, the system pointer specified must address a context, and only this context is used to locate the object. If the object is of a type that can only be addressed through the machine context, then only the machine context is searched, and operand 3 and the process name resolution list are ignored.

If the required authorization field in operand 2 is not set (binary 0's), the instruction resolves the operand 1 system pointer to the first object encountered with the designated type code, subtype code, and object name without regard to the authorization currently available to the process. If one or more authorization (or ownership) states are required (signified by binary 1's), the context is searched until an object is encountered with the designated type, subtype, name, and the user profiles governing the instruction's execution that have all the required authorization states.

Once addressability has been set in the pointer, operand 4 is used to determine which, if any, of the object authority states is to be set into the pointer.

If operand 4 is null, the object authority states required to locate the object are set in the pointer. This required object authority is as specified in operand 2 or in the initial value for operand 1 if operand 2 is null. If the process does not currently have authorized pointer authority for the object, no authority is stored in the system pointer, and no exception is signaled.

If operands 2 and 4 are null and operand 1 is a resolved system pointer, the authority states in the pointer are not modified.

If operand 4 is not null, it specifies the object authority states to be set in the resolved system pointer. The format of operand 4 is as follows:

• Requested authorization (1 = set authority)	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 2
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Reserved (binary 0)	Bits 8-15

The authority states set in the resolved system pointer are based on the following:

- The authority already stored in the pointer can be increased only when the process has authorized pointer authority to the referenced object. If this authority is not available and the pointer was resolved by this instruction, the authority in the operand 1 system pointer is set to the not set state, and no exception is signaled. If operand 2 is null, if operand 1 is a resolved system pointer containing authority, and if authorized pointer authority is not available to the process, additional authorities cannot be stored in the pointer.

- If the process does not currently have all the authority states requested in operand 4, only the requested and available states are set in the pointer, and no exception is signaled.

- The object authority currently available to the process is cumulative based on the following:
  - Authority stored in a resolved system pointer. This authority applies to this instruction when operand 2 is null and operand 1 is a resolved system pointer with authority stored in it.
  - Public authority for the object.
  - Private authority specifically granted to the process user profile or the most current adopted user profile.
  - All object special authority available to the process user profile or the most current adopted user profile.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution (including operand 3)

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution (including operand 3)

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0A Authorization					
01 Unauthorized for operation	X		X		
10 Damage Encountered					
02 Machine context damage state					X
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state	X		X		
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
04 Pointer not resolved					X
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
0A Invalid operand length		X		X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
02 Scalar attributes invalid		X		X	
03 Scalar value invalid		X		X	

Chapter 4. Space Object Addressing Instructions

This chapter describes the instructions used for space object addressing. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*

ADD SPACE POINTER (ADDSPP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0083	Receiver Pointer	Source pointer	Increment

Operand 1: Space pointer.

Operand 2: Space pointer.

Operand 3: Binary scalar.

Description: This instruction adds a signed value to the offset of a space pointer. The value of the binary scalar represented by operand 3 is algebraically added to the space address contained in the space pointer specified by operand 2, and the result is stored in the space pointer identified by operand 1. Operand 3 can have a positive or negative value. The space object that the pointer is addressing is not changed by the instruction.

Operand 2 must contain a space pointer when the execution of the instruction is initiated; otherwise, an invalid pointer type exception is signaled. When the addressability in a space pointer is modified, the instruction signals a space addressing exception only when the space address to be stored in the pointer has a negative offset value or when the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated. If the exception is signaled by this instruction for this reason, the pointer is not modified by the instruction. Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent of the space cause the space addressing exception to be signaled.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 and operand 2 are space pointer machine objects. This occurs when operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition that existed for operand 2. The appropriate exception condition will be signaled for either pointer when a subsequent attempt is made to reference the space data that the pointer addresses.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3 [4-6]**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X

COMPARE POINTER FOR SPACE ADDRESSABILITY (CMPPSPADB or CMPPSPADI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4-6]
1CE6	Branch options	Compare operand 1	Compare operand 2	Branch target
18E6	Indicator options			Indicator target

Operand 1: Space pointer or data pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, space pointer, or data pointer.

Operand 3 [*4-6*]:

- *Branch Form*—Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*—Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction.

The extender field is required along with from one to four branch targets (for branch option) or one to four indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4-6. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The space addressability contained in the pointer specified by operand 1 is compared with the space addressability defined by operand 2.

The value of the operand 1 pointer is compared based on the following:

- If operand 2 is a scalar data object (element or array), the space addressability of that data object is compared with the space addressability contained in the operand 1 pointer.
- If operand 2 is a pointer, it must be a space pointer or data pointer, and the space addressability contained in the pointer is compared with the space addressability contained in the operand 1 pointer.

Based on the results of the comparison, the resulting condition is used with the extender to transfer control (branch form) or to assign a value to each of the indicator operands (indicator form). If the operands are not in the same space, the resultant condition is unequal. If the operands are in the same space and the offset into the space of operand 1 is larger or smaller than the offset of operand 2, the resultant condition is high or low, respectively. An equal condition occurs only if the operands are in the same space at the same offset. Therefore, the resultant conditions (high, low, equal, and unequal) are mutually exclusive. Consequently, if you specify that an action be taken upon the nonexistence of a condition, this results in the action being taken upon the occurrence of any of the other three possible conditions. For example, a branch not high would result in the branch being taken on a low, equal, or unequal condition.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 or operand 2 is a space pointer machine object or when operand 2 is a scalar based on a space pointer machine object. This occurs when the space pointer machine object contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, the resulting condition of the comparison operation is not defined other than that it will be one of the four valid resultant conditions for this instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: High, low, equal, unequal.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3 [4-6]**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X		
03 Range	X	X		
04 External data object not found	X	X		
06 Optimized addressability invalid	X	X		
08 Argument/Parameter				
01 Parameter reference violation	X	X		
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
09 Invalid branch target operand			X	
0C Invalid operand ODT reference	X	X	X	X
0D Reserved bits are not zero	X	X	X	X

COMPARE SPACE ADDRESSABILITY (CMPSPADB or CMPSPADI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4-6]
1CF2	Branch options	Compare operand 1	Compare operand 2	Branch target
18F2	Indicator options			Indicator target

Operand 1: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, pointer, or pointer array.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, or pointer data object array.

Operand 3 [*4-6*]:

- *Branch Form*—Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*—Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction.

The extender field is required along with from one to four branch targets (for branch option) or one to four indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4-6. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The space addressability of the object specified by operand 1 is compared with the space addressability of the object specified by operand 2.

Based on the results of the comparison, the resulting condition is used with the extender to transfer control (branch form) or to assign a value to each of the indicator operands (indicator form). If the operands are not in the same space, the resultant condition is unequal. If the operands are in the same space and the offset of operand 1 is larger or smaller than the offset of operand 2, the resultant condition is high or low, respectively. Equal occurs only if the operands are in the same space at the same offset. Therefore, the resultant conditions (high, low, equal, and unequal) are mutually exclusive. Consequently, if you specify that an action be taken upon the nonexistence of a condition, this results in the action being taken upon the occurrence of any of the other three possible conditions. For example, a branch not high would result in the branch being taken on a low, equal, or unequal condition.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 or operand 2 is based on a space pointer machine object. This occurs when the space pointer machine object contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, the resulting condition of the comparison operation is not defined other than that it will be one of the four valid resultant conditions for this instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: High, low, equal, unequal.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**[4-6]**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X		
02 Boundary alignment	X	X	X		
03 Range	X	X	X		
06 Optimized addressability invalid	X	X	X		
08 Argument/Parameter					
01 Parameter reference violation	X	X	X		
10 Damage Encountered					
04 System object damage state	X	X	X		X
44 Partial system object damage	X	X	X		X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X		
02 Object destroyed	X	X	X		
03 Object suspended	X	X	X		
24 Pointer Specification					
01 Pointer does not exist	X	X	X		
02 Pointer type invalid	X	X	X		
2A Program Creation					
05 Invalid op code extender field					X
06 Invalid operand type	X	X	X		
07 Invalid operand attribute	X	X	X		
08 Invalid operand value range	X	X	X		
09 Invalid branch target operand			X		
0C Invalid operand ODT reference	X	X	X		X
0D Reserved bits are not zero	X	X	X		X

SET DATA POINTER (SETDP)

Op Code (Hex)	Operand 1	Operand 2
0096	Receiver	Source

Operand 1: Data pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, or character variable array.

Description: A data pointer is created and returned in the storage area specified by operand 1 and has the attributes and space addressability of the object specified by operand 2. Addressability is set to the low-order (leftmost) byte of the object specified by operand 2. The attributes given to the data pointer include scalar type and scalar length.

If operand 2 is a substring compound operand, the length attribute is set equal to the length of the substring. If operand 2 is a subscript compound operand, the attributes and addressability of the single array element specified are assigned to the data pointer. If operand 2 is an array, the attributes and addressability of the first element of the array are assigned to the data pointer. A data pointer can only be set to describe an element of a data array, not a data array in its entirety.

When the addressability in the data pointer is modified, the instruction signals the space addressing exception when one of the following conditions occurs:

- When the space address to be stored in the pointer would have a negative offset value.

- When the offset would address an area beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated.

If the exception is signaled by this instruction for one of these reasons, the pointer is not modified by the instruction.

Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent cause the space addressing exception to be signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found		X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
08 Invalid operand value range		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

SET DATA POINTER ADDRESSABILITY (SETDPADR)

Op Code (Hex)	Operand 1	Operand 2
0046	Receiver	Source

Operand 1: Data pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, or character variable array.

Description: The space addressability of the object specified for operand 2 is assigned to the data pointer specified by operand 1. If operand 1 contains a resolved data pointer at the initiation of the instruction's execution, the data pointer's scalar attribute component is not changed by the instruction. If operand 1 contains an initialized but unresolved data pointer at the initiation of the instruction's execution, the data pointer is resolved in order to establish the scalar attribute component of the pointer. If operand 1 contains other than a resolved data pointer at the initiation of the instruction's execution, the instruction creates and returns a data pointer in operand 1 with the addressability of the object specified for operand 2, and the instruction establishes the attributes as a character(1) scalar.

When the addressability is set into a data pointer, the space addressing exception is signaled by the instruction only when the space address to be stored in the pointer has a negative offset value or if the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated. If the exception is signaled for this reason, the pointer is not modified by the instruction. Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent of the space cause the space addressing exception to be signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process control limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
04 External data object not found	X		
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

SET DATA POINTER ATTRIBUTES (SETDPAT)

Op Code (Hex)	Operand 1	Operand 2
004A	Receiver	Attributes

Operand 1: Data pointer.

Operand 2: Character(7) scalar (fixed-length).

Description: The value of the character scalar specified by operand 2 is interpreted as an encoded representation of an attribute set that is assigned to the attribute portion of the data pointer specified by operand 1. The addressability portion of the data pointer is not modified. If operand 1 contains an initialized but unresolved data pointer at the initiation of the instruction's execution, the data pointer is resolved in order to establish the addressability in the pointer. The attributes specified by the instruction are then assigned to the data pointer. If operand 1 does not contain a data pointer at the initiation of the instruction's execution, an exception is signaled.

The format of the attribute set is as follows:

• Data pointer attributes	Char(7)
– Scalar type	Char(1)
Hex 00 = Binary	
Hex 01 = Floating-point	
Hex 02 = Zoned decimal	
Hex 03 = Packed decimal	
Hex 04 = Character	
– Scalar length	Bin(2)
If binary or character:	
Length (only 2 or 4 for binary)	
If floating-point:	
Length (only 4 or 8 for floating-point)	
If zoned decimal or packed decimal:	
Fractional digits (F)	Bits 0-7
Total digits (T)	Bits 8-15
(where 1 ≤ T ≤ 31, 0 ≤ F ≤ T)	
If character:	
Length (L, where 1 ≤ L ≤ 32767)	
– Reserved (binary 0)	Bin(4)

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

0000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
04 External data object not found	X		
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	

SET SPACE POINTER (SETSPP)

Op Code (Hex)	Operand 1	Operand 2
0082	Receiver	Source

Operand 1: Space pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, or pointer data object.

Description: A space pointer is returned in operand 1 and is set to address the lowest order (leftmost) byte of the byte string identified by operand 2.

When the addressability is set in a space pointer, the instruction signals the space addressing exception when the offset addresses beyond the largest space allocatable in the object or when the space address to be stored in the pointer has a nonpositive offset value. This offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated. If the exception is signaled for this reason, the pointer is not modified by the instruction. Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent of the space cause the space addressing exception to be signaled.

The object destroyed exception, the optimized addressability invalid exception, the parameter reference violation exception, and the pointer does not exist exception are not signaled when operand 1 is a space pointer machine object and operand 2 is based on a space pointer machine object. This occurs when the basing space pointer machine object for operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition which existed for operand 2. The appropriate exception condition is signaled for either pointer upon a subsequent attempt to reference the space data the pointer addresses.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found		X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	

SET SPACE POINTER WITH DISPLACEMENT (SETSPPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0093	Receiver	Source	Displacement

Operand 1: Space pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, or pointer data object.

Operand 3: Binary scalar.

Description: A space pointer is returned in operand 1 and is set to the space addressability of the lowest (leftmost) byte of the object specified for operand 2 as modified algebraically by an integer displacement specified by operand 3. Operand 3 can have a positive or negative value.

When the addressability is set in a space pointer, the instruction signals the space addressing exception when the space address to be stored in the pointer has a negative offset value or when the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated. If the exception is signaled for this reason, the pointer is not modified by the instruction. Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent of the space cause the space addressing exception to be signaled.

The object destroyed exception, the optimized addressability invalid exception, the parameter reference violation exception, and the pointer does not exist exception are not signaled when operand 1 is a space pointer machine object and operand 2 is based on a space pointer machine object. This occurs when the basing space pointer machine object for operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition which existed for operand 2. The appropriate exception condition is signaled for either pointer upon a subsequent attempt is made to reference the space data the pointer addresses.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X

SET SPACE POINTER FROM POINTER (SETSPPFP)

Op Code (Hex)	Operand 1	Operand 2
0022	Receiver	Source pointer

Operand 1: Space pointer.

Operand 2: Data pointer, system pointer, or space pointer.

Description: A space pointer is returned in operand 1 with the addressability to a space object from the pointer specified by operand 2.

The meaning of the pointers allowed for operand 2 is as follows:

Pointer	Meaning
Data pointer or space pointer	The space pointer returned in operand 1 is set to address the leftmost byte of the byte string addressed by the source pointer for operand 2.
System pointer	The space pointer returned in operand 1 is set to address the first byte of the space contained in the system object addressed by the system pointer for operand 2. The space object addressed is either the created system space or an associated space for the system object addressed by the system pointer. If the operand 2 system pointer addresses a system object with no associated space, the invalid space reference exception is signaled.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 and operand 2 are space pointer machine objects. This occurs when operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition that existed for operand 2. The appropriate exception condition will be signaled for either pointer when a subsequent attempt is made to reference the space data that the pointer addresses.

Authorization Required

- Space authority
  - Operand 2 (if a system pointer)

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
04 External data object not found		X	
05 Invalid space reference		X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation			X
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

SET SPACE POINTER OFFSET (SETSPPO)

Op Code (Hex)	Operand 1	Operand 2
0092	Receiver	Source

Operand 1: Space pointer.

Operand 2: Binary scalar.

Description: The value of the binary scalar specified by operand 2 is assigned to the offset portion of the space pointer identified by operand 1. The space pointer continues to address the same space object.

Operand 1 must contain a space pointer at the initiation of the instruction's execution; otherwise, an invalid pointer type exception is signaled.

When the addressability in the space pointer is modified, the instruction signals a space addressing exception when one of the following conditions occurs:

- When the space address to be stored in the pointer has a negative offset value.

- When the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated.

If the exception is signaled by this instruction for this reason, the pointer is not modified by the instruction.

Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent cause the space addressing exception to be signaled.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	

SET SYSTEM POINTER FROM POINTER (SETSPFP)

Op Code (Hex)	Operand 1	Operand 2
0032	Receiver	Source pointer

Operand 1: System pointer.

Operand 2: System pointer, space pointer, data pointer, or instruction pointer.

Description: This instruction returns a system pointer to the system object address by the supplied pointer.

If operand 2 is a system pointer, then a system pointer addressing the same object is returned in operand 1 containing the same authority as the input pointer.

If operand 2 is a space pointer or a data pointer, then a system pointer addressing the system object that contains the associated space addressed by operand 2 is returned in operand 1.

If operand 2 is an instruction pointer, then a system pointer addressing the program system object that contains the instruction addressed by operand 2 is returned in operand 1.

If operand 2 is an unresolved system pointer or data pointer, the pointer is resolved first.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
04 External data object not found	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation			X
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
02 Machine context damage			X
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock state			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found		X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	

STORE SPACE POINTER OFFSET (STSPPO)

Op Code (Hex)	Operand 1	Operand 2
00A2	Receiver	Source

Operand 1: Binary variable scalar.

Operand 2: Space pointer.

Description: The offset value of the space pointer referenced by operand 2 is stored in the binary variable scalar defined by operand 1.

If operand 2 does not contain a space pointer at the initiation of the instruction's execution, an invalid pointer type exception is signaled. If the offset value is greater than 32 767 and operand 1 is a binary (2) scalar, a size exception is signaled.

Events

0000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computations			
0A Size	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X		
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

SUBTRACT SPACE POINTER OFFSET (SUBSPP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0087	Receiver pointer	Source pointer	Decrement

Operand 1: Space pointer.

Operand 2: Space pointer.

Operand 3: Binary scalar.

Description: The value of the binary scalar specified by operand 3 is algebraically subtracted from the space address contained in the space pointer specified by operand 2; the result is stored in the space pointer identified by operand 1. Operand 3 can have a positive or negative value. The space object that the pointer is addressing is not changed by the instruction. If operand 2 does not contain a space pointer at the initiation of the instruction's execution, an invalid pointer type exception is signaled.

When the addressability in the space pointer is modified, the instruction signals a space addressing exception when one of the following conditions occurs:

- When the space address to be stored in the pointer has a negative offset value.

- When the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated.

If the exception is signaled by this instruction for this reason, the pointer is not modified by the instruction.

Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent cause the space addressing exception to be signaled.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 and operand 2 are space pointer machine objects. This occurs when operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition that existed for operand 2. The appropriate exception condition will be signaled for either pointer when a subsequent attempt is made to reference the space data that the pointer addresses.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X		X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
03 Scalar value invalid		X	X	

Chapter 5. Space Management Instructions

This chapter describes the instructions used for space management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CREATE SPACE (CRTS)

Op Code (Hex)	Operand 1	Operand 2
0072	Pointer for space addressability	Space creation template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: A space object is created with the attributes that are specified in the space creation template specified by operand 2, and addressability to the created space is placed in a system pointer that is returned in the addressing object specified by operand 1.

Space objects, unlike other types of system objects, are used to contain a space and serve no other purposes.

The template identified by operand 2 must be 16-byte aligned in the space. The following is the format of the space creation template:

• Template size specification	Char(8)*
– Size of template	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attribute 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context 0 = Addressability is not inserted into context 1 = Addressability is inserted into context	Bit 2
– Access group 0 = Do not create as member of access group 1 = Create as member of access group	Bit 3
– Reserved (binary 0)	Bits 4-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Transient storage pool selection	Bit 6
0 = Default main storage pool (process default or machine default as specified for main storage pool selection) is used for object.	
1 = Transient storage pool is used for object.	
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Unit number	Bits 8-15
– Reserved (binary 0)	Bits 16-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Access group	System pointer

Note: The instruction ignores the values associated with template entries annotated with an asterisk (*).

If the created object is permanent, it is owned by the user profile governing process execution. The owning user profile is implicitly assigned all private authority states for the object. The storage occupied by the created object is charged to this owning user profile. If the created object is temporary, there is no owning user profile, and all authority states are assigned as public. Storage occupied by the created context is charged to the creating process.

The object identification specifies the symbolic name that identifies the space within the machine. A type code of hex 19 is implicitly supplied by the machine. The object identification is used to identify the object on materialize instructions as well as to locate the object in a context that addresses the object.

The existence attributes specify whether the space is to be created as temporary or permanent. A temporary space, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated. A permanent space exists in the machine until it is explicitly destroyed by the user.

The space may have a fixed size or a variable size. The initial allocation is as specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated depends on an algorithm defined by a specific implementation. A fixed size space of zero length causes no space to be allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this byte value is also used to initialize the new allocation. If no space is allocated, this value is ignored.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created space is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

If the access group creation attributes entry indicates that the space is to be created in an access group, the access group entry must be a system pointer that identifies the access group in which the space is to be created. Since access groups may be created only as temporary objects, the existence attribute entry must be temporary (bit 0 equals 0) when the access group object is created. If the space is not to be created into an access group, the access group entry is ignored.

The performance class parameter provides information allowing the machine to more effectively manage the space object considering the overall performance objectives of operations involving the space. The unit number field indicates the auxiliary storage unit on which the space should be located, if possible.

Authorization Required

- Insert
  - User profile of creating process
  - Context identified in operand 2

- Retrieve
  - Context referenced for address resolution

- Object control
  - Operand 1 if being replaced

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Context identified in operand 2
  - User profile of creating process
  - Access group identified in operand 2

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
02 Object exceeds available space		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute		X	
08 Invalid operand value range		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
38 Template Specification			
01 Template value invalid		X	

DESTROY SPACE (DESS)

Op Code (Hex)	**Operand 1**
0025	Space to be destroyed

Operand 1: System pointer.

Description: The designated space is destroyed, and addressability to the space is deleted from a context if it is currently addressing the object. The pointer identified by operand 1 is not modified by the instruction, and a subsequent reference to the pointer causes an object destroyed exception.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Object control
  - Operand 1

Lock Enforcement

- Modify
  - User profile owning object
  - Context addressing object
  - Access group containing object

- Object control
  - Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	**Operand 1**	**Other**
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object		
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

MATERIALIZE SPACE ATTRIBUTES (MATS)

Op Code (Hex)	Operand 1	Operand 2
0036	Receiver	Space object

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The current attributes of the space object specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes that are materialized identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes a materialization length exception.

The second 4 bytes that are materialized identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled in the event that the receiver contains insufficient area for the materialization.

The template identified by operand 1 must be 16-byte aligned in the space. The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization (always 96 for this instruction)	Bin(4)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Context 0 = Addressability not in context 1 = Addressability in context	Bit 2
– Access group 0 = Not member of access group 1 = Member of access group	Bit 3
– Reserved (binary 0)	Bits 4-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Transient storage pool selection	Bit 6
0 = Default main storage pool (process default or machine default as specified for main storage pool selection) is used for object.	
1 = Transient storage pool is used for object.	
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Unit number	Bits 8-15
– Reserved (binary 0)	Bits 16-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Access group	System pointer

Authorization Required

- Operational or space authority
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
03 Materialization length exception	X		

MODIFY SPACE ATTRIBUTES (MODS)

Op Code (Hex)	Operand 1	Operand 2
0062	System object	Space modification template

Operand 1: System pointer.

Operand 2: Binary scalar or character(28) scalar.

Description: The attributes of the space associated with the system object specified for operand 1 are modified with the attribute values specified in operand 2. Operand 1 may address any system object.

The operand 2 space modification template is specified with one of two formats. The abbreviated format, operand 2 specified as a binary scalar, only provides for modifying the size of space attribute. The full format, operand 2 specified as a character scalar, provides for modifying the full set of space attributes.

When operand 2 is a binary value, it specifies the size in bytes to which the space size is to be modified. The current allocation of the space is extended or truncated accordingly to match as closely as possible the specified size. The modified space size will be of at least the size specified. The actual size allocated is dependent upon the algorithm used within the specific implementation of the machine. If the space is of fixed size, or if the value of operand 2 is negative, or if the operand 2 size is larger than that for the largest space that can be associated with the object, the space extension/truncation exception is signaled.

When operand 2 is a character scalar, it specifies a selection of space attribute values to be used to modify the attributes of the space. It must be at least 28 bytes long and have the following format:

• Modification selection	Char(4)
– Modify space length attribute 0 = No 1 = Yes	Bit 0
– Modify size of space 0 = No 1 = Yes	Bit 1
– Modify initial value of space 0 = No 1 = Yes	Bit 2
– Modify performance class 0 = No 1 = Yes	Bit 2
– Reserved (binary 0)	Bit 4-31
• Indicator attributes	Char(4)
– Reserved (binary 0)	Bit 0
– Space length 0 = Fixed length 1 = Variable length	Bit 1
– Reserved binary 0)	Bit 2-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
• Reserved (binary 0)	Char(7)

The modification selection indicator fields select the modifications to be performed on the space.

The modify space length attribute modification selection field controls whether or not the space length attribute is to be modified. When yes is specified, the value of the space length indicator is used to modify the space to be specified fixed or variable length attribute. When no is specified, the space length indicator attribute value is ignored and the space length attribute is not modified.

The modify size of space modification selection field controls whether or not the allocation size of the space is to be modified. When yes is specified, the current allocation of the space is extended or truncated accordingly to match as closely as possible the specified size. The modified size will be at least the size specified. The actual size allocated is dependent upon the algorithm used within the specific implementation of the machine. When no is specified, the current allocation of the space is not modified and the size of space field is ignored.

Modification of the size of space attribute for a space of fixed length can only be performed in conjunction with modification of the space length attribute. In this case, the space length attribute may be modified to the same fixed length attribute or to the variable length attribute. An attempt to modify the size of space attribute for a space of fixed length without modification of the space length attribute results in the signaling of the space extension/truncation exception. Modification of the size of space attribute for a space of variable length can always be performed separately from a modification of the space length attribute.

When the size of space attribute is to be modified, if the value of the size of space field is negative or specifies a size larger than that for the largest space that can be associated with the object, the space extension/truncation exception is signaled.

The modify initial value of space modification selection field controls whether or not the initial value of space attribute is to be modified. When yes is specified, the value of the initial value of space field is used to modify the corresponding attribute of this space. This byte value will be used to initialize any new space allocations for this space due to an extension to the size of space attribute on the current execution of this instruction as well as any subsequent modifications. When no is specified, the initial value of space field is ignored and the initial value of space attribute is not modified.

The modify performance class modification selection field controls whether or not the performance class attribute of the specified system object is to be modified with the values relating to space objects. When yes is specified, the value of the performance class field is used to modify the corresponding attribute of the specified system object. When no is specified, the performance class attribute of the specified system object is not modified.

Modification to or from the state of a space being fixed length of size zero can not be performed for the following objects:

Controller description
Cursor
Data space
Logical unit description
Space
Network description

If such a modification is attempted for these objects, the invalid space modification exception is signaled.

Specifying yes for the modify performance class modification selection field is only allowed when the space to be modified is a fixed length space of size zero. This modification may be specified in conjunction with other modifications. Only bit 0 of the performance class field is used to modify the performance class attribute of the specified system object. A bit value of zero requests that the start of the space storage provide 16-byte multiple (pointer) machine address alignment. A bit value of one requests that the start of the space storage provide 512-byte multiple (buffer) machine address alignment. Bits 1 through 31 are ignored. Specifying yes for the modify performance class modification selection field when the space to be modified is not a fixed length space of size zero results in the signaling of the invalid space modification exception.

A fixed length space of size zero is defined by the machine to have no internal storage allocation. Due to this, a modification to or from this state is, in essence, the same as a destroy or create for the space associated with the specified system object. The effect of modifying to this state is similar to destroying the associated space in that address references to the space through previously set pointers will result in signaling of the object destroyed exception. Additionally, an attempt to set a space pointer to the space associated with the specified system object through the Set Space Pointer from Pointer instruction will result in the signaling of the invalid space reference exception. To the contrary, modifying the space attributes from this state is similar to creating an associated space in that the Set Space Pointer from Pointer instruction can be used to set a space pointer to the start of a storage within the associated space and the allocated space storage can be used to contain space data.

The extension and truncation of a space is always by an implementation-defined multiple of 256 bytes. This means that if, for example, the implementation defined multiple is 2 (or 512 bytes), any modification of the space size will be in increments of 512 bytes.

Authorization Required

- Object management
  – Operand 1

- Retrieve
  – Contexts referenced for address resolution

Lock Enforcement

- Materialize
  – Contexts referenced for address resolution

- Object control
  – Operand 1 (when operand 2 is binary)

- Modify
  – Operand 1 (when operand 2 is character)

Events

0002 Authorization
 0101 Object authorization violation

000C Machine resource
 0201 Machine auxiliary storage threshold exceeded
 0501 Machine address threshold exceeded

000D Machine status
 0101 Machine check

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded	X		
36 Space Management			
01 Space extension/truncation	X	X	

Chapter 6. Independent Index Instructions

This chapter describes the instructions used for indexes. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CREATE INDEPENDENT INDEX (CRTINX)

Op Code (Hex)	**Operand 1**	**Operand 2**
0446	Index	Index description template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction creates an independent index based on the index template specified by operand 2 and returns addressability to the index in a system pointer stored in the addressing object specified by operand 1. The maximum length allowed for the independent index entry is 120 bytes.

The format of the index description template described by operand 2 is as follows (must be aligned on a 16-byte multiple):

• Template size specification	Char(8)
– Number of bytes provided	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context 0 = Do not insert addressability in context 1 = Insert addressability in context	Bit 2
– Access group 0 = Do not create as member of access group 1 = Create as member of access group	Bit 3
– Reserved (binary 0)	Bits 4-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, 0 must be specified for the performance class.	Bit 0
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Access group	System pointer
• Index attributes	Char(1)
– Entry length attribute	Bit 0
0 = Fixed-length entries	
1 = Variable-length entries	
– Immediate update	Bit 1
0 = No immediate update	
1 = Immediate update	
– Key insertion	Bit 2
0 = No insertion by key	
1 = Insertion by key	
– Entry format	Bit 3
0 = Scalar data only	
1 = Both pointers and scalar data	
– Optimized processing mode	Bit 4
0 = Optimize for random references	
1 = Optimize for sequential references	
– Reserved (binary 0)	Bits 5-7
• Argument length	Bin(2)
• Key length	Bin(2)

Note: This instruction ignores the values associated with the entries annotated with an asterisk (*).

The template identified by operand 2 must be 16-byte aligned.

If the created object is permanent, it is owned by the user profile governing process execution. The owning user profile is implicitly assigned all private authority states for the object. The storage occupied by the created object is charged to this owning user profile. If the created object is temporary, there is no owning user profile, and all authority states are assigned as public. Storage occupied by the created context is charged to the creating process.

The object identification specifies the symbolic name that identifies the space within the machine. A type code of hex 0E is implicitly supplied by the machine. The object identification is used to identify the object on materialize instructions as well as to locate the object in a context that addresses the object.

The existence attribute specifies that the index is to be created as a temporary object. A temporary index, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated.

A space may be associated with the created object. The space may be fixed or variable in size. The initial allocation is as specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated is dependent on an algorithm defined by a specific implementation. Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this byte value is also used to initialize the new allocation. If no space is allocated, this value is ignored.

If the initial context creation attribute entry indicates that addressability is to be placed in a context, the context entry must be a system pointer that identifies a context where addressability to the newly created object is to be placed. If the initial context indicates that addressability is not to be placed in a context, the context entry is ignored.

If the access group creation attribute entry indicates that the object is to be created in an access group, the access group entry must be a system pointer that identifies an access group in which the object is to be created. The existence attribute of the object must be identical to the existence attribute of the access group. If the object is not to be created in the access group, the access group entry is ignored.

The performance class parameter provides information allowing the machine to more effectively manage the object considering the overall performance objectives of operations involving the index.

If the entry length attribute field specifies fixed-length (bit 0 = 0), the entry length of every index entry is established at creation by the value in the argument length field of the index description template. If the length attribute field specifies variable-length, then entries will be variable-length (the length of each entry is supplied when the entry is inserted), and the argument length value is ignored.

If the immediate update field specifies that an immediate update should occur (bit 1 = 1), then every update to the index will be written to auxiliary storage after every insert or remove operation.

If the key insertion field specifies insertion by key (bit 2 = 1), then the key length field must be specified. This allows the specification of a portion of the argument (the key), which may be manipulated in either of the following ways in the Insert Index Entry instruction:

- The insert will not take place if the key portion of the argument is already in the index.

- The insert will cause the nonkey portion of the argument to be replaced if the key is already in the index.

The entry format field designates the index entries as containing both pointers and scalar data or only scalar data. The both pointers and scalar data entry can be used only for indexes with fixed-length entries. If the index is created to contain both pointers and data (bit 3 = 1), then:

- Entries to be inserted must be 16-byte aligned.

- Each entry retrieved by the Find Independent Index Entry instruction or the Remove Independent Index Entry is 16-byte aligned.

- Pointers are allowed in both the key and nonkey portions of an index entry.

- Pointers need not be at the same location in every index entry.

- Pointers inserted into the index remain unchanged. No resolution is performed before insertion.

If the index is created to contain only scalar data, then:

- Entries to be inserted need not be aligned.

- Entries returned by the Find Independent Index Entry instruction or the Remove Independent Index Entry instruction are not aligned.

- Any pointers inserted into the index will be invalidated.

The optimized processing mode index attribute field is used to designate whether the index should be created and maintained in a manner that optimizes performance for either random or sequential operations.

The key length must have a value less than or equal to the argument length whether specified during creation (for fixed-length entries) or during insertion (for variable length). The key length is not used if the key insertion field specifies no insertion by key (bit 3 = 0).

Authorization Required

- Insert
  - Context identified by operand 2
  - User profile of creating process

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Modify
  - Access group identified by operand 2
  - User profile of creating process
  - Context identified by operand 2

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded
0501 Machine address threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
01 Object ineligible for access group		X	
02 Object exceeds available space		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context			
01 Duplicate object identification		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute		X	
08 Invalid operand value range		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
38 Template Specification			
01 Template value invalid		X	

DESTROY INDEPENDENT INDEX (DESINX)

Op Code (Hex)	Operand 1
0451	Index

Operand 1: System pointer.

Description: A previously created index identified by operand 1 is destroyed, and addressability to the object is removed from any context in which addressability exists. The system pointer identified by operand 1 is not modified by the instruction, and a subsequent reference to the destroyed index through the pointer results in an object destroyed exception.

Authorization Required

- Object control
  - Operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Object control
  - Operand 1

- Modify
  - Access group which contains operand 1
  - Context which addresses operand 1
  - User profile which owns index

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

FIND INDEPENDENT INDEX ENTRY (FNDINXEN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0494	Receiver	Index	Option list	Search argument

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Space pointer.

Operand 4: Space pointer.

Description: This instruction searches the independent index identified by operand 2 according to the search criteria specified in the option list (operand 3) and the search argument (operand 4); then it returns the desired entry or entries in the receiver field (operand 1). The maximum size of the independent index entry is 120 bytes.

The option list is a variable-length area that identifies the type of search to be performed, the length of the search argument(s), the number of resultant arguments to be returned, the lengths of the entries returned, and the offsets to the entries within the receiver identified by the operand 1 space pointer. The option list has the following format:

• Rule option	Char(2)
• Argument length	Bin(2)
• Argument offset	Bin(2)
• Occurrence count	Bin(2)
• Return count	Bin(2)

Each entry that is returned to the receiver operand contains the following:

• Entry length	Bin(2)
• Offset	Bin(2)

The rule option identifies the type of search to be performed and has the following meaning:

Search Type	Value (Hex)	Meaning
=	0001	Find equal occurrences of operand 4.
>	0002	Find occurrences that are greater than operand 4.
<	0003	Find occurrences that are less than operand 4.
≥	0004	Find occurrences that are greater than or equal to operand 4.
≤	0005	Find occurrences that are less than or equal to operand 4.
First	0006	Find the first index entry or entries.
Last	0007	Find the last index entry or entries.
Between	0008	Find all entries between the two arguments specified by operand 4 (inclusive).

The option to find between limits requires that operand 4 be a 2-element vector in which element 1 is the starting argument and element 2 is the ending argument. All arguments between (and including) the starting and ending arguments are returned, but the occurrence count specified is not exceeded.

If the index was created to contain both pointers and scalar data, then the search argument must be 16-byte aligned. For the option to find between limits, both search arguments must be 16-byte aligned.

The rule option and the argument length determine the search criteria used for the index search. The argument length must be greater than or equal to one. The argument length for fixed-length entries must be less than or equal to the argument length specified when the index is created.

The argument length entry specifies the length of the search argument (operand 4) to be used for the index search. When the rule option equals first or last, the argument length entry is ignored. For the option to find between limits, the argument length option specifies the lengths of one vector element. The lengths of the vector elements must be equal.

The argument offset is the offset of the second search argument from the beginning of the entire argument field (operand 4). The argument offset field is ignored unless the rule option is find between.

The occurrence count specifies the maximum number of index entries that satisfy the search criteria to be returned. This field is limited to a maximum value of 4095. If this value is exceeded, a template value invalid exception is signaled.

The return count specifies the number of index entries satisfying the search criteria that were returned in the receiver (operand 1). If this field is 0, no index arguments satisfied the search criteria.

There are two fields in the option list for each entry returned in the receiver (operand 1). The entry length is the length of the entry retrieved from the index. The offset has the following meaning:

- For the first entry, the offset is the number of bytes from the beginning of the receiver (operand 1) to the first byte of the first entry.

- For any succeeding entry, the offset is the number of bytes from the beginning of the immediately preceding entry to the first byte of the entry returned.

The entries that are retrieved as a result of the Find Independent Index Entry instruction are always returned starting with the entry that is closest to or equal to the search argument and then proceeding away from the search argument. For example, a search that is for < (less than) or ≤ (less than or equal to) returns the entries in order of decreasing value.

All the entries that satisfy the search criteria (up to the occurrence count) are returned in the space starting at the location designated by the operand 1 space pointer.

If the index was created to contain both pointers and scalar data, then each returned entry is 16-byte aligned.

If the index was created to contain only scalar data, then returned entries are contiguous.

Every entry retrieved causes the count of the find operations to be incremented by 1. The current value of this count is available through the Materialize Index Attributes instruction.

Authorization Required

- Retrieve
  - Operand 2
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	4	Other
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0A Authorization					
01 Unauthorized for operation		X			
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state		X			
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
03 Pointer addressing invalid object		X			
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
0A Invalid operand length	X	X	X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
38 Template Specification					
01 Template value invalid			X		
02 Template size invalid			X		

INSERT INDEPENDENT INDEX ENTRY (INSINXEN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04A3	Index	Argument	Option list

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: Space pointer.

Description: This instruction inserts one or more new entries into the independent index identified by operand 1 according to the criteria specified in the option list (operand 3). Each entry is inserted into the index at the appropriate location based on the EBCDIC value of the argument. The maximum length allowed for the independent index entry is 120 bytes.

The argument (operand 2) and the option list (operand 3) have the same format as the argument and option list for the Find Independent Index Entry instruction.

The rule option identifies the type of insert to be performed and has the following meaning:

Insert Type	Value (Hex)	Meaning	Authorization
Insert	0001	Insert unique argument	Insert
Insert with replacement	0002	Insert argument, replacing the nonkey portion if the key is already	
in the index			Update
Insert without replacement	003	Insert argument only if the key is not already in the index	Insert

The insert rule option is valid only for indexes not containing keys. The insert with replacement rule option and the insert without replacement rule option are valid for indexes containing either fixed- or variable-length entries with keys. The duplicate key argument exception is signaled for the following conditions:

- If the rule option is insert and the argument to be inserted (operand 2) is already in the index

- If the rule option is insert without replacement and the key portion of the argument to be inserted (operand 2) is already in the index

The argument length and argument offset fields are ignored.

The occurrence count specifies the number of arguments to be inserted. This field is limited to a maximum value of 4095. If this value is exceeded, a template value invalid exception is signaled.

If the index was created to contain both pointers and data, then each entry to be inserted must be 16-byte aligned. If the index was created to contain variable-length entries, then the entry length and offset fields must be specified in the option list for each argument in the space identified by operand 2. The entry length is the length of the entry to be inserted.

If the index was created to contain both pointer and scalar data, the offset field in the option list must be supplied for each entry to be inserted. The offset is the number of bytes from the beginning of the previous entry to the beginning of the entry to be inserted. For the first entry, this is the offset from the start of the space identified by operand 2.

The return count specifies the number of entries inserted into the index. If the index was created to contain only data, then any pointers inserted are invalidated.

Authorization Required

- Insert or update depending on insert type
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
02 Access Group				
02 Object exceeds available space	X			
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
18 Independent Index				
01 Duplicate key argument in index	X			
1A Lock State				
01 Invalid lock state	X			
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded	X			
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2E Resource Control Limit				
01 User profile storage limit exceeded	X			
38 Template Specification				
01 Template value invalid			X	
02 Template size invalid			X	

MATERIALIZE INDEPENDENT INDEX ATTRIBUTES (MATINXAT)

Op Code (Hex)	Operand 1	Operand 2
0462	Receiver	Index

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The instruction materializes the creation attributes and current operational statistics of the independent index identified by operand 2 into the space identified by operand 1. The format of the attributes materialized is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temporary 1 = Reserved	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Context 0 = Addressability not in context 1 = Addressability in context	Bit 2
– Access group 0 = Not a member of access group 1 = Member of access group	Bit 3
– Reserved (binary 0)	Bits 4-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)

• Initial value of space	Char(1)
• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool used for object.	
1 = Machine default main storage pool used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = The minimum storage transfer size for this object is a value of 1 storage unit.	
1 = The machine default storage transfer size for this object is a value of 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Access group	System pointer
• Index attributes	Char(1)
• Argument length	Bin(2)
• Key length	Bin(2)
• Index statistics	Char(12)
– Entries inserted	Bin(4)
– Entries removed	Bin(4)
– Find operations	Bin(4)

The number of arguments in the index equals the number of entries inserted minus entries removed. The value of the find operations field is initialized to 0 each time the index is materialized. The value may not be correct after an abnormal system termination.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged.

No exceptions other than the materialization length exception described previously are signaled in the event that the receiver contains insufficient area for the materialization.

The template identified by the operand 1 space pointer must be 16-byte aligned. Values in the template remain the same as the values specified at the creation of the independent index except that the object identification, context, and size of the associated space contain current values.

If the entry length is fixed, then the argument length is the value supplied in the template when the index was created. If the entry length is variable, then the argument length entry is equal to the length of the longest entry that has ever been inserted into the index.

Authorization Required

- Operational
  - Operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
01 Template value invalid		X	
03 Materialization length exception	X		

MODIFY INDEPENDENT INDEX (MODINX)

Op Code (Hex)	Operand 1	Operand 2
0452	Independent index	Modification option

Operand 1: System pointer.

Operand 2: Character (4) scalar.

Description: This instruction modifies the selected attributes of the independent index specified by operand 1 to have the values specified in operand 2. The modification options specified in operand 2 have the following format:

• Modification selection	Char(1)
– Reserved (binary 0)	Bit 0
– Immediate update	Bit 1
0 = Do not change immediate update attribute	
1 = Change immediate update attribute	
– Reserved (binary 0)	Bits 2-7
• New attribute value	Char(1)
– Reserved (binary 0)	Bit 0
– Immediate update	Bit 1
0 = No immediate update	
1 = Immediate update	
– Reserved (binary 0)	Bits 2-7
• Reserved (binary 0)	Char(2)

If the modification selection immediate update is 0, then the immediate update attribute is not changed. If the modification selection immediate update bit is 1, the immediate update attribute is changed to the new immediate update attribute value.

If the immediate update attribute of the index was previously set to no immediate update, and it is being modified to immediate update, then the index is ensured before the attribute is modified.

Modification of the immediate update attribute of an independent index will occur only if the index was created in release 2 or later.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Object management
  - Operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Modify
  - Operand 1

- Materialization
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
10 Damage Encountered			
04 System object damage	X		X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer address invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid		X	

REMOVE INDEPENDENT INDEX ENTRY (RMVINXEN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0484	Receiver	Index	Option list	Argument

Operand 1: Space pointer or null.

Operand 2: System pointer.

Operand 3: Space pointer.

Operand 4: Space pointer.

Description: The index entries identified by operands 3 and 4 are removed from the independent index identified by operand 2 and optionally returned in the receiver specified by operand 1. The maximum length of an independent index entry is 120 bytes.

The option list (operand 3) and the argument (operand 4) have the same format and meaning as the option list and argument for the Find Independent Index Entry instruction. The return count designates the number of index entries that were removed from the index.

The arguments removed are returned in the receiver field if a space pointer is specified for operand 1. If operand 1 is null, the entries removed from the index are not returned. If neither space pointer nor null is specified for operand 1, the entries are returned in the same way that entries are returned for the Find Independent Index Entry instruction.

Every entry removed causes the occurrence count to be incremented by 1. The current value of this count is available through the Materialize Index Attributes instruction. The occurrence count field must be less then 4096.

Authorization Required

- Delete
  - Operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Operand 2

Events

0002 Authorization
 0101 Object authorization violation

000C Machine resource
 0201 Machine auxiliary storage threshold exceeded

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
02 Access Group					
02 Object exceeds available space		X			
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0A Authorization					
01 Unauthorized for operation		X			
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state		X			X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
04 Object storage limit exceeded		X			
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
03 Pointer addressing invalid object		X			
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2E Resource Control Limit					
01 User profile storage limit exceeded		X			
38 Template Specification					
01 Template value invalid			X		
02 Template size invalid			X		

Chapter 7. Authorization Management Instructions

This chapter describes the instructions used for authorization management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CREATE USER PROFILE (CRTUP)

Op Code (Hex)	**Operand 1**	**Operand 2**
0116	User profile	User profile creation template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: A user profile is created in accordance with the user profile template specification. A system pointer addressing the created user profile is returned in the addressing object specified by operand 1.

A privileged instruction exception is signaled if the user profile(s) governing the execution of the process is not authorized to create a user profile. An exception is signaled if the new user profile is either for a privileged instruction or for a special authorization state that is not authorized the user profile(s) that governs the execution of the instruction.

The template identified by operand 2 must be 16-byte aligned in the space. Following is the format of the user profile template:

• Template size specification	Char(8)*
– Size of template	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attribute 1 = Permanent (required)	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Reserved (binary 0)	Bits 2-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

- Performance class Char(4)
  - Space alignment Bit 0
    - 0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    - 1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.
  - Reserved (binary 0) Bits 1-4
  - Main storage pool selection Bit 5
    - 0 = Process default main storage pool is used for object.
    - 1 = Machine default main storage pool is used for object.
  - Reserved (binary 0) Bit 6
  - Block transfer on implicit access state modification Bit 7
    - 0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
    - 1 = Transfer the machine default storage transfer size. This value is 8 storage units.
  - Reserved (binary 0) Bits 8-31

- Reserved (binary 0) Char(39)

- Privileged instructions (1 = authorized) Char(4)
  - Create logical unit description Bit 0
  - Create network description Bit 1
  - Create controller description Bit 2
  - Create user profile Bit 3
  - Modify user profile Bit 4
  - Diagnose Bit 5
  - Terminate machine processing Bit 6
  - Initiate process Bit 7
  - Modify resource management control Bit 8
  - Reserved (binary 0) Bits 9-31

- Special authorizations (1 = authorized) Char (4)
  - All object authority Bit 0
  - Load (unrestricted) Bit 1
  - Dump (unrestricted) Bit 2
  - Suspend object (unrestricted) Bit 3
  - Load (restricted) Bit 4
  - Dump (restricted) Bit 5
  - Suspend (restricted) Bit 6
  - Process control Bit 7
  - Reserved (binary 0) Bit 8
  - Service authority Bit 9
  - Reserved (binary 0) Bits 10-23
  - Modify machine attributes Bits 24-31

Group 2	Bit 24
Group 3	Bit 25
Group 4	Bit 26
Group 5	Bit 27
Group 6	Bit 28
Group 7	Bit 29
Group 8	Bit 30
Group 9	Bit 31

  Note: Group 1 requires no authorization.

- Storage authorization–The maximum amount of auxiliary storage (in units of 1024 bytes) that can be allocated for the storage of objects owned by this user profile Bin(4)

- Storage utilization–The current amount of auxiliary storage (in units of 1024 bytes) allocated for the storage of objects owned by this user profile Bin(4)

Note: The values associated with the template parameters identified by an asterisk (*) are ignored by the create user profile instruction.

The created user profile is owned by the user profile governing process execution. All private object authorization states are implicitly assigned to the owning user profile. No user profile is charged for the storage occupied by the newly created user profile.

The object identification specifies the symbolic name that identifies the user profile within the machine. An object type of hex 08 is implicitly supplied by the machine. The object identification is used to identify the object for materialize instructions as well as to locate the object through the machine context. The object identification for a user profile must be unique throughout the machine.

The user profile is created as a permanent object and exists until explicitly destroyed. Addressability to the created user profile is implicitly inserted into the machine context.

A space may be associated with the created user profile. The size of the space may be fixed or variable. The initial allocation is as specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated depends on an algorithm defined by a specific implementation. A fixed space size of zero length causes no space to be allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended, this byte value is also used to initialize the new allocation.

When a permanent object is created, it is owned by the user profile governing process execution. The owning user profile is implicitly assigned all private authority states for the object. The storage occupied by the associated space is charged to the owning user profile.

The performance class parameter provides information that allows the machine to more effectively manage the object by considering the overall performance objectives of operations involving the context.

Authorization Required

- Privileged instruction
- Privileges and special authorizations being granted to the created user profile
- Insert
  - User profile of creating process
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Modify
  - User profile of creating process

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
02 Privileged instruction			X
05 Create/modify user profile beyond level of authorization		X	
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
02 Machine context damage state			X
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
38 Template Specification			
01 Template value invalid		X	

DESTROY USER PROFILE (DESUP)

Op Code (Hex)	**Operand 1**
0125	User profile

Operand 1: System pointer.

Description: The user profile specified by operand 1 is destroyed, and addressability to the profile is deleted from the machine context. The system pointer specified by operand 1 is not modified by the instruction, and any future reference to the destroyed user profile through the pointer causes an object destroyed exception.

If the referenced user profile owns any object (other than itself) when the Destroy User Profile instruction is executed, an object not eligible for destruction exception is signaled and the user profile is not destroyed. The exception is also signaled if the process executing the instruction is controlled by the user profile to be destroyed.

Because a user profile is implicitly locked (LSRD) by the machine when a process is initiated by the user profile, an invalid lock state exception is signaled if any process is currently initiated by the referenced user profile and an attempt is made to destroy the user profile.

Authorization Required

- Object control
  - Operand 1

Lock Enforcement

- Modify
  - User profile of owner of operand 1
- Object control
  - Operand 1

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
02 Machine context damage state		X
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	

GRANT AUTHORITY (GRANT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0173	User profile	System object	Authorization template

Operand 1: System pointer or null.

Operand 2: System pointer.

Operand 3: Character(2) scalar (fixed-length).

Description: This instruction grants authority to a specified object. This authority may include all new authority codes or a new authority code to be added to the authority codes previously granted. Public authority for an object can also be granted. If operand 1 is addressing a user profile, that user profile will be granted the private authorization states specified by operand 3 for the system object specified by operand 2. If the user profile previously had no authority for the specified object, the object and the specified authorization states are added to the user profile's set of authorized objects. If the user profile previously had some authority for the specified object, then the authorization states specified by operand 3 are logically ORed to those authorization states previously held. If no private authorization states that apply to the designated object type are defined in the authorization template then no change is made to the user profile's authorization.

If operand 1 is null, the instruction grants public authorization. If public authorization has been previously granted for the object, then the authorization states specified by operand 3 are logically ORed to those public authorization states previously granted. Operand 3 is a 2-byte character scalar and employs the following bit representations to designate the authorization states: (1 = authorized)

• Authorization template	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Reserved (binary 0)	Bits 8-15

The four authorities (bits 4-7)–retrieve, insert, delete, and update–constitute the operational authorities. Granting any of these four authorities is sufficient for instructions requiring operational authority. For those objects (except space objects) that do not support these operational authorities individually, all four of these authorities must be granted when operational authority is to be granted. The operational authority provided by these bits is considered reserved for objects that do not have any distinction between them.

The user profile governing the execution of the instruction (process user profile or most current adopted user profile) must have object management authority as well as any authority state being granted for the object, or it must indirectly have authority through the all-object authority special authorization or through ownership of the object.

Ownership or all-object authority is required in order to grant object management authority. The owner is always allowed to grant any authority, even if it has been retracted from him. A nonowner must have the authorities he is granting in addition to object management authority. Authorization bits that do not support any function for a particular object type are considered reserved.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Authorities being granted with object management or ownership
  - Operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Operand 2

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands			Other
	1	**2**	**3**	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
03 Attempt to grant/restrict authority state to that which is not authorized			X	
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state	X	X	X	
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded	X			
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X		
02 Object destroyed	X	X	X	
03 Object suspended	X	X		
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	
38 Template Specification				
02 Template size invalid				X

GRANT-LIKE AUTHORITY (GRNTLIKE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0174	Receiving user profile	Reference user profile	Authorization template	Receiver

Operand 1: System pointer.

Operand 2: System pointer.

Operand 3: Character(8) scalar.

Operand 4: Space pointer or null.

Description: This instruction grants authority from a specified reference user profile for all objects (owned and authorized) in the reference user profile. For each object, all new authority codes may be granted or new authority codes may be added to authority codes previously granted. The receiving user profile (operand 1) is granted the private authorization states for all objects (as specified in the authorization template) in the reference user profile (operand 2). If the receiving user profile previously had no authority for an object in the reference user profile, then the object and its authorization states are added to the receiving user profile's set of authorized objects.

If the receiving user profile previously had some authority for an object in the reference user profile, then the authorization states for that object in the reference user profile are logically OR-ed to the object's authorization states in the receiving user profile.

In order to grant authorities for objects in the reference user profile to the receiving user profile, the user profile governing the execution of the instruction (process user profile or adopted user profile) must meet one of the following conditions:

- Object management authority in addition to the authorities being granted for each object in the reference user profile. If the user profile governing the execution of the command does not have object management authority and all the authorities to be granted for an object, then no authority will be granted for that object to the receiver user profile.

- Must indirectly have authority through the all object authority special authority.

- Must be the owner of the object.

In order to grant object management authority, ownership or all object authority is required. The owner is allowed to grant any authorities, even if they have been retracted from him.

When this instruction is executed, ownership of an object is not changed nor are the authorities held by the reference user profile changed.

The authorization template is input to the instruction and has the following format:

• Authorization template	Char(8)
– Flags	Char(2)
Grant authority for all authorized objects in reference user profile	Bit 0
Grant authority for all owned objects in reference user profile	Bit 1
Explicit authority to be granted is provided in authorization template for authorized objects	Bit 2
Explicit authority to be granted is provided in authorization template for owned objects	Bit 3
Reserved (binary 0)	Bits 4-15
– Explicit authority for authorized objects	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Reserved (binary 0)	Bits 8-15
– Explicit authority for owned objects	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Reserved (binary 0)	Bits 8-15
– Reserved (binary 0)	Char(2)

The explicit authorities (authorized and owned) specified in the authorization template may be used to limit the authorities granted to the receiving profile. They are logically ANDed with the authorities in the reference user profile. If the user profile governing the execution of the instruction is authorized, the result is logically ORed with any existing authorities in the receiving user profile.

The scalar value invalid exception is signaled when the flags for grant authority for all authorized objects in reference user profile or grant authority for all owned objects in reference user profile are not specified, but the corresponding explicit authorization mask is specified.

The first 4 bytes of the information returned in the receiver identify the total quantity of bytes in the receiver area. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 16 causes the template value invalid exception to be signaled.

The second 4 bytes are the total number of objects for which authorization was granted.

The third 4 bytes are filled in by the instruction and are the number of objects for which detail information (objects for which authorization was not granted) is supplied in the receiver area.

The fourth 4 bytes are filled in by the instruction and are the total number of objects for which authorization was not granted. This number will be greater than the previous field when the receiver area does not contain sufficient space for detail information for all objects for which authorization was not granted.

The instruction provides detail information for as many objects as will fit in the receiver area supplied to the instruction. If the context not found flag is returned, then the context name field does not contain valid data. Any excess bytes in the receiver area are unchanged.

If operand 4 is null, then no information is returned from the instruction.

The information returned in the receiver has the following format:

• Number of bytes provided for containing feedback information	Bin(4)
• Total number of objects for which authorization was granted	Bin(4)
• Number of objects for which authorization was not granted and detail information is provided (number of 66-byte entries below)	Bin(4)
• Total number of objects for which authorization was not granted	Bin(4)

Detail information for each object for which authorization was not granted:

• Object information	Char(66)
– Type code	Char(1)
– Subtype code	Char(1)
– Object name	Char(30)
– Context name	Char(30)
– Flags	Char(2)
– – Insufficient authority to grant authorization	Bit 0
– – Lock conflict prevented granting authorization	Bit 1
– – Context not found	Bit 2
– – Reserved (binary 0)	Bits 3-15
– Authorization not granted	Char(2)
– – Object control	Bit 0
– – Object management	Bit 1
– – Authorized pointer	Bit 2
– – Space authority	Bit 3
– – Retrieve	Bit 4
– – Insert	Bit 5
– – Delete	Bit 6
– – Update	Bit 7
– – Reserved (binary 0)	Bits 8-15

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Authorities being granted and object management, or ownership
  - To each object identified in operand 2–reference user profile. (See description below for specific details.)

- Operational
  - Operand 2

- Retrieve
  - Operand 2, if granting owned objects
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2, if granting owned objects
  - Contexts referenced for address resolution

- Modification
  - For each object identified in operand 2–reference user profile.

Events

0002 Authorization
0101 Authorization violation

000C Machine resources
0201 Machine auxiliary storage exceeded

000D Machine status
0101 Machine check

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0A Authorization					
01 Unauthorized for operation					X
03 Attempt to grant/retract authority state to an object that is not authorized		X			X
10 Damage Encountered					
01 Machine context					X
04 System object damage	X	X			X
44 Partial system object damage					X
1A Lock State					
01 Invalid lock state	X	X			
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
04 Object storage limit exceeded	X				
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
03 Pointer addressing invalid object	X	X			
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
0A Invalid operand length			X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
01 Scalar type invalid	X	X	X	X	
02 Scalar attributes invalid			X		
03 Scalar value invalid			X		
38 Template Specification					
02 Template size invalid				X	

MATERIALIZE AUTHORITY (MATAU)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0153	Receiver	System object	User profile

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: System pointer or null.

Description: This instruction materializes the specific types of authority for a system object available to the specified user profile. The private authorization that the user profile specified by operand 3 is assigned to the permanent system object specified by operand 2, and the object's public authorization is materialized in operand 1. If operand 3 is null, then only the object's public authorization is materialized, and the private authorization field in the materialization is set to binary 0.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized (12 for this instruction). The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization (contains a value of 12 for this instruction)	Bin(4)
• Private authorization (1 = authorized)	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Ownership (1 = yes)	Bit 8
– Reserved (binary 0)	Bits 9-15
• Public authorization (1 = authorized)	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Reserved (binary 0)	Bits 8-15

Any of the four authorizations–retrieve, insert, delete, or update–constitute operational authority.

If this instruction references a temporary object, all public authority states are materialized. Private authority states are not materialized.

Authorization Required

- Operational
  - Operand 3

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Operand 3
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X	X	
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state		X	X	
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object		X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X			
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
38 Template Specification				
03 Materialization length exception	X			

MATERIALIZE AUTHORIZED OBJECTS (MATAUOBJ)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
013B	Receiver	User profile	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar (fixed-length).

Description: This instruction materializes the identification and the system pointers to system objects that are privately owned or that are owned by a specified user profile. The materialization options (operand 3) for the user profile (operand 2) are returned in the receiver (operand 1). The materialization options for operand 3 for the short template header have the following format:

Value (Hex)	Meaning
11	Materialize count of owned objects with no description.
12	Materialize count of authorized objects with no description (excludes owned objects).
13	Materialize count of all authorized and owned objects with no description.
21	Materialize identification of owned objects with short description.
22	Materialize identification of authorized objects with short description (excludes owned objects.)
23	Materialize identification of all authorized and owned objects with short description.
31	Materialize identification of owned objects with long description.
32	Materialize identification of authorized objects with long description (excludes owned objects).
33	Materialize identification of all authorized and owned objects with long description.

The long template header materialization options hex 51 through hex 63 are the same as the short template materialization options hex 11 through 23.

The long template header materialization options hex 71 through hex 73 are the same as the short template materialization options hex 31 through hex 33 except that context extension is materialized for each object as well.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The order of materialization is owned objects (if requested by the materialization options operand) followed by objects privately authorized to the user profile (if requested by the materialization options operand). No authorizations are stored in the system pointers that are returned.

The template identified by operand 1 must be 16-byte aligned in the space. For options hex 11 through hex 33, the short template header is materialized. It has the following format:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Number of objects owned by user profile	Bin(2)
• Number of objects privately authorized to user profile	Bin(2)
• Reserved (binary 0)	Char(4)

For options hex 51 through 73, the long template header is materialized. It has the following format:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Number of objects owned by user profile	Bin(4)
• Number of objects privately authorized to user profile	Bin(4)

If no description is requested in the materialization options parameter, the above constitutes the information available for materialization. If a description (short or long) is requested by the materialization options parameter, a description entry is present (assuming there is a sufficient sized receiver) for each object materialized into the receiver. Either of the following entries may be selected.

• Short description entry	Char(32)
– Type code	Char(1)
– Subtype code	Char(1)
– Private authorization (1 = authorized)	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Ownership (1 = yes)	Bit 8
Reserved (binary 0)	Bits 9-15
– Reserved (binary 0)	Char(12)
– System object	System pointer

• Long description entry	Char(64)
– Type code	Char(1)
– Subtype code	Char(1)
– Object name	Char(30)
– Private authorization (1 = authorized)	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Ownership (1 = yes)	Bit 8
Reserved (binary 0)	Bits 9-15
– Public authorization (1 = authorized)	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Reserved (binary 0)	Bits 8-15
– Reserved (binary 0)	Char(12)
– System object	System pointer

• Context extension (options hex 71 through hex 73)	Char(48)
– Type code	Char(1)
– Subtype code	Char(1)
– Context name	Char(30)
– Context pointer	System pointer

The context extension portion of the long description entry is optional. It is only provided as an extension to the base form of the long description entry when options hex 71 through hex 73 are requested. For these options, if the object addressed by the system pointer is not addressed by a context, the context type entry is set to hex 00 or if the object is addressed by the machine context, the context type entry is set to hex 81. Additionally, in either of these cases, the context pointer is set to the system default pointer does not exist value.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Operational
  - Operand 2

- Retrieve
  - Contexts referenced for address resolution
  - Operand 2 if materializing owned objects

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
  - Operand 2 if materializing owned objects

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0201 Machine context damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X		X	
04 Object not eligible for operation	X	X		
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X			
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MATERIALIZE AUTHORIZED USERS (MATAUU)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0143	Receiver	System object	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar (fixed-length).

Description: The instruction materializes the authorization states and the identification of the user profile(s). The materialization options (operand 3) for the system object (operand 2) are returned in the receiver (operand 1). The materialization options for operand 3 have the following format:

Value (Hex)	Meaning
11	Materialize public authority with no description.
12	Materialize public authority and number of privately authorized profiles with no description.
21	Materialize identification of owning profile with short description.
22	Materialize identification of privately authorized profiles with short description.
23	Materialize identification of owning and privately authorized profiles with short description.
31	Materialize identification of owning profile with long description.
32	Materialize identification of privately authorized profiles with long description.
33	Materialize identification of owning and privately authorized profiles with long description.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The order of materialization is an entry for the owning user profile (if requested by the materialization options operand) followed by a list (0 to n entries) of entries for user profiles having private authorization to the object (if requested by the materialization options operand). The authorization field within the system pointers will not be set.

The template identified by operand 1 must be 16-byte aligned in the space and has the following format:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Public authorization (1 = authorized)	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Reserved (binary 0)	Bits 8-15
• Number of privately authorized user profiles	Bin(2)
• Reserved (binary 0)	Char(4)

If no description is requested by the materialization options operand, the template identified by operand 1 constitutes the information available for materialization. If a description (short or long) is requested by the materialization options operand, a description entry is present (assuming there is a sufficient sized receiver) for each user profile materialized or available to be materialized into the receiver. Either of the following entry types may be selected.

• Short description entry	Char(32)
– User profile type code	Char(1)
– User profile subtype code	Char(1)
– Private authorization (1 = authorized)	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Ownership (1 = yes)	Bit 8
Reserved (binary 0)	Bits 9-15
– Reserved (binary 0)	Char(12)
– User profile	System pointer

• Long description entry	Char(64)
– User profile type code	Char(1)
– User profile subtype code	Char(1)
– User profile name	Char(30)
– Private authorization (1 = authorized)	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Ownership	Bit 8
Reserved (binary 0)	Bits 9-15
– Reserved (binary 0)	Char(14)
– User profile	System pointer

If this instruction references a temporary object, all public authority states are materialized. The privately authorized user and owner profile(s) description is not materialized (binary 0).

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Object management
  - Operand 2

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X			
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MATERIALIZE USER PROFILE (MATUP)

Op Code (Hex)	Operand 1	Operand 2
013E	Receiver	User profile

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The attributes of the user profile specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The receiver identified by operand 1 must be 16-byte aligned in the space. The following is the format of the materialized information:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attribute 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Reserved (binary 1)	Bit 2
– Reserved (binary 0)	Bits 3-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
• Reserved (binary 0)	Char(7)
• Reserved (binary 0)	Char(16)
• Reserved (binary 0)	Char(16)
• Privileged instructions (1 = authorized)	Char(4)
– Create logical unit description	Bit 0
– Create network description	Bit 1
– Create controller description	Bit 2
– Create user profile	Bit 3
– Modify user profile	Bit 4
– Diagnose	Bit 5
– Terminate machine processing	Bit 6
– Initiate process	Bit 7
– Modify resource management control	Bit 8
– Reserved (binary 0)	Bits 9-31

• Special authorizations (1 = authorized)	Char(4)
– All object authority	Bit 0
– Load (unrestricted)	Bit 1
– Dump (unrestricted)	Bit 2
– Suspend object (unrestricted)	Bit 3
– Load (restricted)	Bit 4
– Dump (restricted)	Bit 5
– Suspend object (restricted)	Bit 6
– Process control	Bit 7
– Reserved (binary 0)	Bit 8
– Service authority	Bit 9
– Reserved (binary 0)	Bits 10-23
– Modify machine attributes	Bits 24-31
Group 2	Bit 24
Group 3	Bit 25
Group 4	Bit 26
Group 5	Bit 27
Group 6	Bit 28
Group 7	Bit 29
Group 8	Bit 30
Group 9	Bit 31

Note: Group 1 requires no authorization.

• Storage authorization–The maximum amount of auxiliary storage (in units of 1024 bytes) that can be allocated for the storage of objects owned by this user profile	Bin(4)
• Storage utilization–The current amount of auxiliary storage (in units of 1024 bytes) allocated for the storage of objects owned by this user profile	Bin(4)

The attributes that the instruction can materialize are described in the Create User Profile instruction.

Authorization Required

- Operational
  - Operand 2

Lock Enforcement

- Materialize
  - Operand 2

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
03 Materialization length exception	X		

MODIFY USER PROFILE (MODUP)

Op Code (Hex)	Operand 1	Operand 2
0142	User profile	User profile modification template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: The user profile specified by operand 1 is modified in accordance with the user profile modification template specified by operand 2. The instruction replaces the privileged instruction authorizations, special authorizations, and resource authorization values in the user profile with the new values specified in the user profile template. All other values in the user profile are unchanged.

A privileged instruction exception is signaled if the instruction is operating under a user profile(s) that does not have the modify user profile privileged instruction authorization. If the instruction attempts to set a privileged instruction authorization or special authorization state for which its governing user profile(s) is not authorized, an exception will also be signaled.

No exception is signaled when the resource authorization parameter is set to a value that is less than the amount of auxiliary storage currently allocated for the storage of permanent objects owned by the user profile specified by operand 1. An exception is signaled when storage is being allocated for a permanent object and the new total exceeds the limit established by the resource authorization parameter.

Following is the format of the user profile modification template:

• Template size specification	Char(8)*
– Number of bytes provided	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)*
– Object type	Char(1)*
– Object subtype	Char(1)*
– Object name	Char(30)*
• Object creation options	Char(4)*
• Reserved (binary 0)	Char(4)*
• Size of space	Bin(4)*
• Initial value of space	Char(1)*
• Performance class	Char(4)*
• Reserved (binary 0)	Char(39)*
• Privileged instructions (1 = authorized)	Char(4)
– Create logical unit description	Bit 0
– Create network description	Bit 1
– Create controller description	Bit 2
– Create user profile	Bit 3
– Modify user profile	Bit 4
– Diagnose	Bit 5
– Terminate machine processing	Bit 6
– Initiate process	Bit 7
– Modify resource management control	Bit 8
– Reserved (binary 0)	Bits 9-31
• Special authorization (1 = authorized)	Char(4)
– All object authority	Bit 0
– Load (unrestricted)	Bit 1
– Dump (unrestricted)	Bit 2
– Suspend object (unrestricted)	Bit 3
– Load (restricted)	Bit 4
– Dump (restricted)	Bit 5
– Suspend (restricted)	Bit 6
– Process control	Bit 7
– Default owner	Bit 8
– Service authority	Bit 9
– Reserved (binary 0)	Bits 10-23
– Modify machine attributes	Bits 24-31
Group 2	Bit 24
Group 3	Bit 25
Group 4	Bit 26
Group 5	Bit 27
Group 6	Bit 28
Group 7	Bit 29
Group 8	Bit 30
Group 9	Bit 31

Note: Group 1 requires no authorization.

- Storage authorization–The maximum amount of auxiliary storage (in units of 1024 bytes) that can be allocated for the storage of permanent objects owned by this user profile — Bin(4)

- Storage utilization–The current amount of auxiliary storage (in units of 1024 bytes) allocated for storage of objects owned by this user profile — Bin(4)*

Note: The template parameters identified by an asterisk (*) are ignored by the Modify User Profile instruction.

The attributes defined in the template are included in the description of the Create User Profile instruction.

Authorization Required

- Object management
  - Operand 1

- Privileged instruction

Lock Enforcement

- Modify
  - Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
02 Privileged instruction			X
05 Create/modify user profile beyond level of authorization		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
01 Template value invalid		X	

RETRACT AUTHORITY (RETRACT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0193	User profile	System object	Authorization template

Operand 1: System pointer or null.

Operand 2: System pointer.

Operand 3: Character(2) scalar (fixed-length).

Description: When operand 1 is addressing a user profile, the private authorization states (operand 3) for the permanent system object (operand 2) will be retracted from the specified user profile. Authorization may be retracted from the owning user profile.

When operand 1 is null, the instruction is retracting public authorization. The process user profile or adopted user profile(s) currently governing the execution of the instruction when public or private authorization is being retracted must own the object specified by operand 2, have object management authority in addition to the authority being retracted, or have the all object authority special authorization.

Authorization may be retracted from the owning user profile. Ownership does not imply default authorization to a specific object except as it applies for a specific instruction. An object owner may, however, grant any object authority to any user profile, including himself.

Operand 3 is a 2-byte character scalar and employs the following bit representations to designate the authorization states to be retracted:
(1 = retract authorization)

• Authorization template	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Materialize retracted authority	Bit 8
– Reserved (binary 0)	Bits 9-15

If the materialize retracted authority bit is on (1), then operand 2 must be specified as a variable, and the authorization template must contain the authorities actually retracted from the specified user profile at completion of the instruction.

Note: Authority can be effectively retracted only if pointer authorization has never been granted to the object. A pointer with authority stored in it may be saved and used after authority has been retracted.

If this instruction references a temporary object, no operation is performed, and no exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Ownership or object management with authorization states being retracted
  - Operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modification
  - Operand 2

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
03 Attempt to grant/retract authority state to that which is not authorized			X	
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state	X	X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	

TEST AUTHORITY (TESTAU)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10F7	Available authority template receiver	System object or object template	Required authority template

Operand 1: Character(2) variable scalar or null (fixed-length).

Operand 2: System pointer or space pointer data object.

Operand 3: Character(2) scalar (fixed-length).

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
TESTAUI	18F7	Indicator
TESTAUB	1CF7	Branch

Extender: Branch or indicator options

If the branch option is specified in the op code, the extender field must be present along with one or two branch targets. If the indicator option is specified in the op code, the extender field must be present along with one or two indicator operands. The branch or indicator operands immediately follow operand 3. See *Chapter 1. Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction verifies that the object authorities and/or ownership rights specified by operand 3 are currently available to the process for the object specified by operand 2.

If operand 1 is not null, all of the authorities and/or ownership specified by operand 3 that are currently available to the process are returned in operand 1.

If an object template is not specified, operand 2 is a system pointer, the authority verification is performed relative to the invocation executing this instruction. If an object template is specified, operand 2 is a space pointer, the authority verification is performed relative to the invocation specified in the template. Specifying an invocation causes the invocations subsequent to it to be bypassed in the authority verification process. This has the influence of excluding the program adopted user profiles for any of these excluded invocations from acting as a source of authority to the authority verification process.

The required authorities and/or ownership are specified by the required authority template of operand 3. This template includes a test option that indicates whether all of the specified authorities are required or whether any one or more of the specified authorities is sufficient. This option can be used, for example, to test for operational authority by coding a template value of hex 0F01 in operand 3. Using the *any* option does not affect what is returned in operand 1. If operand 1 is not null and the *any* option is specified, all of the authorities specified by operand 3 that are available to the process are returned in operand 1.

If the required authority is available, one of the following occurs:

- Branch form indicated
  - Conditional transfer of control to the instruction indicated by the appropriate branch target operand.

- Indicator form specified
  - The leftmost byte of each of the indicator operands is assigned the following values.

    Hex F1–If the result of the test matches the corresponding indicator option

    Hex F0–If the result of the test does not match the corresponding indicator option

If no branch options are specified, instruction execution proceeds to the next instruction. If operand 1 is null and neither the branch or indicator form is used, an invalid operand type exception is signaled.

The format for the available authority template (operand 1) is as follows: (1 = authorized)

• Authorization template	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Ownership (1 = yes)	Bit 8
– Reserved (binary 0)	Bits 9-15

If operand 2 is a system pointer, it identifies the object for which authority is to be tested. If operand 2 is a space pointer, it provides addressability to the object template. The format for the optional object template is as follows:

• Object template	Char(32)
– Relative invocation	Bin(2)
– Reserved (binary 0)	Char(14)
– System object	System pointer

The *relative invocation* field in the object template identifies an invocation relative to the current invocation at which the authority verification is to be performed. The value of the relative invocation field must be less than or equal to zero. A value of zero identifies the current invocation, -1 identifies the prior invocation, -2, the invocation prior to that, and so on. A value larger than the number of invocations currently on the invocation stack or a positive value results in the signaling of the template value invalid exception. The program adopted and propagated user profiles for the identified invocation and older invocations will be included in the authority verification process. Program adopted user profiles for invocations newer than the identified invocation will not be included in the authority verification process. If the current invocation is specified, its program adopted user profile is included whether or not it is to be propagated.

The *system object* field specifies a system pointer which identifies the object for which authority is to be tested.

The format for the required authority template (operand 3) is as follows: (1 = authorized)

• Authorization template	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Ownership (1 = yes)	Bit 8
– Reserved (binary 0)	Bits 9-14
– Test option	Bit 15

0 = All of the above authorities must be present.

1 = Any one or more of the above authorities must be present.

The authority available to the process is accumulated from the following sources:

- Authority stored in the operand 2 system pointer

- Public authority to the object

- Process user profile and adopted user profiles
  - Private authorization held by these user profiles
  - Ownership, if any, if one of these user profiles owns the object
  - All authorities implied by all object special authority in any of these profiles

This instruction will tolerate a damaged object referenced by operand 2 when the reference is a resolved pointer. The instruction will not tolerate damaged contexts or programs when resolving pointers. Damaged user profiles contribute no authority to the process and are ignored.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions:

• Required authority is available.	Bin 0100
• Required authority is not available.	Bin 1100

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
02 Machine context damage state		X		
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
05 Invalid op code extender field				X
06 Invalid operand type	X	X	X	X
07 Invalid operand attribute	X		X	X
09 Invalid branch target operand				X
0C Invalid operand ODT reference	X	X	X	X
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
04 Invalid branch target				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
03 Scalar value invalid			X	
38 Template Specification				
01 Template value invalid		X		

TRANSFER OWNERSHIP (XFRO)

Op Code (Hex)	Operand 1	Operand 2
01A2	User profile	System object

Operand 1: System pointer.

Operand 2: System pointer.

Description: The ownership of a system object (operand 2) is transferred to the user profile (operand 1). A user profile with all object authority may always transfer ownership of an object. If a program which adopts a user profile is being transferred, all object authority is required. After ownership is transferred, the former owning user profile retains the private object authorities it had before the transfer. The new owner is implicitly granted all of the object authorities to the transferred object. All other user profile authorities are unchanged as a result of this instruction.

An attempt to transfer ownership of a temporary object causes the object ineligible for operation exception to be signaled.

Authorization Required

- Object control
  - Operand 2

- Retrieve
  - Contexts referenced for address resolution

- Delete
  - User profile owning operand 2

- Insert
  - Operand 1

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Operand 1
  - Operand 2
  - User profile owning the object referenced by operand 2

Events

000F Ownership
0101 Ownership changed

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
02 Object exceeds available space	X		
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage Encountered			
02 Machine context damage state			X
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X	X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded	X		
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
04 Object not eligible for operation	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	

Chapter 8. Program Management Instructions

This chapter describes all instructions used for program management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CREATE PROGRAM (CRTPG)

Op Code (Hex)	**Operand 1**	**Operand 2**
023A	Program	Program Template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: A program is created from the program template (operand 2), and a system pointer to the created program is returned in operand 1.

The program template (operand 2) has the following format:

• Control information	
– Template size specification	Char(8)
Number of bytes provided	Bin(4)
Number of bytes available for materialization (used only when the program is materialized)	Bin(4)*
– Program identification	Char(32)
Type	Char(1)*
Subtype	Char(1)
Name	Char(30)
– Program creation options	Char(4)
Existence attributes	Bit 0
0 = Temporary	
1 = Permanent	
Space attribute	Bit 1
0 = Fixed-length	
1 = Variable-length	
Initial context	Bit 2
0 = Do not insert addressability into context	
1 = Insert addressability into context	
Access group	Bit 3
0 = Do not create as a member of an access group	
1 = Create as a member of an access group	
Reserved (binary 0)	Bits 4-31
– Reserved (binary 0)	Char(4)
– Size of space	Bin(4)
– Initial value of space	Char(1)
– Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, a zero value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	

– Transient storage pool selection	Bit 6
0 = Default main storage pool (process default or machine default as specified for main storage pool selection) is used for object.	
1 = Transient storage pool is used for object.	
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
– Reserved (binary 0)	Char(7)
– Context	System pointer
– Access group	System pointer
– Program attributes	Char(2)
Adopted user profile	Bit 0
0 = No adoption of user profile.	
1 = Adopt program owner's user profile on invocation.	
Array constraint	Bit 1
0 = Arrays are constrained.	
1 = Arrays are not constrained.	
String constraint	Bit 2
0 = Strings are constrained.	
1 = Strings are not constrained.	
User exit	Bit 3*
0 = Not allowed as user exit	
1 = Allowed as user exit	
Adopted user profile propagation	Bit 4
0 = Adopted user profile authorities are not propagated to external invocations.	
1 = Adopted user profile authorities are propagated to all subinvocations.	
Static storage	Bit 5
0 = Initialize storage to binary 0.	
1 = Do not initialize storage to binary 0.	
Automatic storage	Bit 6
0 = Initialize storage to binary 0.	
1 = Do not initialize storage to binary 0.	
Associated journal entry	Bit 7
0 = Program name is recorded in journal entries	
1 = Program name is not recorded in journal entries	
Update PASA stack	Bit 8
0 = Program requires PASA stack update	
1 = Program does not require PASA stack update	
Reserved (binary 0)	Bits 9-11
Template version	Bits 12-15
0000=Version 0	
0001=Version 1	
0010 through 1111 reserved	
– Optimization options	Char(1)
Optimization	Bit 0
0 = No optimization	
1 = Perform optimization	
Space pointer machine objects	Bit 1
0 = Disallow space pointer machine objects in ODV component	
1 = Allow space pointer machine objects in ODV component	
Coincident operand overlap	Bit(2)
0 = Do not assume coincident operand overlap	
1 = Assume coincident operand overlap	
Reserved	Bits 3-7
– Observation attributes	Char(1)
Hex 00=Program data cannot be materialized	
Hex FC=Program data can be materialized	
– Size of static storage	Bin(4)
– Size of automatic storage	Bin(4)
– Template version 0 sensitive For version number 0, this field indicates the number of instructions. For version number 1, this field is reserved (binary 0).	Bin(2)
– Template version 0 sensitive For version number 0, this field indicates the number of ODV entries. For version number 1, this field is reserved (binary 0).	Bin(2)

  - Offset (in bytes) from beginning of template to the instruction stream component Bin(4)
  - Offset (in bytes) from beginning of template to the ODV component Bin(4)
  - Offset (in bytes) from beginning of template to the OES component Bin(4)
  - User data part 3 Char(4)
  - Length of data part 1 Bin(4)
  - Offset (in bytes) from beginning of template to the user data part 1 Bin(4)
  - User data part 4 Char(4)
  - Length of user data part 2 Bin(4)
  - Offset (in bytes) from beginning of template to the user data part 2 Bin(4)
  - Offset (in bytes) from beginning of template to the object mapping table (OMT) component Bin(4)*
  - Template version 1 sensitive Bin(4)
    For version number 1, this field indicates the number of instructions.
    For version number 0, this field is reserved (binary 0).
  - Template version 1 sensitive Bin(4)
    For version number 1, this field indicates the number of ODV entries.
    For version number 0, this field is reserved (binary 0).
- Program data
  - Instruction stream component
  - ODV component
  - OES component

- User data parts 1 and 2

- Object mapping table*

Note: The value associated with the template entry annotated with an asterisk (*) is ignored by the instruction.

The template identified by operand 2 must be 16-byte aligned.

If the created object is permanent, it is owned by the user profile governing process execution. The owning user profile is implicitly assigned all private authority states for the object. The storage occupied by the created object is charged to this owning user profile. If the created object is temporary, there is no owning user profile, and all authority states are assigned as public. Storage occupied by the created context is charged to the creating process.

The order and location of the program data and the user defined data in the template are established by the control information parameters. The entries in the parameter need not be contiguous, but the number of bytes provided entry must include any unused bytes between entries.

The program identification specifies the symbolic name that identifies the program within the machine. A type code of hex 02 is implicitly supplied by the machine. The program identification is used to identify the program on materialize instructions as well as to locate the program in the context that addresses it.

The existence attribute specifies whether the object is to be temporary or permanent. A temporary program object, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated. A permanent program object exists in the machine until explicitly destroyed by the user.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created program is to be placed. Addressability is inserted into the context based on the object identification (type, subtype, and name). If addressability is not to be inserted into a context, the context entry is ignored.

If the access group creation attribute entry indicates that the object is to be created in an access group, the access group entry must contain a system pointer that identifies an access group in which the object is to be created. The existence attribute of the object must be temporary because access groups are temporary objects. If the object is not to be created in an access group, the access group entry is ignored.

A space may be associated with the created program. The space may be fixed or variable in size. The initial allocation is as specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated depends on an algorithm defined for a specific implementation. A fixed size space of zero length causes no space to be allocated.

Each byte of the space is initialized to the value specified by the initial value of space entry. When the space is extended, this byte value is also used to initialize the new allocation. If no space is allocated, this value is ignored.

The performance class parameter provides information that allows the machine to more effectively manage the program by considering overall performance objectives of operations involving the program.

If the adopted user profile attribute is specified, any reference to a system object from an invocation of this program uses the user profile of the owner of this program and other sources of authority to determine the authorization to system objects, privileged instructions, ownership rights, and all authorizations. If the adopted user profile propagation attribute is specified, then the authorities available from the adopted user profile are available to any further invocations while this program is invoked. If the adopted user profile propagation attribute is not specified, then the authorities available to the program's owning user profile are not available to further subinvocations and are available only to this invocation. These attributes do not affect the propagation of authority from higher existing invocations. The adopted user profile propagation attribute must not be specified if this program does not have the adopted user profile specified; otherwise, a template value invalid exception is signaled.

If constrainment (string or array) is not specified, the references are assumed to be within the defined bounds of the array or string. No execution time checks are performed to ensure this is the case. However, if the reference is outside the defined bounds, unpredictable results may occur. There may be significant savings in performance if constrainment is not specified.

The user exit attribute is ignored when the program is created, but is an attribute that can be materialized by specifying that the program is allowed to be referenced as a user exit program.

Whenever a new invocation or activation is allocated, the automatic or static storage areas are initialized to bytes of binary 0's, respectively. The static storage and automatic storage program attributes control this default initialization. There is a significant performance advantage when these areas are not initialized by default. However, initial values specified for individual data objects are still set.

The associated journal entry attribute controls which program is associated with a journal entry. As a journal entry is made, a newest-to-oldest interrogation of the invocation stack is performed. The first program encountered that has the associated journal entry attribute specified is associated with the journal entry by a record of the program name in the journal entry. If a program is encountered for which the associated journal entry attribute is not specified, the program is ignored unless the program is on the top of the invocation stack. If the program is on the top of the invocation stack, it is associated with the journal entry by a record of the program name in the journal entry.

The update PASA stack attribute specifies whether or not the program requires that the PASA stack information be updated when it is invoked. This attribute allows a program which has no dependency on the stack information in the PASA base entry and the PASA invocation entries to avoid the overhead associated with updating this information upon invocation of the program. This attribute applies only to the invocation of the program specifying the attribute. It is not propagated to subsequent invocations. Refer to the description of the Call External and Return External instructions for additional detail on the attribute.

The optimization options provide information that allows the machine to create a program based on the intended use of the program.

If the performance optimization attribute is specified, additional processing is performed which provides for creating a program that requires less processor resource for execution. This optimizes addressability used within the program. If this attribute is not specified, normal optimization processing is performed.

If the allow space pointer machine objects in ODV component attribute is specified, additional processing is performed which allows for space pointer machine objects within the program. If this attribute is not specified, space pointer machine objects are not allowed in the ODV component.

The coincident operand overlap optimization option controls whether or not additional processing is performed during the encapsulation of certain computation and branching instructions which affects the processor resource required to execute these instructions. The effect of the option controls whether or not the encapsulation process for these instructions should assume that coincident operand overlap may occur between the source and receiver operands during execution of the instruction. This assumption applies to cases of nonidentical coincident operand overlap where the Create Program instruction cannot determine if coincident operand overlap may occur during execution of the instruction. These instructions may produce invalid results if nonidentical coincident overlap occurs during execution, but the instruction was encapsulated with the assumption that it would not occur.

Specifying the do not assume coincident operand overlap attribute indicates that nonidentical coincident overlap will not occur during execution and therefore the receiver on an instruction may be used as a work area during operations performed to produce the final result. Using the receiver as a work area does not require the processor resource that would be required to move the final result from an internal work area to the receiver.

Specifying the assume coincident operand overlap attribute indicates that nonidentical coincident operand overlap may occur during execution and therefore the receiver on an instruction should not be used as a work area during operations that produce the final result. This can require more processor resource for instruction execution but it insures valid results if overlap occurs.

The following is a list of instructions that can be affected by the coincident operand overlap optimization option during the encapsulation process:

- Add Logical Character
- Add Numeric
- And
- Compute Math Function Using One Input Value
- Concatenate
- Convert Character To Numeric
- Convert Decimal Form To Floating-Point
- Convert External Form To Numeric Value
- Convert Floating-Point To Decimal Form
- Convert Numeric To Character
- Copy Bytes Left Adjusted With Pad
- Copy Bytes Right Adjusted With Pad
- Divide
- Divide With Remainder
- Exclusive OR
- Multiply
- Or
- Remainder
- Scale
- Subtract Logical Character
- Subtract Numeric
- Trim Length

If the observation attribute is specified, the program data in the program template is available through the Materialize Program instruction.

Less storage is used by the program when the program is created without the capability to materialize. If the program is created without the capability to materialize, the program data (instruction stream, ODV, OES, break offset mapping table, symbol table, and object mapping table components) cannot be materialized by the Materialize Program instruction.

The size of static storage entry consists of a 4-byte binary value that defines the total amount of static storage required for this program's static data. A value of 0 indicates that the amount of static storage required is to be calculated by the Create Program instruction based upon the amount of static data specified for the program. A value greater than 0 specifies the amount of static storage required, and that value must be sufficient to provide for the amount of static data specified for the program. If it is not, a create program exception is signaled.

The size of automatic storage entry consists of a 4-byte binary value that defines the total amount of automatic storage required for this program's automatic data. A value of 0 indicates that the amount of automatic storage required is to be calculated by the Create Program instruction based upon the amount of automatic data specified for the program. A value greater than 0 specifies the amount of automatic storage required, and that value must be sufficient to provide for the amount of automatic data specified for the program. If it is not, a create program exception is signaled.

The template version attribute is used to define different versions of the program template. Template version 0 limits the number of instructions to a maximum of 32 767 and the number of ODV entries to a maximum of 8191. Programs that exceed one of these maximums cannot be created with template version 0. Template version 1 limits the number of instructions to a maximum of 32 767 and the number of ODV entries to a maximum of 65 526. Programs that exceed one of these maximums cannot be created with template version 1. All other values for the template version are reserved.

The instruction stream component consists of a 4-byte binary value that defines the total length of the instruction stream component and a variable-length vector of 2-byte entries that defines the instruction stream. The 2-byte entries define instruction operation codes, instruction operation code extenders, or instruction operands.

The format of the instructions is defined in Chapter 1. *Introduction*. The instruction stream component is optional (that is, instructions need not be defined), and its absence is indicated by a value of 0 in the offset to instruction stream component entry. If the instruction stream is not present, an End instruction is assumed and, should the program be executed, an immediate Return External instruction results.

The ODV (object definition vector) component consists of a 4-byte binary value that defines the total length of the ODV and a variable-length vector of 4-byte entries. Each entry describes a program object either by a complete description or through an offset into the OES (object entry string) to a location that contains a description. If no program objects are defined, the ODV can be omitted, and its absence is noted with a value of 0 in the offset to ODV component entry. The ODV is required if the OES is present.

The OES consists of a 4-byte binary value that defines the total length of the OES and a series of variable-length entries that are used to complete an object description. Entries in the ODV contain offsets into the OES. The OES is optional, and its absence is indicated with a value of 0 in the offset to OES component entry.

The format of the ODT (object definition table) (ODV and OES) is defined in *Chapter 23. Program Object Specifications*.

The user data components can be used by compilers to relate high-level language statement numbers to instruction numbers and high-level language names to ODT numbers. The format of the user data components is defined by the user.

The object mapping table is a component constructed by the machine and is available through the Materialize Program instruction. It describes the location of pointers and scalars that are defined in the program. See the Materialize Program instruction for a description of this component.

Authorization Required

- Insert
  - User profile of creating process
  - Context identified by operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - User profile of creating process
  - Context identified by operand 2
  - Access group identified by operand 2

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
02 Program limitation exceeded		X	
03 Machine storage limit exceeded		X	X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
01 Program header invalid		X	
02 ODT syntax error		X	
03 ODT relation error		X	
04 Operation code invalid		X	
05 Invalid op code extender field		X	
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
09 Invalid branch target operand		X	
0A Invalid operand length		X	
0B Invalid number of operands		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
06 Instruction cancellation			X
07 Instruction termination			X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
38 Template Specification			
01 Template value invalid		X	
02 Template size invalid		X	

DELETE PROGRAM OBSERVABILITY (DELPGOBS)

Op Code (Hex)	Operand 1
0211	Program

Operand 1: System pointer.

Description: The instruction eliminates the capability to materialize the components, other than the control information component, of the program template associated with the program identified by operand 1. After deleting observability, only the control information component of the program template can be materialized.

In general, the instruction causes the amount of storage used by the referenced program to be decreased. The amount of storage released is equal to the size of the program template and all of its components.

Authorization Required

- Object control
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Object control
  - Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

DESTROY PROGRAM (DESPG)

Op Code (Hex)	Operand 1
0221	Program

Operand 1: System pointer.

Description: The program referenced by the system pointer specified by operand 1 is destroyed. The program's identification is deleted from the context currently addressing the object if it is addressed by a context. The system pointer identified by operand 1 is not modified by the instruction. Any subsequent reference to the destroyed object through the pointer causes the object destroyed exception.

If the referenced program is currently activated in some process, an attempt to invoke the program causes the object destroyed exception to be signaled. If the referenced program is currently invoked in some process, execution of the next instruction in the program causes the object destroyed exception. Any use of an unresolved pointer that has its initial value specified by this referenced program causes an object destroyed exception.

Authorization Required

- Object control
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Object control
  - Operand 1
- Modify
  - Access group containing operand 1
  - Context which addresses operand 1
  - User profile owning operand 1

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

MATERIALIZE PROGRAM (MATPG)

Op Code (Hex)	Operand 1	Operand 2
0232	Attribute receiver	Program

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The program identified by operand 2 is materialized into the template identified by operand 1.

Operand 2 is a system pointer that identifies the program to be materialized. The format of the materialization is identical to the program template identified on the Create Program instruction. The values in the materialization relate to the current attributes of the materialized program. Components of the program template, other than the control information component, may not be available for materialization because they were removed by the Delete Program Observability instruction or because they were absent from the Create Program instruction.

The template identified by operand 1 must be 16-byte aligned.

The first 4 bytes of the materialization template identify the total number of bytes in the template. This value is supplied as input to the instruction and is not modified. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization template are modified by the instruction to contain a value identifying the template size required to provide for the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified by the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The following attributes apply to the materialization of a program:

- The existence attribute indicates whether the program is temporary or permanent.

- The observation attribute entry specifies the template components of the programs that currently can be materialized.

- If the program has an associated space, then the space attribute is set to indicate either fixed- or variable-length; the initial value for the space is returned in the initial value of space entry, and the size of space entry is set to the current size value of the space. If the program has no associated space, the size of space entry is set to a zero value, and the space attribute and initial value of space entry values are meaningless.

- If the program is addressed by a context, then the context addressability attribute is set to indicate this, and a system pointer to the addressing context is returned in the context entry. If the program is not addressed by a context, then the context addressability attribute is set to indicate this, and binary 0's are returned in the context entry.

- If the program is a member of an access group, then the access group attribute is set to indicate this, and a system pointer to the access group is returned in the access group entry. If the program is not a member of an access group, then the access group attribute is set to indicate this, and binary 0's are returned in the access group entry.

- The performance class entry is set to reflect the performance class information associated with the program.

- The user exit attribute defines if the referenced program is allowed to be used as a user exit program.

The program data cannot be materialized if a Delete Program Observability instruction has been issued for this program. If the program was created with an observation attribute that cannot be materialized, the program data (instruction stream, ODV, OES, user data, and object mapping table components) cannot be materialized by this instruction. If the program data cannot be materialized, 0's are placed in the fields of the program template that describe the size and offsets to the program data components. The only information that can be materialized is that part of the program template up to and including the offset to the OMT (object mapping template) entry.

The offset to the OMT component entry specifies the location of the OMT component in the materialized program template. The OMT consists of a variable-length vector of 6-byte entries. The number of entries is identical to the number of ODV entries because there is one OMT entry for each ODV entry. The OMT entries correspond one for one with the ODV entries; each OMT entry gives a location mapping for the object defined by its associated ODV entry.

The following describes the formats for an OMT entry:

- OMT entry Char(6)
  - Addressability type Char(1)

    Hex 00= Base addressability is from the start of the static storage area.

    Hex 01= Base addressability is from the start of the automatic storage area.

    Hex 02= Base addressability is from the start of the storage area addressed by a space pointer.

    Hex 03= Base addressability is from the start of the storage area of a parameter.

    Hex 04= Base addressability is from the start of the storage area addressed by the space pointer found in the process communication object attribute of the process executing the program.

    Hex FF= Base addressability not provided. The object is contained in machine storage areas to which addressability cannot be given, or a parameter has addressability to an object that is in the storage of another program.
  - Offset from base Char(3)

    For types hex 00, hex 01, hex 02, hex 03, and hex 04, this is a 3-byte logical binary value representing the offset to the object from the base addressability. For type hex FF, the value is binary 0.
  - Base addressability Char(2)

    For types hex 02 and hex 03, this is a 2-byte binary field containing the number of the OMT entry for the space pointer or a parameter that provides base addressability for this object. For types hex 00, hex 01, hex 04, and hex FF, the value is binary 0.

Authorization Required

- Retrieve
  - Operand 2
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X		
0D Reserved bits are not zero	X	X	X
38 Template Specification			
03 Materialization length exception	X		

Chapter 9. Program Execution Instructions

This chapter describes the instructions used for program execution control. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary.*

ACTIVATE PROGRAM (ACTPG)

Op Code (Hex)	Operand 1	Operand 2
0212	Program or program activation entry	Program

Operand 1: Space pointer, system pointer, or data object.

Operand 2: System pointer.

Description: This instruction allocates and initializes storage for static objects that are declared for a specified program within the executing process. The program identified by operand 2 is activated in the executing process. The program is activated by allocating an area in the PSSA (process static storage area) to contain the program static storage. This static storage is then available each time the program is invoked within the process. The pointer object specified by operand 1 receives a space pointer addressing the activation of the referenced program. The activation consists of storage for the program's static objects as well as a system pointer to the associated program, a space pointer to the next activation entry (if one exists) in the PSSA, a space pointer to the preceding activation entry in the PSSA, and attributes specifying the status of the activation.

Each activation entry in the PSSA is 16-byte aligned and has the following format:

• Previous activation entry pointer (the first activation entry locates the PSSA base entry)	Space pointer
• Next activation entry pointer (undefined if this activation is last in the PSSA chain)	Space pointer
• Associated program pointer	System pointer
• Activation number	Bin(2)
• Activation attributes	Char(2)
– Activation status 0 = Not currently active 1 = Currently active	Bit 0
– Reserved (binary 0)	Bits 1-15
• Reserved (binary 0)	Char(2)
• Invocation count	Bin(2)
• Activation mark	Bin(4)
• Length of this PSSA entry	Bin(4)
• Program static storage	Char(*)

The PSSA is located by a space pointer specified when the process was initiated. The location identified by the space pointer is considered to be the beginning of the PSSA and must be 16-byte aligned. At this location is a 96-byte PSSA base entry that consists of the following:

• Last activation entry in process PSSA chain (addresses the base entry if no programs are activated)	Space pointer
• First activation entry in process (ignored if no programs are activated)	Space pointer
• Next available storage location in current space containing PSSA	Space pointer
• Reserved	Char(16)
• PSSA control	Char(1)
– Chain being modified	Bit 0
0 = Chain not being modified	
1 = Chain being modified	
– Chain was modified	Bit 1
0 = Chain was not modified	
1 = Chain was modified	
– Reserved (binary 0)	Bits 2-7
• Reserved (binary 0)	Char(31)

The user must properly initialize the PSSA base entry before the first program is activated in the process.

A space pointer locating the PSSA can be materialized using the Materialize Process instruction.

If the chain being modified bit is on and an attempt is made to activate or de-activate a program with static storage, a stack control invalid exception is signaled.

The program is activated by allocating an area in the PSSA space sufficient to contain the activation entry. The area used for allocating the first activation in a space is located by the next available storage location pointer in the PSSA base entry; otherwise, this pointer locates the first free byte after all activation entries in the space. This pointer must address a 16-byte aligned area in the space, or a boundary alignment exception is signaled. The pointer may be set to address beyond the currently allocated storage in the space, which is implicitly extended, and no exception is signaled. If the space is not currently large enough to contain the entry and if it is extendable, it is implicitly extended by the machine. The owner's authority to the space is included with the authority of the extending process when checking for object management authority when the space is extended. If the space is of a fixed size or cannot be extended to contain the entry, a space extension truncation exception is signaled.

The new activation entry is initialized as follows:

- The previous activation entry pointer is copied from the most recent activation entry in the PSSA base entry.
- The next activation entry pointer field is unchanged by the instruction (the last activation is process pointer in the PSSA base entry specifies the last activation on the chain).
- The associated program pointer is copied from the operand 2 system pointer.
- The activation number is set to a value one greater than the activation number entry in the previous activation.
- The activation is marked as active (the activation status is set to binary 1).
- The invocation count is set to 0.
- The activation mark is obtained by incrementing the mark counter field in the machine by one and copying the resulting value.
- The length field is set to the number of bytes of storage occupied by the PSSA header and the static data following it.
- The reserved fields are set to binary 0.

A space pointer addressing the new activation entry is stored in the last activation entry pointer of the PSSA base entry, and the next available storage location in the PSSA base entry is set to address the next available 16-byte aligned area beyond the new activation entry.

If the referenced program's activation already exists within the process PSSA chain when the Activate Program instruction is executed, the program's static storage is reused if the activation was active, and may or may not be reused if the activation was inactive. In either case, the storage is reinitialized, the activation is set to the active state, and the operand 1 space pointer is set to the reinitialized activation. No chain pointers are modified, and the activation entry remains at the same relative location in the chain of PSSA entries.

When a new activation is allocated or an existing inactive allocation is reactivated, the mark counter in the machine is incremented by 1 and the resulting value is copied to the active mark field of the activation. If an attempt is made to activate an already active activation, the activation mark and mark counter values are not updated.

When a new activation is allocated, space occupied by other activations in the inactive state may be used for the new activation. The current PSSA space is the space located by the next available location pointer within the PSSA base entry.

PSSA entries that have all the following conditions are removed from the PSSA chain:

- Inactive

- Reside in the current PSSA space

- Have an invocation count of 0

- Have no active activations or activations with a nonzero invocation count at a higher address in the current PSSA space

- Appear as the last entries in the linked PSSA chain

The new activation is placed at the lowest address within the current PSSA space that is higher than both the address of any activation in the chain which is in the current PSSA space and the address of any unallocated space between previously existing noncontiguous activations. If no previous activations remain in the current PSSA space (after being removed under the above conditions), the new activation is placed at the lowest address (in the current PSSA space) of the removed activations. If no previous activations existed in the current PSSA space, the next available location pointer in the PSSA base entry specifies the location where the new activation is to be allocated.

If the program addressed by the operand 2 system pointer addresses a program that requires no static storage, no activation entry is allocated, and the operand 2 system pointer is copied to the operand 1 pointer.

A space pointer machine object may not be specified for operand 1.

Authorization Required

- Operational
  - Program referenced by operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
- 0101 Object authorization violation

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object state	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist		X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Exception			
03 Stack control invalid			X
36 Space Management			
01 Space extension/truncation			X

CALL EXTERNAL (CALLX)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0283	Program	Argument list	Return list

Operand 1: System pointer.

Operand 2: Operand list or null.

Operand 3: Instruction definition list or null.

Description: The instruction preserves the calling invocation and causes control to be passed to the external entry point of the program specified by operand 1. Operand 1 is a system pointer addressing the program that is to receive control.

The instruction ensures that the program is properly activated in the process, if required. The following conditions are allowed:

- If the referenced program requires no static storage, the program is invoked, and no activation is created.

- If operand 1 is a system pointer to a program that requires static storage, the program is implicitly activated. The chain of activation entries located by the PSSA (process static storage area) is searched for an entry for the referenced program. If an entry is located that is not active, it is set to the active state, and the static storage is reinitialized based on the program definition. If no activated entry exists for the program, a new entry is allocated and initialized. See the Activate Program instruction for a definition of this function. The activation mark value for a newly created activation will be the same as the invocation mark value described later.

After any needed static storage has been allocated or located, automatic storage is allocated and initialized for the newly invoked program. The automatic storage is obtained from the PASA (process automatic storage area).

The update PASA stack program attribute specified on program creation indicates whether or not the program requires that the PASA stack information contained in the PASA base entry and invocation entries must be updated. Refer to the Create Program instruction for the detail on how to specify this program attribute. Upon invocation of a program that requires that the stack be updated, it is possible that prior invocations may exist that did not require the stack update. These invocations would not have their associated stack information updated to reflect the current chain of invocations active in the PASA. If necessary, the PASA stack information in the PASA base entry and all prior invocation entries is updated with the current status prior to continuing with the invocation of a program requiring update of the PASA stack.

Each invocation entry in the PASA is 16-byte aligned and has the following format:

• Previous invocation entry pointer (the first invocation entry addresses the PASA base entry)	Space pointer
• Next invocation entry pointer (not defined for the current invocation entry)	Space pointer
• Associated program pointer (0 for data base select/omit program)	System pointer
• Invocation attributes	Char(8)
– Invocation number	Bin(2)
– Invocation type Hex 00=Data base select/omit program Hex 01=Call external Hex 02=Transfer control Hex 03=Event handler Hex 04=External exception handler Hex 05=Initial program in process problem state Hex 06=Initial program in process initiation state Hex 07=Initial program in process termination state Hex 08=Invocation exit	Char(1)
– Reserved (initialized to binary 0)	Char(1)
– Invocation mark	Bin(4)

• User area	Char(8)
• Program's automatic storage	Char(*)

The PASA is located by a space pointer specified when the process is initiated. The location identified by the space pointer is considered to be the beginning of the PASA and must be 16-byte aligned. At this location is a 64-byte PASA header entry that consists of the following:

• Current invocation entry in process (if no programs are invoked, this pointer must address the PASA base entry)	Space pointer
• First invocation entry in process (ignored if no programs are invoked)	Space pointer
• Next available storage location	Space pointer
• Reserved	Char(16)
• Reserved (binary 0)	Char(12)
• Mark counter	Bin(4)
• Reserved (binary 0)	Char(16)

The PASA base entry must be initialized by the user before the process is initiated. The current invocation entry in process and next available storage location and mark counter values are accessed as input to the machine only during the initiation of the process. Thereafter, the machine maintains these values internally. The PASA base entry fields are optionally updated on each program invocation depending upon whether or not the program being invoked has specified the update PASA stack program attribute.

A space pointer locating the PASA can be materialized by using the Materialize Process instruction.

A space pointer locating the PASA invocation entry for the currently executing program can be materialized using the Materialize Invocation Entry instruction.

The program is invoked by allocating an area in the PASA space sufficient to contain the invocation entry. The area used for allocation is located by the next available storage location pointer in the PASA base entry for the invocation of the initial program in the process. For all other invocations of programs within the process, the area used for the allocation is located by an internal machine value that is maintained with the space address of the next available storage location. This pointer must address a 16-byte aligned area in the space, or a boundary alignment exception is signaled. If the space is not currently large enough to contain the entry and if it is extendable, it is implicitly extended by the machine. The owner's authority to the space is included with the authority of the process when checking for object management authority when the space is extended. If the space is of a fixed size or cannot be extended enough to contain the entry, a space extension/truncation exception is signaled.

For programs created with the update PASA stack attribute specifying that they require the PASA stack update, the new invocation entry is updated as follows:

- The previous invocation entry pointer is set from the current invocation entry in the process address value in the machine.

- The next invocation entry is not modified.

- The associated program pointer is copied from the operand 1 system pointer.

- The invocation number is incremented by 1 beyond that in the calling invocation. The first invocation in the current process state has an invocation number of 1.

- The invocation type value is set to hex 01 to indicate how the program was invoked.

- The value of the mark counter in the machine is incremented by one and the new value is copied to the invocation mark field. The new value is also copied to the activation mark field of the program's activation if the activation was initialized by this instruction.

- The user area field is set to binary 0.

- The program's automatic storage is initialized as defined in the program definition.

- The invocation count, if any, in the associated activation is incremented by 1.

For programs created with the update PASA stack attribute specifying that they do not require the PASA stack update, the new invocation entry is updated as follows:

- The value of the mark counter in the machine is incremented by one. The new value is also copied to the activation mark field of the program's activation if the activation was initialized by this instruction.

- PASA stack information necessary to provide for subsequent program invocations or updating of stack information for this invocation is stored in the machine. This includes values associated with this invocation for the previous invocation entry address, next available storage location, program pointer, invocation number, invocation type, and mark counter.

- The program's automatic storage is initialized as defined in the program definition.

For programs created with the update PASA stack attribute that specifies that they require the PASA stack update, a space pointer addressing the new invocation entry is stored in the next invocation entry pointer of the invoking invocation.

For programs created with the update PASA stack attribute that specifies that they require the PASA stack update, a space pointer addressing the new invocation entry is stored in the current invocation entry pointer of the PASA base entry and the next available storage location in the PASA base entry is set to address the next available 16-byte aligned area beyond the new invocation entry.

A program with no automatic data has a PASA entry created for it. The created PASA entry consists of only a stack control entry.

The user defines the invocation attribute entry. This entry is not used after the program is initialized.

Following the allocation and initialization of the invocation entry, control is passed to the invoked program.

Operand 2 specifies an operand list that identifies the arguments to be passed to the invocation entry to be called. If operand 2 is null, no arguments are passed by the instruction. A parameter list length exception is signaled if the number of arguments passed does not correspond to the number required by the parameter list of the target program.

Operand 3 specifies an IDL (instruction definition list) that identifies the instruction number(s) of alternate return points within the calling invocation. A Return External instruction in an invocation immediately subordinate to the calling invocation can indirectly reference a specific entry in the IDL to cause a return of control to the instruction associated with the referenced IDL entry. If operand 3 is null, then the calling invocation has no alternate return points associated with the call.

Authorization Required

- Operational
  - Program referenced by operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

- 0002 Authorization
  - 0101 Object authorization violation
- 000C Machine resource
  - 0201 Machine auxiliary storage threshold exceeded
- 0010 Process
  - 0701 Maximum processor time exceeded
  - 0801 Process storage limit exceeded
- 0016 Machine observation
  - 0101 Instruction reference
- 0017 Damage set
  - 0401 System object damage set
  - 0801 Partial system object damage set

Exceptions

Exception	**Operands** 1	2	3	**Other**
06 Addressing				
01 Space addressing violation	X			
02 Boundary alignment	X			
03 Range	X			
06 Optimized addressability invalid	X			
08 Argument/Parameter				
01 Parameter reference violation	X			
02 Parameter list length violation		X		
0A Authorization				
01 Unauthorized for operation	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state	X			
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X			
02 Object destroyed	X			
03 Object suspended	X			
24 Pointer Specification				
01 Pointer does not exist	X			
02 Pointer type invalid	X			
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X			
08 Invalid operand value range	X			
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
03 Stack control invalid				X
36 Space Management				
01 Space extension/truncation				X

CALL INTERNAL (CALLI)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0293	Internal entry point	Argument list	Return target

Operand 1: Internal entry point.

Operand 2: Operand list or null.

Operand 3: Instruction pointer.

Description: The internal entry point specified by operand 1 is located in the same invocation in which the Call Internal instruction is executed. A subinvocation is defined, and execution control is transferred to the first instruction associated with the internal entry point. The instruction does not cause a new invocation to be established. Therefore, there is no allocation of objects, and instructions in the subinvocation have access to all invocation objects.

Operand 2 specifies an operand list that identifies the arguments to be passed to the subinvocation. If operand 2 is null, no arguments are passed. After an argument has been passed on a Call Internal instruction, the corresponding parameter may be referenced. This causes an indirect reference to the storage area located by the argument. This mapping exists until the parameter is assigned a new mapping based on a subsequent Call Internal instruction. A reference to an internal parameter before its being assigned an argument mapping causes a parameter reference violation exception to be signaled.

Operand 3 specifies an instruction pointer that identifies the pointer into which the machine places addressability to the instruction immediately following the Call Internal instruction. A branch instruction in the called subinvocation can directly reference this instruction pointer to cause control to be passed back to the instruction immediately following the Call Internal instruction.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation			X	
02 Boundary alignment			X	
03 Range			X	
06 Optimized addressability invalid			X	
08 Argument/Parameter				
01 Parameter reference violation		X		
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
24 Pointer Specification				
01 Pointer does not exist			X	
02 Pointer type invalid			X	
2A Program Creation				
06 Invalid operand type	X	X	X	
09 Invalid branch target			X	
0B Invalid number of operands		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X

CLEAR INVOCATION EXIT (CLRIEXIT)

Op Code (Hex)
0250

Description: The instruction removes the invocation exit program for the requesting invocation. No exception is signaled if an invocation exit program is not specified for the current invocation. Also, an implicit clear of the invocation exit occurs when the invocation exit program is given control, or the program which set the invocation exit completes execution.

For more information about the invocation exit program, see *Program Execution* in the *Functional Concepts Manual.*

Events

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	**Other**
10 Damage Encountered	
04 System object damage state	X
44 Partial system object damage	X
1C Machine-Dependent Exception	
03 Machine storage limit exceeded	X
20 Machine Support	
02 Machine check	X
03 Function check	X
2A Program Creation	
0D Reserved bits are not zero	X

DE-ACTIVATE PROGRAM (DEACTPG)

Op Code (Hex)	**Operand 1**
0225	Program

Operand 1: System pointer or null.

Description: The instruction locates the activation entry addressed through operand 1 and marks it as inactive if the appropriate conditions are satisfied.

If operand 1 is null, the program issuing the instruction is to be de-activated. An activation in use by invocation exception is signaled if the activation entry's invocation count is not equal to 1.

If operand 1 is a system pointer to a program, then that program's activation entry is de-activated if its invocation count is 0. Otherwise, an activation in use by invocation exception is signaled.

In the previous two cases, if the program has no static storage or no activation, no operation is performed and no exception is signaled.

The activation is de-activated when the activation status is set to not currently active (0). When the activation is not active and its invocation count is 0, the storage occupied by the activation is subject to reuse for allocating other activations.

If the user de-activates a program by setting the activation status bit with an instruction other than the De-activate Program instruction, the following steps must be taken to ensure proper stack operation:

1. The chain being modified and the chain was modified bits must be turned on in the PSSA base entry.

2. The contents and linking of the PSSA chain of activation headers can be modified as necessary.

3. The chain being modified bit must be turned off.

4. The machine subsequently turns off the chain was modified bit.

If the chain being modified bit is on and an attempt is made to activate or de-activate a program with static storage, a stack control invalid exception is signaled.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
0A Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
2C Program Execution		
03 Stack control invalid		X
05 Activation in use by invocation	X	
32 Scalar Specification		
01 Scalar type invalid	X	

END (END)

Op Code (Hex)

0260

No operands are specified.

Description: The instruction delimits the end of a program's instruction stream. When this instruction is encountered in execution, it causes a return to the preceding invocation (if present) or causes termination of the process phase if the instruction is executed in the highest-level invocation for a process. The End instruction delineates the end of the instruction stream. When it is encountered in execution, the instruction functions as a Return External instruction with a null operand. Refer to the Return External instruction for a description of that instruction.

Events

0000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0202 Process terminated
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference
  0301 Invocation reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception		Other
10	Damage Encountered	
	04 System object damage state	X
	44 Partial system object damage	X
1C	Machine-Dependent Exception	
	03 Machine storage limit exceeded	X
20	Machine Support	
	02 Machine check	X
	03 Function check	X
2A	Program Creation	
	0D Reserved bits are not zero	X

MODIFY AUTOMATIC STORAGE ALLOCATION (MODASA)

Op Code (Hex)	Operand 1	Operand 2
02F2	Storage allocation	Modification size

Operand 1: Space pointer data object or null.

Operand 2: Binary scalar.

Description: The size of automatic storage assigned to the invocation of the currently executing program is extended or truncated by the size specified by operand 2. A positive value indicates that the storage allocation is to be extended; a negative value indicates that the storage allocation is to be truncated. The instruction also returns addressability of the allocated or deallocated storage area in the space pointer identified by operand 1. When allocating additional space, the space pointer locates the first byte of the allocated area. If space is deallocated, the space pointer locates the first byte of the deallocated area. If operand 1 is null, the storage is allocated or deallocated but no addressability is returned. The space pointer identified by operand 1 always addresses storage that is on a 16-byte boundary.

This instruction modifies the next available storage location address value in the machine. Additionally, if the program executing this instruction specified the program requires PASA stack update program attribute, the instruction modifies the next available storage location pointer in the PASA (process automatic storage area) base entry.

The owner's authority to the space is included with the authority of the process when a space is extended and when checked for object management authority.

If the space is extended, the new bytes contain the initial value for the space; otherwise, no initialization is done to the allocated area.

A space extension/truncation exception is signaled if the space containing the PASA cannot be extended. A scalar value invalid exception is signaled if truncation causes the next available storage location pointer in the PASA to point to a location that precedes the beginning of the data of the automatic storage entry for the executing invocation.

The storage allocated with this instruction is not initialized to any value. If implicit space extension occurs, however, the extended portion is initialized to the default value specified for the space when it was created.

A space pointer machine object cannot be specified for operand 1.

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
03 Stack control invalid			X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	
36 Space Management			
01 Space extension/truncation			X

RETURN EXTERNAL (RTX)

Op Code (Hex)	Operand 1
02A1	Return point

Operand 1: Binary (2) scalar or null.

Description: The instruction terminates execution of the invocation in which the instruction is specified. All automatic program objects in the invocation are destroyed by removing the returning program's automatic storage from the PASA (process automatic storage area) by the updating of the PASA chaining pointers.

A Return External instruction can be specified within an invocation's subinvocation, and no exception is signaled.

If a higher invocation exists in the invocation hierarchy, the instruction causes execution to resume in the preceding invocation in the process' invocation hierarchy at an instruction location indirectly specified by operand 1. If operand 1 is binary 0 or null, the next instruction following the Call External instruction from which control was relinquished in the preceding invocation in the hierarchy is given execution control. If the value of operand 1 is not 0, the value represents an index into the IDL (instruction definition list) specified as the return list operand in the Call External instruction, and the value causes control to be passed to the instruction referenced by the corresponding IDL entry. The first IDL entry is referenced by a value of one. If operand 1 is not 0 and no return list was specified in the Call External instruction, or if the value of operand 1 exceeds the number of entries in the IDL, or if the value is negative, a return point invalid exception is signaled.

If the prior invocation is for a program created with the update PASA stack attribute specifying that it requires the PASA stack update, the instruction sets the current invocation entry in the PASA base entry to address the immediately preceding invocation, and it also sets addressability to the returning invocation into the next available storage location entry in the PASA header.

If the prior invocation is for a program created with the update PASA stack attribute specifying that it does not require the PASA stack update, the instruction only updates internal machine values related to the invocation stack.

If a higher invocation does not exist, the Return External instruction causes termination of the current process state. If operand 1 is not 0 and is not null, the return point invalid exception is signaled. Refer to the Terminate Process instruction for the functions performed in process termination.

If the returning invocation has received control to process an event, then control is returned to the point where the event handler was invoked. In this case, if operand 1 is not 0 and is not null, then a return point invalid exception is signaled.

If the returning invocation has received control from the machine to process an exception, the return instruction invalid exception is signaled.

If the returning invocation has an activation, the invocation count in the activation is decremented by 1.

If the returning invocation currently has an invocation exit set, the invocation exit is not given control and is implicitly cleared.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference
    0301 Invocation reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operand	
Exception	**1**	**Other**
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation		X
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded	X	
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	X
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	X
02 Pointer type invalid	X	X
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
2C Program Execution		
01 Return instruction invalid		X
02 Return point invalid	X	

SET ARGUMENT LIST LENGTH (SETALLEN)

Op Code (Hex)	Operand 1	Operand 2
0242	Argument list	Length

Operand 1: Operand list.

Operand 2: Binary scalar.

Description: This instruction specifies the number of arguments to be passed on a succeeding Call External or Transfer Control instruction. The current length of the variable-length operand list (used as an argument list) specified by operand 1 is modified to the value indicated in the binary scalar specified by operand 2. This length value specifies the number of arguments (starting from the first) to be passed from the list when the operand list is referenced on a Call External or Transfer Control instruction.

Only variable-length operand lists with the argument list attribute may be modified by the instruction.

The value in operand 2 may range from 0 (meaning no arguments are to be passed) to the maximum size specified in the ODT definition of the operand list (meaning all defined arguments are to be passed).

The length of the argument list remains in effect for the duration of the current invocation or until a Set Argument List Length instruction is issued against this operand list.

Events

0O0C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands		
Exception	1	2	Other
06 Addressing			
01 Space addressing violation		X	
02 Boundary alignment		X	
03 Range		X	
06 Optimized addressability invalid		X	
08 Argument/Parameter			
01 Parameter reference violation		X	
03 Argument list length modification violation	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded	X		
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found		X	
02 Object destroyed		X	
03 Object suspended		X	
24 Pointer Specification			
01 Pointer does not exist		X	
02 Pointer type invalid		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range		X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
03 Scalar value invalid	X		

SET INVOCATION EXIT (SETIEXIT)

Op Code (Hex)	Operand 1	Operand 2
0252	Invocation exit program	Argument list

Operand 1: System pointer.

Operand 2: Operand list or null.

Description: This instruction allows the external entry point of the program specified by operand 1 to be given control when the requesting invocation is destroyed due to normal exception handling actions, or due to any process termination. Normal exception handling actions are considered to be those actions performed by the Return From Exception or the Signal Exception instructions.

Operand 1 is a system pointer addressing the program that is to receive control. The operand 1 system pointer must be in either the static or automatic storage of the program invoking this instruction.

Operand 2 specifies an operand list that identifies the arguments to be passed to the invocation exit program being called. If operand 2 is null, no arguments are passed to the invocation.

No operand verification takes place when this instruction is invoked. Operand verification occurs when the invocation exit program is invoked. At that time operational authorization to the invocation exit program and retrieve authorization to any contexts referenced for materialization take place. Also, materialization lock enforcement occurs to contexts referenced for materialization.

If an invocation exit program currently exists for the requesting invocation, it is replaced, and no exception is signaled. The invocation exit set by this instruction is implicitly cleared when the invocation exit program is given control, or the program which set the invocation exit completes execution.

If any invocations are to be destroyed due to normal exception handling actions, then those invocation exit programs associated with the invocations to be destroyed are given control before execution proceeds to the signaled exception handler.

An invocation exit bypassed due to a RTNEXCP or a SIGEXCP instruction event is signaled when both of the following conditions occur:

- Exception management is destroying an invocation stack due to a Signal Exception instruction, a Return From Exception instruction, or process termination.

- An invocation exit program is to be destroyed due to a second Signal Exception or a second Return From Exception instruction.

The invocation exit program that is being destroyed is terminated, and its associated invocation execution is terminated. Termination of invocations due to a previous Signal Exception instruction, a Return From Exception instruction, or a process termination is then resumed.

If a process phase is terminated and the process was not in termination phase, then the invocations are terminated. Invocation exit programs set for the terminated invocations are allowed to run. If an invocation to be terminated is an invocation exit program, then the following occurs:

- An invocation exit bypassed due to process termination event is signaled.

- If an invocation exit has been set for this invocation exit, it is allowed to run.

- The invocation exit is terminated and the associated invocation is terminated (the invocation exit is not reinvoked).

Invocation exit programs for the remaining invocations to be terminated are then allowed to run.

For more information about the invocation exit program, see *Program Execution* in the *Functional Concepts Manual.*

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference
    0301 Invocation reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X				
02 Boundary alignment	X				
03 Range	X				
06 Optimized addressability invalid	X				
08 Argument/Parameter					
01 Parameter reference violation	X				
10 Damage Encountered					
04 System object damage state					X
44 Partial system object damage					X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
2A Program Creation					
06 Invalid operand type	X	X			
07 Invalid operand attribute	X				
08 Invalid operand value range	X				
0C Invalid operand ODT reference	X	X			
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
01 Scalar type invalid	X				

STORE PARAMETER LIST LENGTH (STPLLEN)

Op Code (Hex)	**Operand 1**
0241	Length

Operand 1: Binary variable scalar.

Description: A value is returned in operand 1 that represents the number of parameters associated with the invocation's external entry point for which arguments have been passed on the preceding Call External or Transfer Control instruction.

The value can range from 0 (no parameters were received) to the maximum size possible for the parameter list associated with the external entry point.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded	X	
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	
02 Scalar attributes invalid	X	

TRANSFER CONTROL (XCTL)

Op Code (Hex)	Operand 1	Operand 2
0282	Program	Argument list

Operand 1: System pointer.

Operand 2: Operand list or null.

Description: The instruction destroys the calling invocation and causes control to be passed to the external entry point of the program specified by operand 1. Operand 1 is a system pointer addressing the program that is to receive control.

The invocation count in the activation (if any) of the calling program is decremented by 1. The instruction ensures that the called program is properly activated in the process, if required. See the Activate Program instruction for a definition of this activation verification process.

After any needed static storage has been allocated or located, the invocation entry to the program issuing the Transfer Control instruction is made available for the new invocation. Unless precluded by internal machine alignment requirements, the new invocation's stack control entry and automatic storage overlay that of the invocation issuing the Transfer Control instruction. The new invocation entry is allocated beginning at the same location as that of the current (transferring) invocation. See the Call External instruction for a definition of a PASA (process automatic storage area) entry.

The new invocation's stack control entry is initialized as follows:

- The previous invocation entry pointer and the next invocation entry pointer are the same as that of the invoking program's entry.
- The associated program pointer is copied from the associated activation entry (or from the operand 1 system pointer if no activation entry exists).
- The invocation number entry is unchanged.
- The invocation type value is set to indicate that the program was invoked via a Transfer Control instruction (hex 20).
- The program's automatic storage is allocated and initialized as specified in the program definition.

The invocation entry for the preceding invocation is unchanged by the instruction. The current invocation entry pointer in the PASA base entry is unchanged by the instruction. The next available storage location entry in the PASA base entry is set to address the next available 16-byte aligned area beyond the new invocation entry.

The program is invoked by allocating an area in the PASA space that is sufficient to contain the invocation entry. The area used for allocation is located by the next available storage location pointer in the PASA base entry. This pointer must address a 16-byte aligned area in the space, or a boundary alignment exception is signaled.

The maximum addressable location in the PASA space limits the amount of storage that may be allocated for PASA storage. If this limit is exceeded, the process storage limit exceeded exception is signaled. If the maximum addressable location entry does not address the same space as that addressed by the next available storage location entry, the stack control invalid exception is signaled.

If insufficient space is available in the PASA for the entire new entry, the PASA space is implicitly extended by the machine. If the space is fixed size or may not be extended enough to contain the entry, a space extension/truncation exception is signaled.

Following the allocation and initialization of automatic storage, control is passed to the invoked program.

Operand 2 specifies an operand list that identifies the arguments to be passed to the invocation to which control is being transferred. Automatic objects allocated by the transferring invocation are destroyed as a result of the transfer operation and, therefore, cannot be passed as arguments. A parameter list length exception is signaled if the number of arguments passed does not correspond to the number required by the parameter list of the target program.

If the transferring invocation has received control to process an exception, an event, or an invocation exit, the return instruction invalid exception is signaled.

If the transferring invocation currently has an invocation exit set, the invocation exit is not given control and is implicitly cleared.

Authorization Required

- Operand 1
  - Operational
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X		
02 Boundary alignment	X		
03 Range	X		
06 Optimized addressability invalid	X		
08 Argument/Parameter			
01 Parameter reference violation			X
02 Parameter list length violation		X	
0A Authorization			
01 Unauthorized for operation	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
02 Program limitation exceeded			X
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X		
02 Object destroyed	X		
03 Object suspended	X		
24 Pointer Specification			
01 Pointer does not exist	X		
02 Pointer type invalid	X		
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
01 Return instruction invalid			X
03 Stack control invalid			X
36 Space Management			
01 Space extension/truncation			X

Chapter 10. Exception Management Instructions

This chapter describes all instructions used for exception management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

MATERIALIZE EXCEPTION DESCRIPTION (MATEXCPD)

Op Code (Hex)	**Operand 1**	**Operand 2**	**Operand 3**
03D7	Attribute receiver	Exception description	Materialization option

Operand 1: Space pointer.

Operand 2: Exception description.

Operand 3: Character(1) scalar.

Description: The instruction materializes the attributes (operand 3) of an exception description (operand 2) into the receiver specified by operand 1.

The template identified by operand 1 must be a 16-byte aligned area in the space if the materialization option is hex 00.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver operand contains insufficient area for the materialization.

Operand 2 identifies the exception description to be materialized.

The value of operand 3 specifies the materialization option. If the materialization option is hex 00, the format of the exception description materialization is as follows:

• Template size	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Control flags	Char(2)
– Exception handling action	Bits 0-2
000 = Do not handle. (Ignore occurrence of exception and continue processing.)	
001 = Do not handle. (Disable this exception description and continue to search this invocation for another exception description to handle the exception.)	
010 = Do not handle. (Continue to search for an exception description by resignaling the exception to the preceding invocation.)	
100 = Defer handling. (Save exception data for later exception handling.)	
101 = Pass control to the specified exception handler.	
– No data	Bit 3
0 = Exception data is returned	
1 = Exception data is not returned	
– Reserved (binary 0)	Bit 4
– User data indicator	Bit 5
0 = User data not present	
1 = User data present	
– Reserved (binary 0)	Bits 6-7
– Exception handler type	Bits 8-9
00 = External entry point	
01 = Internal entry point	
10 = Branch point	
– Reserved (binary 0)	Bits 10-15
• Instruction number to be given control (if internal entry point or branch point; otherwise, 0)	Bin(2)
• Length of compare value (maximum of 32 bytes)	Bin(2)
• Compare value (size established by value of length of compare value parameter)	Char(32)
• Number of exception IDs	Bin(2)
• System pointer to the exception handling program if an external exception handler is specified	System pointer
• Pointer to user data (not present if value of user data indicator is binary 0)	Space pointer
• Exception ID (one for each exception ID dictated by the number of exception IDs attribute)	Char(2)

If the materialization option is hex 01, the format of the materialization is as follows:

• Template size		Char(8)
– Number of bytes provided for materialization		Bin(4)
– Number of bytes available for materialization		Bin(4)
• Control flags		Char(2)
– Exception handling action		Bits 0-2
000 =	Do not handle. (Ignore occurrence of exception and continue processing.)	
001 =	Do not handle. (Disable this exception description and continue to search this invocation for another exception description to handle the exception.)	
010 =	Do not handle. (Continue to search for an exception description by resignaling the exception to the preceding invocation.)	
100 =	Defer handling. (Save exception data for later exception handling.)	
101 =	Pass control to the specified exception handler.	
– No data		Bit 3
0 =	Exception data is returned	
1 =	Exception data is not returned	
– Reserved (binary 0)		Bit 4
– User data indicator		Bit 5
0 =	User data not present	
1 =	User data present	
– Reserved (binary 0)		Bits 6-15

If the materialization option is hex 02, the format of the materialization is as follows:

• Template size	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Compare value length (maximum of 32 bytes)	Bin(2)
• Compare value	Char(32)

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	 2	 3	Other
06 Addressing				
01 Space addressing violation	X		X	
02 Boundary alignment	X		X	
03 Range	X		X	
06 Optimized addressability invalid	X		X	
08 Argument/Parameter				
01 Parameter reference violation	X		X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X		X	
02 Object destroyed	X		X	
03 Object suspended	X		X	
24 Pointer Specification				
01 Pointer does not exist	X		X	
02 Pointer type invalid	X		X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X		X	
08 Invalid operand value range	X		X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MODIFY EXCEPTION DESCRIPTION (MODEXCPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
03EF	Exception description	Modifying attributes	Modification option

Operand 1: Exception description.

Operand 2: Space pointer, or character(2) constant.

Operand 3: Character(1) scalar.

Description: The exception description attributes specified by operand 3 are modified with the values of operand 2.

Operand 1 references the exception description.

Operand 2 specifies the new attribute values. Operand 2 may be either a character constant or a space pointer to the modification template. When operand 3 is a constant, operand 2 is a character constant; when operand 3 is not a constant, operand 2 is a space pointer.

The value of operand 3 specifies the modification option. If the modification option is hex 01 and operand 2 specifies a space pointer, the format of the modifying attributes pointed to by operand 2 is as follows:

• Template size	Char(8)
– Number of bytes provided for materialization (must be at least 10)	Bin(4)
– Number of bytes available for materialization	Bin(4)*
• Control flags	Char(2)
– Exception handling action	Bits 0-2
000 = Do not handle. (Ignore occurrence of exception and continue processing.)	
001 = Do not handle. (Disable this exception description and continue to search this invocation for another exception description to handle the exception.)	
010 = Do not handle. (Continue to search for an exception description by resignaling the exception to the preceding invocation.)	
100 = Defer handling. (Save exception data for later exception handling.)	
101 = Pass control to the specified exception handler.	
– No data	Bit 3
0 = Exception data is returned.	
1 = Exception data is not returned.	
– Reserved (binary 0)	Bits 4-15

If the exception description was in the deferred state prior to the modification, the deferred signal, if present, is lost.

When the option to not return exception data is selected, no data is returned for the Retrieve Exception Data or Test Exception instructions, and the number of bytes available for the materialization field is set to 0. This option can also be selected in the ODT definition of the exception description.

If the modification option of operand 3 is a constant value of hex 01, then operand 2 may specify a character constant. The operand 2 constant has the same format as the control flags entry previously described.

If the modification option is hex 02, then operand 2 must specify a space pointer. The format of the modification is as follows:

• Template size	Char(8)
– Number of bytes provided (must be at least 10 plus the length of the compare value in the exception description)	Bin(4)
– Number of bytes available for materialization	Bin(4)*
• Compare value length (maximum of 32 bytes)	Bin(2)*
• Compare value	Char(32)

Note: Entries shown here with an asterisk (*) are ignored by the instruction.

The number of bytes in the compare value is dictated by the compare value length specified in the exception description as originally specified in the object definition table.

An external exception handling program can be modified by resolving addressability to a new program into the system pointer designated for the exception description.

The presence of user data is not a modifiable attribute of exception descriptions. If the exception description has user data, it can be modified by changing the value of the data object specified in the exception description.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation		X	X	
03 Range		X	X	
06 Optimized addressability invalid		X	X	
08 Argument/Parameter				
01 Parameter reference violation		X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found		X	X	
02 Object destroyed		X	X	
03 Object suspended		X	X	
24 Pointer Specification				
01 Pointer does not exist		X	X	
02 Pointer type invalid		X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute		X	X	
08 Invalid operand value range		X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
01 Template value invalid		X		
02 Template size invalid		X		

RETRIEVE EXCEPTION DATA (RETEXCPD)

Op Code (Hex)	Operand 1	Operand 2
03E2	Receiver	Retrieve options

Operand 1: Space pointer.

Operand 2: Character(1) scalar (fixed-length).

Description: The data related to a particular occurrence of an exception is returned and placed in the specified space.

Operand 1 is a space pointer that identifies the receiver template. The template identified by operand 1 must be 16-byte aligned in the space.

The value of operand 2 specifies the type of exception handler for which the exception data is to be retrieved. The exception handler may be a branch point exception handler, an internal entry point exception handler, or an external entry point exception handler.

An exception state of process invalid exception is signaled to the invocation issuing the Retrieve Exception Data instruction if the retrieve option is not consistent with the process's exception handling state. For example, the exception is signaled if the retrieve option specifies retrieve for internal entry point exception handler and the process exception state indicates that an internal exception handler has not been invoked.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

After an invocation has been destroyed, exception data associated with a signaled exception description within that invocation is lost.

The format of operand 1 for the materialization is as follows:

• Template size	Char(8)
– Number of bytes provided for retrieval	Bin(4)
– Number of bytes available for retrieval	Bin(4)
• Exception identification	Char(2)
• Compare value length (maximum of 32 bytes)	Bin(2)
• Compare value	Char(32)
• Reserved (binary 0)	Char(4)
• Exception specific data	Char(*)
• Signaling program invocation	Space pointer
• Signaled program invocation	Space pointer
• Signaling program instruction address	Bin(2)
• Signaled program instruction address	Bin(2)
• Machine-dependent data	Char(10)

The signaling program invocation address entry locates the invocation entry in the PASA (process automatic storage area) that corresponds to the invocation that caused the exception to be signaled. For machine exceptions, this space pointer locates the invocation executing when the exception occurred. For user-signaled exceptions, this space pointer locates the invocation that executed the Signal Exception instruction. The signaling program instruction address entry locates the instruction that caused the exception to be signaled.

The signaled program invocation entry locates the invocation entry in the PASA that is signaled to handle the exception. This invocation is the last invocation signaled or resignaled to handle the exception. For machine exceptions, the first invocation signaled is the invocation incurring the exception. For user-signaled exceptions, the Signal Exception instruction may initially locate the current or any previous invocation. If the invocation to be signaled handles the exception by resignaling the exception, the immediately previous invocation is considered to be the last signaled invocation. This may occur repetitively until no more prior invocations exist in the process and the signaled program invocation entry is assigned a value of binary 0. If an invocation to be signaled handles the exception in any manner other than resignaling or does not handle the exception, that invocation is considered to be the last signaled.

The signaled program instruction address entry specifies the number of the instruction that is currently being executed in the signaled invocation.

The machine extends the area beyond the exception specific data area with binary 0's so that the pointers to program invocations are properly aligned.

The operand 2 values are defined as follows:

- Retrieve options Char(1)
  - Hex 00=Retrieve for a branch point exception handler
  - Hex 01=Retrieve for an internal entry point exception handler
  - Hex 02=Retrieve for an external entry point exception handler

If the exception data retention option is set to 1 (do not save), the number of bytes available for retrieval is set to 0.

Exception data is always available to the process default exception handler.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
16 Exception Management			
02 Exception state of process invalid		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
03 Scalar value invalid		X	
38 Template Specification			
03 Materialization length exception	X		

RETURN FROM EXCEPTION (RTNEXCP)

Op Code (Hex)	Operand 1
03E1	Return target

Operand 1: Space pointer.

Description: An internal exception handler subinvocation or an external exception handler invocation is terminated, and control is passed to the specified instruction in the specified invocation.

The template identified by operand 1 must be 16-byte aligned in the space. It specifies the target invocation and target instruction in the invocation where control is to be passed. The format of operand 1 is as follows:

- Invocation address — Space pointer
- Reserved (binary 0) — Char(1)
- Action — Char(1)
  - Reserved (binary 0) — Bits 0-6
  - Action Code — Bit 7
    0 = Re-execute the instruction that caused the exception or the instruction that invoked the invocation.
    1 = Resume execution with the instruction that follows the instruction that caused the exception or resume execution with the instruction that follows the instruction that invoked the invocation.
  - Reserved (binary 0) — Char(1)

The invocation address entry is a space pointer that locates an invocation entry in the PASA (process automatic storage area) chain to which control will be passed. The current instruction in an invocation is the one that caused another invocation to be created. If an event handler was invoked, then the current instruction is the instruction that executed prior to the invocation of the event handler.

If the action code is 0, then the current instruction of the addressed invocation is reexecuted. If the action code is 1, execution resumes with the instruction following the current instruction of the addressed invocation.

When a Return From Exception instruction returns control to an invocation that was interrupted by an event, the action code in the operand 1 template is ignored and execution continues at the point of interruption. That is, the interrupted instruction is not reexecuted and execution of the instruction is completed as if no interruption occurred. For example, if a Dequeue instruction is waiting for a message to arrive on a queue when an event handler is invoked that produces an exception, the exception handler returns control to the interrupted Dequeue instruction and the instruction continues to wait for the message.

The Return From Exception instruction may be issued only from the initial invocation of an external exception handling sequence or from an invocation that has an active internal exception handler.

If the instruction is issued from an invocation that is not an external exception handler and has no internal exception handler subinvocations, the return instruction invalid exception is signaled.

The following table shows the actions performed by the Return From Exception instruction:

Invocation Issuing Instruction	Addressing Own Invocation/Option		Addressing Higher Invocation/Option	
Not handling exception	Error	1	Error	1
Handling internal exception(s)	Allowed	2	Allowed	3
Handling external exception(s)	Error	1	Allowed	3
Handling external exception(s) and internal exception(s)	Allowed	2	Allowed	3

1. A return instruction invalid exception is signaled. If there are no more internal exception handler subinvocations active and this invocation is not an external exception handler, the instruction may not be issued.

2. The current internal exception handler subinvocation is terminated.

3. All invocations after the addressed invocation are terminated and execution proceeds within the addressed invocation. Any invocation exit programs set for the terminated invocations will be given control before execution proceeds within the addressed invocation.

Whenever an invocation is terminated, the invocation count in the corresponding activation entry (if any) is decremented by 1.

An action code of 1 specifies completion of an instruction rather than execution of the following instruction if the current instruction in the addressed invocation signaled a size exception or a floating-point inexact result exception.

Note: The previous condition does not apply if any of the above exceptions were explicitly signaled by a Signal Exception instruction.

A Return From Exception instruction cannot be used or recognized in conjunction with a branch point internal exception handler.

If a failure to invoke an invocation exit handler occurs, a failure to invoke program event is signaled.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0011 Program
    0301 Invocation exit bypassed due to a RTNEXCP or a SIGEXCP instruction
    0401 Failure to invoke program

0016 Machine observation
    0101 Instruction reference
    0301 Invocation reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
01 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
16 Exception Management		
03 Invalid invocation	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
09 Invalid branch target operand		X
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
2C Program Execution		
01 Return instruction invalid		X
38 Template Specification		
01 Template value invalid	X	

SENSE EXCEPTION DESCRIPTION (SNSEXCPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
03E3	Attribute receiver	Invocation template	Exception template

Operand 1: Space pointer.

Operand 2: Space pointer.

Operand 3: Space pointer.

Description: The Sense Exception Description instruction searches the invocation specified by operand 2 for an exception description that matches the exception identifier and compare value specified by operand 3 and returns the user data and exception handling action specified in the exception description. The exception descriptions of the invocation are searched in ascending ODT number sequence.

The exception identifier in the exception description can be specified in one of the following ways:

Hex 0000	=	Any exception ID will result in a match
Hex nn00	=	Any exception ID in class nn will result in a match
Hex nnmm	=	Only exception ID nnmm will result in a match

If a match on exception ID is detected, the corresponding compare values are matched. If the compare value length in the exception description is less than the compare value in the search template, the length of the compare value in the exception description is used for the match. If the compare value length in the exception description is greater than the compare value in the search template, an automatic mismatch results.

If a match on exception ID and compare value is detected, the exception handling action of the exception description determines which of the following actions is taken:

IGNORE The operand 1 template is materialized.

DISABLE The exception description is bypassed and the search for an exception description continues with the next exception description defined for the invocation.

RESIGNAL The operand 1 template is materialized.

DEFER The operand 1 template is materialized.

HANDLE The operand 1 template is materialized.

If no exception description of the invocation matches the exception ID and compare value of operand 3, the number of bytes available for materialization on the operand 1 template is set to 0.

The template identified by operand 1 must be 16-byte aligned.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exception is signaled in the event the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The format of the attribute receiver is as follows:

• Template size	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Control flags	Char(2)
– Exception handling action	Bits 0-2
000 = Do not handle–ignore occurrence of exception and continue processing	
010 = Do not handle–continue search for an exception description by resignaling the exception to the immediately preceding invocation	
100 = Defer handling–save exception data for later exception handling	
101 = Pass control to the specified exception handler	
– No data	Bit 3
0 = Exception data is returned	
1 = Exception data is not returned	
– Reserved (binary 0)	Bit 4
– User data indicator	Bit 5
0 = User data not present	
1 = User data present	
– Reserved (binary 0)	Bits 6-7
– Exception handler type	Bits 8-9
00 = External entry point	
01 = Internal entry point	
10 = Branch point	
– Reserved (binary 0)	Bits 10-15
• Relative exception description number	Bin(2)
• Reserved (binary 0)	Char(4)
• Pointer to user data (binary 0 if value of user data indicator is binary 0)	Space pointer

The relative exception description number entry identifies the relative number of the exception description that matched the search criteria. The order of definition of the exception descriptions in the ODT determines the value of the index. A value of 1 indicates that the first exception description defined in the ODT matched the search criteria.

The template identified by operand 1 must be 16-byte aligned. The invocation address entry is a space pointer that locates an invocation entry in the PASA (process automatic storage area). The invocation is searched for a matching exception description. If the space pointer locates the PASA base entry, the operand 1 template is materialized with the number of bytes available for materialization set to 0. If the space pointer locates neither a valid invocation entry nor the PASA base entry, the invalid invocation address exception is signaled.

The first exception description to search entry specifies the relative number of the exception description to be used to start the search. The number must be a nonzero positive binary number determined by the order of definition of exception descriptions in the ODT. A value of 1 indicates that the first exception description in the invocation is to be used to begin the search. If the value is greater than the number of exception descriptions for the invocation, the operand 1 template is materialized with the number of bytes available for materialization set to 0.

The format of the invocation template is as follows:

• Invocation address	Space pointer
• Reserved (binary 0)	Char(2)
• First exception description to search	Bin(2)

The operand 3 exception template specifies the exception-related data to be used as a search argument. The format of the template is as follows:

• Template size	Char(8)
– Number of bytes provided for materialization (must be at least 44)	Bin(4)
– Number of bytes available for materialization	Bin(4)*
• Exception identifier	Char(2)
• Compare value length (maximum of 32)	Bin(2)
• Compare value	Char(32)

Note: Entries noted with an asterisk (*) are ignored by the instruction.

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check
    0801 Process storage limit exceeded

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage				X
44 Partial system object damage				X
16 Exception Management				
03 Invalid invocation address		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X	X	
02 Template size invalid			X	
03 Materialization length exception	X			

SIGNAL EXCEPTION (SIGEXCP)

Op Code (Hex)	Operand 1	Operand 2
10CA	Attribute template	Exception data

Operand 1: Space pointer.

Operand 2: Space pointer.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SIGEXCPI	18CA	Indicator
SIGEXCPB	1CCA	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Signal Exception instruction signals a new exception or resignals an existing exception to the process. Optionally, the instruction branches to one of the specified targets based on the results of the signal and the selected branch options in the extender field, or it sets indicators based on the results of the signal. The signal is presented starting at the invocation identified in the signal template.

The template identified by operand 1 specifies the signal option and starting point. It must be 16-byte aligned in the space with the following format.

• Signaled to invocation address	Space pointer
• Signal option	Char(1)
– Signal/resignal option 0 = Signal new exception. 1 = Resignal currently handled exception (valid only for an external exception handler).	Bit 0
– Invoke PDEH (process default exception handler) option 0 = Invoke PDEH if no exception description found for invocation. 1 = Do not invoke PDEH if no exception description found for invocation (ignore if PASA base entry specified).	Bit 1
– Exception description search control 0 = Exception description search control not present 1 = Exception description present	Bit 2
– Reserved (binary 0)	Bits 3-7
• Reserved (binary 0)	Char(1)
• First exception description to search	Bin(2)

The signaled to invocation address entry is a space pointer that locates an invocation entry in the PASA (process automatic storage area). The exception is signaled to this invocation. If the space pointer locates the PASA base entry, the exception is signaled to the PDEH. If the space pointer locates neither a valid invocation entry nor the PASA base entry, the invalid invocation address exception is signaled. If the program associated with the invocation has defined an exception description to handle the exception, the specified action is taken; otherwise, the PDEH is invoked unless the invoke PDEH option bit is 1 (the exception is considered ignored). If the PASA base entry is addressed instead of an existing invocation, the PDEH will be invoked.

Exception descriptions of an invocation are searched in ascending ODT number sequence. If the exception description search control is not present, the search begins with the first exception description defined in the ODT. Otherwise, the first exception description to search value identifies the relative number of the exception description to be used to start the search. The value must be a nonzero positive binary number determined by the order of definition of exception descriptions in the ODT. This value is also returned by the Sense Exception Description instruction. A value of 1 indicates that the first exception description in the invocation is to be used to begin the search. If the value is greater than the number of exception descriptions for the invocation, the template value invalid exception is signaled.

The template identified by operand 2 must be 16-byte aligned in the space. It specifies the exception-related data to be passed with the exception signal. The format of the exception data is the same as that returned by the Retrieve Exception Data instruction. The format is as follows:

• Template size	Char(8)
– Number of bytes of data to be signaled (must be at least 48 bytes)	Bin(4)
– Number of bytes available for materialization	Bin(4)*
• Exception identification	Char(2)
• Compare value length (maximum of 32 bytes)	Bin(2)
• Compare value	Char(32)
• Reserved (binary 0)	Char(4)
• Exception specific data	Char(*)

Note: Entries shown here with an asterisk (*) are ignored by the instruction.

Operand 2 is ignored if operand 1 specifies the resignal option, because the exception-related data is the same as for the exception currently being processed; however, it must be specified when signaling a new exception.

The maximum size for exception-related data that is to accompany an exception signaled by the Signal Exception instruction is 32 608 bytes, including the standard signal data.

If an exception ID in an exception description corresponds to the signaled exception, the corresponding compare values are verified. If the compare value length in the exception description is less than the compare value length in the signal template, the length of the compare value in the exception description is used for the match. If the compare value length in the exception description is greater than the compare value length in the signal template, an automatic mismatch results. Machine-signaled exceptions have a 4-byte compare value of binary 0's.

An exception description may monitor for an exception with a generic ID as follows:

Hex 0000 = Any signaled exception ID results in a match.

Hex nn00 = Any signaled exception ID in class nn results in a match.

Hex nnmm = The signaled exception ID must be exactly nnmm in order for a match to occur.

An exception description may be in one of five states, each of which determines an action to be taken when the match criteria on the exception ID and compare value are met.

IGNORE No exception handling occurs. The Signal Exception instruction is assigned a resultant condition of ignored. If a corresponding branch or indicator setting is present, that action takes place.

DISABLE The exception description is bypassed, and the search for a monitor continues with the next exception description defined for the invocation.

RESIGNAL The search for a monitoring exception description is to be reinitiated at the preceding invocation. A resignal from the initial invocation in the process results in the invocation of the process default exception handler. A resignal from an invocation exit program results in an unhandled exception that causes process termination.

DEFER The exception description is signaled, and the Signal Exception instruction is assigned the resultant condition of deferred. If a corresponding branch or indicator setting is present, that action takes place. To take future action on a deferred exception, the exception description must be synchronously tested with the Test Exception instruction in the signaled invocation.

HANDLE Control is passed to the indicated exception handler, which may be a branch point, an internal subinvocation, or an external invocation.

If the exception description is in the ignore or defer state and if the Signal Exception instruction does not specify a branch or indicator condition or if it specifies branch or indicator conditions that are not met, then the instruction following the Signal Exception instruction is executed.

When control is given to an internal or branch point exception handler, all invocations up to, but not including, the exception handling invocation are terminated. Any invocation exit programs set for the terminated invocations will be given control before execution proceeds in the signaled exception handler. For more information about the invocation exit program, see *Program Execution* in the *Functional Concepts Manual.*

If a failure to invoke an external exception handler or an invocation exit occurs, a failure to invoke program event is signaled. For each destroyed invocation, the invocation count in the corresponding activation entry (if any) is decremented by 1.

Resultant Conditions: Exception ignored or exception deferred.

Events

0002 Authorization
- 0101 Object authorization violation

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0011 Program
- 0301 Invocation exit bypassed due to a RTNEXCP or a SIGEXCP instruction
- 0401 Failure to invoke program

0016 Machine observation
- 0101 Instruction reference
- 0301 Invocation reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	**Operands**		**Other**
	1	**2**	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
16 Exception Management			
02 Exception state of process invalid			X
03 Invalid invocation	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
05 Invalid op code extender field			X
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
09 Invalid branch target operand			X
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
01 Template value invalid	X		
02 Template size invalid	X		

TEST EXCEPTION (TESTEXCP)

Op Code (Hex)	Operand 1	Operand 2
104A	Receiver	Exception description

Operand 1: Space pointer.

Operand 2: Exception description.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
TESTEXCPI	184A	Indicator
TESTEXCPB	1C4A	Branch

Extender: Branch options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator targets (for indicator options). The branch or indicator targets immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The instruction tests the signaled status of the exception description specified in operand 2, and optionally alters the control flow or sets the specified indicators based on the test. Exception data is returned at the location identified by operand 1. The branch or indicator setting occurs based on the conditions specified in the extender field depending on whether or not the specified exception description is signaled.

Operand 2 is an exception description whose signaled status is to be tested. An exception can be signaled only if the referenced exception description is in the deferred state.

Operand 1 addresses a space into which the exception data is placed if an exception identified by the exception description has been signaled.

The template identified by operand 1 must be 16-byte aligned in the space.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

If the exception description is not in the signaled state, the number of bytes available for the materialization entry is set to binary 0's, and no other bytes are modified. The format of the data returned in operand 1 is as follows:

• Template size	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization (0 if exception description is not signaled)	Bin(4)
• Exception identification	Char(2)
• Compare value length (maximum of 32 bytes)	Bin(2)
• Compare value	Char(32)
• Reserved (binary 0)	Char(4)
• Exception-specific data	Char(*)
• Signaling program invocation address	Space pointer
• Signaled program invocation address	Space pointer
• Signaling program instruction address	Bin(2)
• Signaled program instruction address	Bin(2)
• Machine-dependent data	Char(10)

The area beyond the exception-specific data area is extended with binary 0's so that pointers to program invocations are properly aligned.

If no branch options are specified, instruction execution proceeds at the instruction following the Test Exception instruction.

If the exception data retention option is set to 1 (do not save), no data is returned by this instruction.

Resultant Conditions: Exception signaled or exception not signaled.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X		
02 Boundary alignment	X		
03 Range	X		
06 Optimized addressability invalid	X		
08 Argument/Parameter			
01 Parameter reference violation	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
16 Exception Management			
01 Exception description status invalid		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
24 Pointer Specification			
01 Pointer does not exist	X		
02 Pointer type invalid	X		
2A Program Creation			
05 Invalid op code extender field			X
06 Invalid operand type	X	X	
09 Invalid branch target operand			X
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
03 Materialization length exception	X		

Chapter 11. Process Management Instructions

This chapter describes instructions used for process management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CREATE PROCESS CONTROL SPACE (CRTPCS)

Op Code (Hex)	Operand 1	Operand 2
0322	Process control space	Creation template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: A process control space is created. That space has the attributes contained in the creation template specified by operand 2. Addressability to the created process control space is placed in a system pointer that is specified by operand 1.

A process control space is required as a machine work area for an initiated process. A system pointer addressing the process control space of an initiated process is used to identify the process.

The size of the process control space is managed by the machine and is not specified by the user.

The template identified by operand 2 must be 16-byte aligned within the control space. Following is the format of the space creation template:

• Template size specification	Char(8)*
– Size of template	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attribute 0 = Temporary (required)	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context 0 = Addressability is not inserted into the context 1 = Addressability is inserted into the context	Bit 2
– Access group 0 = Not created as member of access group 1 = Created as member of access group	Bit 3
– Reserved (binary 0)	Bits 4-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Access group	System pointer

Note: The values associated with template entries annotated with an asterisk (*) are ignored by the instruction.

The created process control space is temporary and has no owning user profile. All authority states for the object are considered to be public. The storage occupied by the created process control space is charged to the creating process.

The object identification specifies the symbolic name that identifies the process control space within the machine. A type code of hex 1A is implicitly supplied by the machine. The object identification identifies the process control space on materialize instructions and locates the process control space in a context that addresses the process control space.

The existence attribute specifies that the process control space is to be created as temporary. A process control space, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated.

A space may be associated with the created process control space. The length of the space may be fixed or variable. The initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated is dependent on an algorithm defined by a specific implementation. A fixed size space of zero length causes no space to be allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, the initial value of space byte is also used to initialize the new allocation.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created process control space is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

If the access group creation attribute entry indicates that the process control space is to be created in an access group, the access group entry must be a system pointer that identifies the access group in which the process control space is to be created. If the process control space is not to be created in an access group, the access group entry is ignored.

The performance class parameter provides information that allows the machine to manage the process control space with consideration for the overall performance objectives of operations involving the space.

Authorization

- Insert
  - Context identified in operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Context identified in operand 2
  - Access group identified in operand 2

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded
0501 Machine address threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
02 Access Group					
01 Object ineligible for access group		X			
06 Addressing					
01 Space addressing violation	X	X			
02 Boundary alignment	X	X			
03 Range	X	X			
06 Optimized addressability invalid	X	X			
08 Argument/Parameter					
01 Parameter reference violation	X	X			
0A Authorization					
01 Unauthorized for operation		X			
0E Context Operation					
01 Duplicate object identification		X			
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state		X			
1C Machine-Dependent Exception					
01 Machine dependent request invalid					X
03 Machine storage limit exceeded					X
04 Object storage limit exceeded	X				
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X			
02 Object destroyed	X	X			
03 Object suspended	X	X			
24 Pointer Specification					
01 Pointer does not exist		X			
02 Pointer type invalid		X			
03 Pointer addressing invalid object		X			
2A Program Creation					
06 Invalid operand type	X	X			
07 Invalid operand attribute	X	X			
08 Invalid operand value range	X	X			
0A Invalid operand length		X			
0C Invalid operand ODT reference	X	X			
0D Reserved bits are not zero	X	X	X	X	X
2E Resource Control Limit					
01 User profile storage limit exceeded					X
38 Template Specification					
01 Template value invalid		X			

DESTROY PROCESS CONTROL SPACE (DESPCS)

Op Code (Hex)	**Operand 1**
0311	Process control space to be destroyed

Operand 1: System pointer.

Description: The designated process control space is destroyed and addressability to the space is deleted from a context if a context is currently addressing the object. The system pointer identified by operand 1 is not modified by the instruction, and a subsequent reference to the destroyed process control space through the pointer results in an object destroyed exception.

If the process control space is currently being used by a process, an object not eligible for destruction exception is signaled.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Modification
  - Context addressing object
  - Access group containing object

- Object control
  - Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

INITIATE PROCESS (INITPR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0324	Process control space	Process definition template	Argument list	Lock list

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: Argument list or null.

Operand 4: Space pointer or null.

Description: A process is established in the machine.

The process control space identified by operand 1 identifies a process to be established.

The process definition template specified by operand 2 defines the attributes of the process.

Operand 3 specifies an argument list to be presented to the first program executed in the process problem phase. When the operand is null, no arguments are presented.

Operand 4 locates an area in a space that identifies object locks (that are to be transferred to the process being established) currently held by the process issuing the Initiate Process instruction. When the operand is null, no locks are transferred.

When a new process is being established, the process control space provided by operand 1 must not be associated or used by any other active or suspended process. If a process is already associated with the process control space, the object not available to process exception is signaled. Privileged instruction authorization is required to establish a new process. The number of initiated processes is dependent on the main storage size and other current demands on main storage. Each initiated process requires a minimum of 1024 bytes of main storage.

Because this instruction requires one process to act upon another process, a portion of the function is controlled by the issuing process, and the remainder of the function is controlled by the new process. When control is returned to the issuing process, the function may not have been performed in its entirety. An event is signaled when the process initiation is complete (either successfully or unsuccessfully). The process terminated event is signaled when the initiation of a process is incomplete. An exception that indicates the reason for the failure of the Initiate Process instruction is signaled if the exception is detected prior to the new process becoming a dispatchable entity in the machine.

The process definition template specified by operand 2 establishes the attributes of the process being established. The template identified by operand 2 must be 16-byte aligned in a space.

The format of the process definition template is as follows:

- Size of process definition template — Char(8)*
  - Number of bytes provided — Bin(4)*
  - Number of bytes available for materialization — Bin(4)*

- Process control attributes — Char(4)
  - Process type — Bit 0
    - 0 = Dependent process
    - 1 = Independent process
  - Instruction wait access state control — Bit 1
    - 0 = Access state modification is not allowed
    - 1 = Access state modification is allowed if specified
  - Time slice end access state control — Bit 2
    - 0 = Access state modification is not allowed
    - 1 = Access state modification is allowed if specified
  - Time slice event option — Bit 3
    - 0 = Time slice expired without entering instruction wait event is not signaled during time slice
    - 1 = Time slice expired without entering instruction wait event is signaled
  - Reserved (binary 0) — Bit 4
  - Initiation phase program option — Bit 5
    - 0 = No initiation phase program specified (do not enter initiation phase)
    - 1 = Initiation phase program specified (enter initiation phase)
  - Problem phase program option — Bit 6
    - 0 = No problem phase program specified (do not enter problem phase)
    - 1 = Problem phase program specified (enter problem phase)
  - Termination phase program option — Bit 7
    - 0 = No termination phase program specified (do not enter termination state)
    - 1 = Termination phase program specified (enter termination state)
  - Process default exception handler option — Bit 8
    - 0 = No process default exception handler
    - 1 = Process default exception handler specified
  - Process name resolution list option — Bit 9
    - 0 = No process name resolution list
    - 1 = Process name resolution list specified
  - Process access group option — Bit 10
    - 0 = No process access group option
    - 1 = Process access group specified
  - Process adopted user profile list option — Bit 11
    - 0 = No process adopted user profile list
    - 1 = Process adopted user profile list specified
  - Reserved (binary 0) — Bits 12-31

- Signal event control mask — Char(2)

- Number of event monitors (0-256) — Bin(2)

• Resource management attributes	
– Process priority	Char(1)
– Process storage pool identification	Char(1)
– Maximum temporary auxiliary storage allowed (in bytes)	Bin(4)
– Time slice interval	Char(8)
– Default time-out interval	Char(8)
– Maximum processor time allowed	Char(8)
– Process multiprogramming level class ID	Char(1)

• Modification control indicators	Char(8)

Each indicator specifies the modification options for a specific attribute of the process being controlled by the process definition template that the modification control indicators are part of. The values and bit assignments are as follows:

00 =Modification of the attribute is not allowed.

01 =Modification is allowed only in the initiation and termination phases, and only by the executing process. Processes external to the initiated process cannot modify this attribute.

11 =Modification is allowed in all phases and by all processes.

The bit assignment is as follows:

– Instruction wait access state control	Bits 0-1
– Time slice end access state control	Bits 2-3
– Time slice event option	Bits 4-5
– Exception event option	Bits 6-7
– Problem phase program option	Bits 8-9
– Termination phase program option	Bits 10-11
– Process default exception handler option	Bits 12-13
– Process NRL option	Bits 14-15
– Signal event control mask	Bits 16-17
– Process priority	Bits 18-19
– Process storage pool identification	Bits 20-21
– Maximum temporary auxiliary storage allowed	Bits 22-23
– Time slice interval	Bits 24-25
– Default wait timeout interval	Bits 26-27
– Maximum processor time allowed	Bits 28-29
– Process MPL class ID	Bits 30-31
– User profile pointer	Bits 32-33
– Process communication object pointer	Bits 34-35
– Process NRL pointer	Bits 36-37
– Termination phase program pointer	Bits 38-39
– Problem phase program pointer	Bits 40-41
– Process default exception handler	Bits 42-43
– Process adopted user profile list	Bits 44-45
– Process adopted user profile list option	Bits 46-47
– Reserved (binary 0)	Bits 48-63

• Reserved (binary 0)	Char(9)

The format of the process pointer attributes is as follows:

• Process user profile	System pointer
• PCO (process communication object)	System pointer Space pointer Data pointer or Char(16)
• Process NRL (name resolution list)	Space pointer
• Initiation phase program	System pointer

• Termination phase program	System pointer
• Problem phase program	System pointer
• PDEH (process default exception handler)	System pointer
• PASA (process automatic storage area)	Space pointer
• PSSA (process static storage area)	Space pointer
• PAG (process access group)	System pointer
• Process status indicators (see the *Materialize Process Attributes* instruction for the details of this attribute)	Char(13)*
• Reserved (binary 0)	Char(3)
• Process resource usage attributes* (see *Materialize Process Attributes* instruction for the details)	Char(14)*
• Subordinate process identification	Char(*)
– Number of immediately subordinate processes	Bin(2)
– Identification of subordinate processes	System pointer(s)
• Process adopted user profile list	Space pointer

Note: The values of the entries associated with an asterisk (*) are ignored by this instruction.

Authorization verification for all objects identified by pointers in the process definition template (except the process user profile, initiation phase program, termination phase program, and problem phase program) employs the user profile identified in the template, the user profiles identified in the process adopted user profile list, or employs the authorization previously set in the system pointers. The initiator must have object management authority for the new process user profile or the new process user profile must be identical to the initiating process user profile.

Process control attributes establish the basic process characteristics. The attributes and definitions are as follows:

- Process type (dependent/independent): This attribute denotes the upper boundary of the process hierarchy (domain). Designating a process as independent produces a direct-dependent relationship so that destruction of the initiator of an independent process does not cause implicit destruction of the independent process and its dependent subordinates. The initiator of an independent process, however, has implied full authority over that independent process and its dependents for explicit termination or suspension.

- Instruction wait access state control: This attribute specifies that the access state of the process access group can be modified when the process enters a wait as a result of a Dequeue, Lock, Wait On Event, Wait On Time, Suspend Process, or Set Cursor (for delete or update) instruction. If the parameter equals binary 1 and the instruction causing the wait also specifies an access state modification, the access state of the process access group is modified.

- Time slice end access state control: This attribute has the same function as the instruction wait access state control attribute, except for time slice end.

- Time slice event option (signal event/do not signal event): This attribute specifies that an event is to be signaled if a process has exhausted its time slice without having entered a wait as a result of a Dequeue, Lock, Wait On Event, Wait On Time, Suspend, or Set Cursor (for delete or update) instruction. The event is signaled if the time slice event option is set to signal event and the condition of the signal event is met.

- Initiation phase program option: This attribute specifies that a system pointer to the initiation phase program is supplied and that the initiation phase is to be entered.

- Problem phase option: This attribute specifies that a system pointer to the problem phase program is supplied and that the problem phase is to be entered. Either an initiation phase option or a program phase option must be specified. The template value invalid exception is signaled if one of the options is not specified.

- Termination phase option: This attribute specifies that a system pointer to the termination phase program is supplied and that the termination phase is to be entered.

- Process default exception handler option: This attribute specifies that a system pointer to a program is supplied as the process default exception handler.

- Process name resolution list option: This attribute specifies that a space pointer is supplied for the process NRL.

- Process adopted user profile list option: This attribute specifies if a space pointer is supplied which addresses a list of user profiles to be adopted by the process.

The signal event control mask controls the signaling of conditionally specified events. If the conditional signal mask in a Signal Event instruction is binary 0 or if one or more matching bit positions in the conditional signal mask and the signal event control mask are set to binary 1, the specified event is signaled.

The number of event monitors allows the machine to more effectively manage event monitors. This number is not a maximum; it represents a performance variable. The allowable value in this entry is from 0 through 256.

Resource management attributes define a process's limitations or restrictions in competing for machine resources. The attributes and definitions are as follows:

- Process priority: This attribute designates the relative importance of this process to other processes in the machine when contending for the processor and main storage. A value of 0 is the highest priority.

- Process storage pool identification: This attribute designates the main storage pool from which the machine is to draw for storage of the process's objects and machine overhead in support of a process. The storage pool identification must be one of the storage pools existing in the machine as defined by the machine attribute. The storage pool identification of hex 00 is reserved for the machine.

- Maximum temporary auxiliary storage allowed: This attribute restricts the amount of auxiliary storage for temporary system objects and machine overhead that a process can consume in the course of its existence.

- Time slice interval: This attribute specifies the amount of processor resource time to be given to the process until it is made temporarily ineligible for the processor.

- Default time-out interval: This attribute specifies a realtime interval that restricts the amount of time the process waits for an object to be made available, a message to arrive on a queue, or an event to occur. This value supplies a default when a wait time-out value is not specified on the Lock, Dequeue, or Wait On Event instruction. The bits in this field are numbered from 0 to 63, and bit 41 is defined as 1024 microseconds. The maximum wait time-out interval allowed is a value equal to (2^{48} - 1) microseconds. Any value that indicates more time than the maximum wait time-out causes the maximum wait time-out to be used.

- Maximum processor time allowed: This attribute specifies the maximum amount of processor time that a process may consume during its existence. An event is signaled when the specified value is exceeded.

- Process MPL (multiprogramming level) class ID: This attribute is used to associate the MPL class of the new process with a previously specified MPL class set as a machine attribute.

Modification control indicator attributes restrict the modification of process attributes through the Modify Process Attributes instruction. Modification of the process can be disallowed, restricted to modification by the process itself only in the initiation and termination phases, or allowed in all phases by any process with proper authority. External modification is allowed implicitly to the initiator of this process, provided the modification control indicators are set to allow modification in all phases. Other processes are allowed to modify this process if they have the special process control authority within their process user profile or current adopted user profile and the modification control indicators of this process are set to allow modification in all phases. The modification control indicators cannot be modified.

The process user profile system pointer is required and identifies the user profile that is to govern the execution of the process. The user profile governing the process issuing the Initiate Process instruction must have object management authorization for the designated user profile or must be identical to the designated user profile. The process user profile provides the basic authorization control for the process. Permanent system object storage allocation and ownership of objects created by the process are always reflected in the user profile specified in the process definition template. A process's authorization can be augmented through specification of a process adopted user profile list or through the invocation of a program created with an adopted user profile. Adopted user profiles are used in conjunction with the process user profile to determine a process's eligibility for access to existing objects, privileged instructions, or special authorizations.

An implicit lock is applied to the process user profile for the duration of the initiation of the process. If a process holds an LENR lock on the user profile, an invalid lock state exception is signaled. The implicit lock is removed when the process is terminated.

The PCO (process communication object) pointer provides addressability to a user object whose use and format is an external convention. The area may contain a system pointer, a space pointer, a data pointer, or any data value. The contents of the area are not verified by the machine. If a PCO is not used, the associated storage area may contain any value.

The process NRL (name resolution list) pointer is a space pointer that provides addressability to a list of resolved system pointers addressing contexts to be used by the machine for address resolution. The list of system pointers is preceded by a binary(2) scalar denoting the number of system pointers in the list. The space pointer must address a 16-byte boundary that has the following format:

• Number of pointers	Bin(2)
• Reserved (binary 0)	Char(14)
• List of resolved system pointers to contexts	System pointer(s)

The process NRL is optional. If not specified, it causes the object not found exception to be signaled when a context is not specified for explicit or implicit system pointer name resolution functions.

The initiation phase program pointer is optional. If specified, it identifies the first program to be given control by the machine at the completion of the Initiate Process instruction. The initiation phase option parameter establishes whether the initiation phase is to be entered and whether the designated program is to be invoked.

The termination phase program pointer is optional. If specified, it indicates the program to be given control when the process enters the termination phase. The termination phase option parameter establishes whether the termination phase is to be entered and whether the designated program (if specified) is to be invoked.

The problem phase program pointer identifies the program to be invoked when the process enters the problem phase. The problem phase option parameter establishes whether the problem phase is to be entered and whether the designated program is to be invoked.

The PDEH (process default exception handler) pointer is optional. If it is present, it identifies the program to be given control when an exception is not handled by a signaled program invocation. This program is invoked as an external exception handler and as the most current invocation in the process. No invocations within the process are destroyed prior to invoking the PDEH. When no PDEH is specified, the process is terminated if a signaled invocation does not handle an exception. The PDEH is given control when an exception occurs invoking the signaled invocation program. The exception data intended for the signaled invocation program is presented to the PDEH.

The PASA (process automatic storage area) space pointer is required and addresses a location in a space the machine uses to allocate invocations. The space pointer must address a 16-byte boundary. The space pointer locates the PASA base entry. At this location, the user must have constructed the PASA base entry. The base entry consists of three space pointers. The succeeding bytes in the space are then assumed to be available for allocation for invocation entries. These entries contain stack control pointers, data, and the automatic storage allocated for program invocations. See the *Call External* instruction, for a description of the PASA.

The PASA base entry must be initialized prior to the Initiate Process instruction that specifies the PASA base entry in the PDT (process definition template). The initialization of the base entry consists of the following:

• Current invocation entry (initialized to address the PASA base entry)	Space pointer
• First invocation entry (need not be initialized, and any value present is ignored by the machine)	Space pointer
• Next available storage location (initialized to the byte location in the space where the first invocation entry is to be allocated)	Space pointer

The space that contains the PASA can be permanent or temporary and can be contained in an access group only if the space is temporary.

The PSSA (process static storage area) space pointer is optional. If this pointer is present, it addresses a space the machine uses to stack activations. The PSSA space pointer is an optional parameter, provided no programs executed within the process require static storage. The space pointer must address a 16-byte boundary and locates the beginning of the PSSA. The succeeding bytes in the space are assumed to be available for allocation for activation entries. These entries contain stack control pointers, data, and the static storage allocated for program activations. See the *Activate Program* instruction for a description of the PSSA.

If a valid PSSA space pointer is not provided prior to the first program activation or prior to the invocation within the process that requires static storage, a pointer does not exist, pointer type invalid, or a space addressing violation exception is signaled at activation or invocation of the program.

The PSSA base entry must be initialized prior to the activation of the first program in the process. The base entry consists of the following:

• Current activation entry (initialized to address the PSSA base entry)	Space pointer
• First activation entry (need not be initialized, and any value present is ignored by the machine)	Space pointer
• Next available storage location (initialized to the byte location in the space where the first activation entry is to be allocated)	Space pointer

The space containing the PSSA can be permanent or temporary and can be contained in an access group only if the space is temporary.

The PAG (process access group) system pointer is optional. If this pointer is present, it addresses an access group that will be managed by the machine at instruction wait entry, at time slice end, and at process predispatching times. The PAG access state modification is controlled by the instruction wait access state control and time slice end access state control indicators, in conjunction with access state modification options supplied with the Dequeue, Lock, Wait On Event, Wait On Time, Suspend Process, and Set Cursor (for delete or update) instructions. The access group and its member objects can be referenced by other processes at any time.

The subordinate process identification information is ignored unless a process adopted user profile list is specified. In this case, the number of immediately subordinate processes field is interrogated for the value (possibly zero) specifying the number of pointers which follow prior to the process adopted user profile list space pointer. The value(s) of the pointers which may be specified as following it are ignored. This number of pointers value is used only to determine the location of the space pointer to the process adopted user profile list.

The process adopted user profile list space pointer is optional and if present identifies user profiles which the process adopts at initiation in addition to the process user profile. The user profile governing the process issuing the Initiate Process instruction must have operational authorization for the designated user profiles. Note that to specify this attribute the number of immediately subordinate processes field in the subordinate process identification information must also be set with a value which is used in determining the location of the process adopted user profile list space pointer.

The list of system pointers addressing these user profiles is preceded by a binary(2) scalar denoting the number of system pointers in the list. The space pointer must address a 16-byte boundary that has the following format:

• Number of pointers	Bin(2)
• Reserved	Char(14)
• List of system pointers to user profiles	System pointer(s)

The process adopted user profiles provide an additional authorization control for the process. Note, however, that system object storage allocation and ownership of objects created by the process are always reflected in the process user profile which must always be specified in the process definition template (see the earlier description of process user profile). A process' authorization may be augmented through the invocation of a program created with an adopted user profile. Adopted user profiles are used in conjunction with the process user profile and the process adopted user profiles to determine a process' eligibility for access to existing objects, privileged instructions, or special authorizations.

Operand 3 can identify an argument list to be presented to the process being initiated, or it can be null (no arguments passed). The argument and parameter functions are the same as defined for interinvocation communications (argument list on Call External and Transfer Control instructions). Refer to *Program Execution* in the *Functional Concepts Manual* for the details on argument/parameter correspondence.

The argument to parameter relationship is established for only the first program invoked in the problem phase. The process initiation and termination programs are not given access to the argument list.

Operand 4 can identify a space that specifies system objects whose locks are to be transferred to the new process. The template identified by operand 4 must be 16-byte aligned in the space.

The space object is organized such that a 1-byte lock state selection entry exists in the space for each addressing object in the template. The addressing objects must be system pointers if the associated lock option entry is active; otherwise, an exception is signaled. If the entry is not active, the associated addressing object is ignored.

The format of the lock template is as follows:

• Number of lock transfer requests	Bin(4)
• Offset to lock state selection entries	Bin(2)
• Reserved (binary 0)	Char(10)
• Object lock(s) to be transferred – System pointer for each object lock to be transferred	System pointer(s)
• Lock state selection entry (repeated for each addressing object in the template)	Char(1)
– Lock state to transfer (only one state may be requested: 1 = transfer)	Bits 0-4
transfer LSRD state	Bit 0
transfer LSRO state	Bit 1
transfer LSUP state	Bit 2
transfer LEAR state	Bit 3
transfer LENR state	Bit 4
– Set lock count option 0 = Transfer the current lock count 1 = Transfer a lock count of 1	Bit 5
– Reserved (binary 0)	Bit 6
– Entry active indicator 0 = Entry not active (do not use this entry and associated system pointer) 1 = Entry active (transfer this lock)	Bit 7

Only one lock state can be transferred within each entry.

The initiating process must hold the locks in the states that are to be transferred, or an exception is signaled.

The initiating process cannot transfer a subset of lock states (to the new process) that would result in conflicting locks. For example, the initiating process could not hold an object locked in the LENR and LSRD state and transfer only the LSRD state. The object not available to process exception is signaled if the transfer request results in conflicting lock states.

See the *Transfer Object Lock* instruction for associated functions and exceptions.

Authorization Required

- Privileged Instruction

- Object management
  - User profile specified as the process user profile when the user profile is different than user profile of the process that issued the instruction

- Retrieve
  - Contexts referenced for address resolution

- Authorized operational to initiated processes
  - Process adopted user profiles

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Process control space

Implicit Locks

- User profile of process to be initiated is implicitly locked LSRD

Events

0002 Authorization
    0201 Privileged instruction violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0102 Process initiated (to initiating process)
    0202 Process terminated (to initiating process)
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	4	Other
06 Addressing					
01 Space addressing violation	X	X		X	
02 Boundary alignment	X	X		X	
03 Range	X	X		X	
06 Optimized addressability invalid	X	X		X	
08 Argument/Parameter					
01 Parameter reference violation	X	X		X	
02 Initiate process				X	
0A Authorization					
01 Unauthorized for operation					X
02 Privileged instruction					X
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1C Machine-Dependent Exception					
01 Machine-dependent request invalid					X
03 Machine storage limit exceeded					X
04 Object storage limit exceeded	X				
06 Machine lock limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X		X	
02 Object destroyed	X	X		X	
03 Object suspended	X	X		X	
05 Object not available to process	X	X			
24 Pointer Specification					
01 Pointer does not exist	X	X		X	
02 Pointer type invalid	X	X		X	
03 Pointer addressing invalid object	X	X			
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X		X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
38 Template Specification					
01 Template value invalid		X		X	

MATERIALIZE PROCESS ATTRIBUTES (MATPRATR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0333	Receiver	Process control space	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer or null.

Operand 3: Character scalar(1).

Description: The instruction causes either one specific attribute or all the attributes of the designated process to be materialized.

Operand 1 specifies a space that is to receive the materialized attribute values. The space pointer specified in operand 1 must address a 16-byte aligned area.

Operand 2 is a system pointer identifying the process control space associated with the process whose attributes are to be materialized. If operand 2 is null, the process issuing the instruction is the subject process. If the subject process's attributes are being materialized by another process, that process must be the original initiator of the subject process or the governing user profile(s) must have process control special authorization.

Operand 3 is a character scalar(1) specifying which process attribute is to be materialized. A value of hex 00 results in all the attributes of a process being materialized in the format described in the Initiate Process instruction for the process definition template. Other options allow materialization of specialized process attributes.

The materialization template has the following general format when a process scalar attribute is materialized:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Process scalar attributes	Char(*)

The materialization template has the following general format when a process pointer attribute is materialized:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Reserved (binary 0)	Char(8)
• Process pointer attribute	System pointer or Space pointer

Note: The values of the entry associated with an asterisk (*) are ignored by this instruction.

The following attributes require materialization targets of varying lengths. The attributes to be materialized and their operand 3 materialization option values follow.

• Process control attributes	Char(4)

Values hex 01 through hex 0B or hex 27 cause the 4-byte process control attributes value to be placed in the byte area identified by operand 1. The individual attributes and the corresponding values are as follows:

– Process type 0 = Dependent process 1 = Independent process	Bit 0
– Instruction wait access state control 0 = Access state modification is not allowed 1 = Access state modification is allowed if specified	Bit 1
– Time slice end access state control 0 = Access state modification is not allowed 1 = Access state modification is allowed if specified	Bit 2
– Time slice end event option 0 = Time slice expired without entering instruction wait event is not signaled 1 = Time slice expired without entering instruction wait event is signaled	Bit 3
– Reserved (binary 0)	Bit 4
– Initiation phase program option 0 = No initiation phase program specified (do not enter initiation phase) 1 = Initiation phase program specified (enter initiation phase)	Bit 5
– Problem phase program option 0 = No problem phase program specified (do not enter problem phase) 1 = Problem phase program specified (enter problem phase)	Bit 6

- Termination phase program option Bit 7
  - 0 = No termination phase program specified (do not enter termination phase)
  - 1 = Termination phase program specified (enter termination phase)
- Process default exception handler option Bit 8
  - 0 = No process default exception handler
  - 1 = Process default exception handler specified
- Process NRL (name resolution list) option Bit 9
  - 0 = No process NRL specified
  - 1 = Process NRL specified
- Process access group option Bit 10
  - 0 = No process access group specified
  - 1 = Process access group specified
- Process adopted user profile list option Bit 11
  - 0 = No process adopted user profile list specified
  - 1 = Process adopted user profile list specified
- Reserved (binary 0) Bits 12-31

- Signal event control mask

  The materialization of the control mask is as follows:
  - Hex 0C = Signal event control mask Char(2)

- Number of event monitors

  The materialization of this attribute is as follows:
  - Hex 0D = Number of event monitors Bin(2)

The resource management attributes and data types are as follows:

- Hex 0E = Process priority Char(1)
- Hex 0F = Process storage pool ID Char(1)
- Hex 10 = Maximum temporary auxiliary storage allowed Bin(4)
- Hex 11 = Time slice interval Char(8)
- Hex 12 = Default time-out interval Char(8)
- Hex 13 = Maximum processor time allowed Char(8)
- Hex 14 = Process multiprogramming level class ID Char(1)
- Hex 15 = Modification control indicators Char(8)

The modification control indicators are materialized when the operand 3 value is hex 15. Each indicator specifies the modification options allowed to a process upon itself by the initiating process. The possible values of each modification control indicator are as follows:

00 = Modification of the attribute is not allowed.

01 = Modification is allowed in the initiation or termination phases only.

10 = Modification is allowed in all phases (initiation, problem, and termination).

The bit assignments of the modification control indicators are as follows:

– Instruction wait access state control	Bits 0-1
– Time slice end access state control	Bits 2-3
– Time slice event option	Bits 4-5
– Reserved (binary 0)	Bits 6-7
– Problem phase program option	Bits 8-9
– Termination phase program option	Bits 10-11
– Process default exception handler option	Bits 12-13
– Process NRL option	Bits 14-15
– Signal event control mask	Bits 16-17
– Process priority	Bits 18-19
– Process storage pool identification	Bits 20-21
– Maximum temporary auxiliary storage allowed	Bits 22-23
– Time slice interval	Bits 24-25
– Default time-out interval	Bits 26-27
– Maximum processor time allowed	Bits 28-29
– Process MPL class ID	Bits 30-31
– User profile pointer	Bits 32-33
– Process communication object pointer	Bits 34-35
– Process NRL pointer	Bits 36-37
– Termination phase program pointer	Bits 38-39
– Problem phase program pointer	Bits 40-41
– Process default exception handler	Bits 42-43
– Process adopted user profile list	Bits 44-45
– Process adopted user profile list option	Bits 46-47
– Reserved (binary 0)	Bits 48-63

The process pointer attributes which may be materialized are the following:

- Hex 16=Process user profile pointer

  The system pointer with addressability to the user profile is placed into the space addressed by operand 1. If the materialization option (hex 00) is specified in operand 3, a reserved character(9) field is included at this point. This user profile is the process user profile assigned by the Initiate Process or Modify Process Attribute instruction.

- Hex 17=Process communication object (PCO) pointer

  The PCO pointer is placed in the space addressed by operand 1.

- Hex 18=Process name resolution List

  The space pointer to the NRL is placed in the space addressed by operand 1.

- Hex 19=Initiation phase program pointer

  The system pointer to the program is placed in the space addressed by operand 1.

- Hex 1A=Termination phase program pointer

  The system pointer to the program is placed in the space addressed by operand 1.

- Hex 1B=Problem phase program pointer

  The system pointer to the program is placed in the space addressed by operand 1.

- Hex 1D=Process automatic storage area

  The space pointer with addressability to the PASA is placed in the space addressed by operand 1.

- Hex 1E=Process static storage area

  The space pointer with addressability to the PSSA is placed in the space addressed by operand 1.

- Hex 1F=Process access group

  The system pointer with addressability to the PAG is placed in the space addressed by operand 1.

Process status indicators are materialized when the value of operand 3 is hex 20. The format and associated values of this attribute are as follows:

- Process states Char(2)
  - External existence state Bits 0-2
    000 = Suspended
    010 = Suspended, in instruction wait
    100 = Active, in ineligible wait
    101 = Active, in current MPL
    110 = Active, in instruction wait
  - Invocation exit active Bit 3
  - Reserved (binary 0) Bits 4-7
  - Internal processing phase Bits 8-10
    001 = Initiation phase
    010 = Problem phase
    100 = Termination phase
  - Reserved (binary 0) Bits 11-15

- Process interrupt status Char(2)
  (Bit = 1 denotes pending)
  - Time slice end pending Bit 0
  - Transfer lock pending Bit 1
  - Asynchronous lock retry pending Bit 2
  - Suspend process pending Bit 3
  - Resume process pending Bit 4
  - Resource management attribute modify pending Bit 5
  - Process attribute modify pending Bit 6
  - Terminate machine processing pending Bit 7
  - Terminate process pending Bit 8
  - Wait time-out pending Bit 9
  - Event schedule pending Bit 10
  - Machine service pending Bit 11
  - Invocation exit active Bit 12
  - Reserved (binary 0) Bits 13-15

- Process initial internal termination status Char(3)
  - Initial internal termination reason Bits 0-7
    Hex 80=Return from first invocation in problem phase.
    Hex 40=Return from first invocation in initiation phase, and no problem phase program specified.
    Hex 20=Terminate Process instruction issued by this process to itself.
    Hex 10=Exception was not handled by the process.
    Hex 00=Process terminated externally.
  - Initial internal termination code Bits 8-23

    The code is assigned in one of the following ways:

    a. If the termination is caused by a Return External instruction from the first invocation, then this code is binary 0's.
    b. The code is assigned by operand 2 of the Terminate Process instruction. This code is also given to subordinate processes involved in the termination.
    c. The code is assigned by the original exception code that caused process termination to commence. This code is also given to subordinate processes involved in the termination.

- Process initial external termination status Char(3)
  - Initial external termination reason: Bits 0-7
    Hex 80=Terminate Process instruction issued explicitly to this process from another process.
    Hex 40=A superordinate process has been terminated.
    Hex 00=Process terminated internally.
  - Initial external termination code: Bits 8-23
    This code is supplied by the termination code in operand 2 of the Terminate Process instruction.

• Process final termination status	Char(3)
– Final termination reason: Hex 80=Return instruction from first invocation. Hex 40=Terminate Process instruction issued by the process being materialized. Hex 20=Terminate Process instruction issued to the process being materialized by another process. Hex 10=Exception not handled by this process. Hex 08=Terminate Process instruction issued to superordinate of the process being materialized. Hex 04=Superordinate process of the process being materialized completed termination phase.	Bits 0-7
– Final termination code is assigned in one of the following ways: a. If the termination is caused by a Return External instruction from first invocation, then this code is binary 0's. b. The termination code is assigned by the Terminate Process instruction. c. The termination code is assigned by the original exception code that caused process termination.	Bits 8-23

The process final termination status is presented as event-related data in the terminate process event. Usually the event is the only source of the process final termination status since the process will cease to exist before its attributes can be materialized.

Process resource usage attributes are materialized when the value of operand 3 is hex 21. The format and associated values of this attribute are as follows:

• Total temporary auxiliary storage used	Bin(4)
• Total processor time used	Char(8)
• Number of locks currently held by the process (including implicit locks)	Bin(2)

Subordinate processes identification attributes are materialized when the value of operand 3 is hex 22. The format and associated values of this attribute are as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Number of immediately subordinate processes	Bin(2)
• Reserved (binary 0)	Char(6)
• System pointer to the process control space for each subordinate process (repeated for each immediately subordinate process)	System pointer(s)

Process performance attributes are materialized when the value of operand 3 is hex 23. The format and associated values of this attribute are as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Number of page reads into main storage associated with data base	Bin(4)
• Number of page reads into main storage not associated with data base	Bin(4)
• Number of page writes from main storage	Bin(4)
• Number of transitions into ineligible wait state	Bin(2)
• Number of transitions into an instruction wait	Bin(2)
• Number of transitions into ineligible wait state from an instruction wait	Bin(2)
• Timestamp of materialization	Char(8)

Each of these counters has a limit of 32 767. If this limit is exceeded, the count is set to 0, and no exception is signaled.

The process performance attributes are not supplied with materialization option hex 00.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes a No exceptions (other than the materialization length exception described previously) are signaled in the event that the receiver contains insufficient area for the materialization.

Process execution status attributes are materialized when the value of operand 3 is hex 24. The format and associated values of this attribute are as follows:

• Process priority	Char(2)
– Machine interface priority	Char(1)
– Machine adjusted priority Normal value is hex 80. This value is dynamically modified by the machine.	Char(1)
• Pending interruptions	Char(2)
– Time slice end	Bit 0
– Transfer lock	Bit 1
– Asynchronous lock retry	Bit 2
– Suspend process	Bit 3
– Resume process	Bit 4
– Modify resource management attribute	Bit 5
– Modify process attribute	Bit 6
– Terminate machine processing	Bit 7
– Terminate process	Bit 8
– Wait time-out	Bit 9
– Event	Bit 10
– Machine service	Bit 11
– Reserved (binary 0)	Bits 12-15
• Execution status	Char(2)
– Suspended	Bit 0
– Instruction wait	Bit 1
– In MPL	Bit 2
– Ineligible wait	Bit 3
– Reserved (binary 0)	Bits 4-15
• Wait status	Char(2)
– Wait on event	Bit 0
– Dequeue	Bit 1
– Lock	Bit 2
– Wait on time	Bit 3
– Reserved (binary 0)	Bits 4-15
• Process class identification	Char(2)
– Storage pool class	Char(1)
– MPL class	Char(1)

• Processor time used	Char(8)
• Performance attributes	Char(18)
– Number of data base read operations	Bin(4)
– Number of nondata base read operations	Bin(4)
– Number of write operations	Bin(4)
– Transitions to ineligible wait	Bin(2)
– Transitions to instruction wait	Bin(2)
– Transitions to ineligible from instruction wait	Bin(2)

A system pointer to the process control space is materialized when the value of operand 3 is hex 25. If a process control space pointer is supplied in operand 2, it is ignored. A pointer to the process that is executing the MATPRATR instruction is always materialized.

A materialization option's value of hex 26 causes the adopted user profile list attributes to be materialized as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Reserved (binary 0)	Char(8)
• Pointer to the adopted user profile list last used to set this attribute	Space pointer
• Number of user profiles in the encapsulated adopted user profile list	Bin(2)
• Reserved	Char(14)
• List of user profiles in the encapsulated adopted user profile list (one system pointer to each user profile in the list)	System pointers

Due to verifications performed on the user profiles specified in an adopted process user profile list input to either the Initiate Process or Modify Process instructions, the encapsulated adopted user profile list may differ from the input list. When verification of an input user profile fails, it is not included in the encapsulated list.

The adopted user profile list attributes are not supplied with materialization option hex 00.

A materialization option's value of hex 27 causes the process control attributes to be materialized. Refer to the description of this materialization provided in prior text for this instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Process control special authorization
  - For materializing a process other than the one executing this instruction
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X		
02 Boundary alignment	X	X	X		
03 Range	X	X	X		
06 Optimized addressability invalid	X	X	X		
08 Argument/Parameter					
01 Parameter reference violation	X	X	X		
0A Authorization					
01 Unauthorized for operation		X			
04 Unauthorized for process control		X			
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state				X	
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X		
02 Object destroyed	X	X	X		
03 Object suspended	X	X	X		
24 Pointer Specification					
01 Pointer does not exist	X	X	X		
02 Pointer type invalid	X	X	X		
03 Pointer addressing invalid object		X			
28 Process State					
02 Process control space not associated with a process		X			
2A Program Creation					
06 Invalid operand type	X	X	X		
07 Invalid operand attribute	X	X	X		
08 Invalid operand value range	X	X	X		
0A Invalid operand length	X				
0C Invalid operand ODT reference	X	X	X		
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
03 Scalar value invalid		X			
38 Template Specification					
03 Materialization length exception	X				

MODIFY PROCESS ATTRIBUTES (MODPRATR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0337	Process control space	Modification template	Modify attribute

Operand 1: System pointer or null.

Operand 2: Space pointer.

Operand 3: Character(1) scalar (fixed-length).

Description: An attribute of the process identified by operand 1 is modified to the value specified by operand 2. Operand 3 identifies the attribute that is to be modified.

If the process is attempting to modify itself (that is, operand 1 is null or operand 1 designates the process itself), the modification is allowed or disallowed based on the modification control indicators specified in the process definition template supplied with the Initiate Process instruction. Modification is also conditioned on the internal phases: initiation, problem or termination phase.

The initiating process always carries implicit modify authority. Any other process can modify another process if the process control special authorization is defined in the process user profile or in a current adopted user profile, provided the modification control indicators are set to allow modification in all phases.

Operand 1 is a system pointer addressing a process control space associated with a process.

Because this instruction may require one process to act upon another process, a portion of the function is controlled by the issuing process, and the remainder of the function is controlled by the target process. When control is returned to the issuing process, the function may not have been performed in its entirety.

The action the machine takes upon modification of an attribute may cause an immediate effect, or the effect may be delayed. The effect will be immediate only if the process is modifying itself; otherwise, it will be delayed to an instruction boundary of the target process. However, if the process termination phase program is changed, the modification does not influence the process until the process enters the termination phase.

When a process scalar attribute is being modified, the modification template has the following general format:

• Template size	Char(8)
– Number of bytes provided	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Scalar modification value	Char(*)

When a process pointer attribute is being modified, the modification template has the following general format (and must be aligned on a 16-byte multiple):

• Template size	Char(8)
– Number of bytes provided	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Reserved (binary 0)	Char(8)
• Process pointer attribute	System pointer or Space pointer

The template identified by operand 2 must be 16-byte aligned in the space.

Operand 3 is a character(1) scalar specifying the process attributes to be modified.

The following attributes require modification values of varying lengths. The attributes and their operand 3 character(1) scalar values are as follows:

- Process control attributes Char(4)

  Bits that are not selected in this option are ignored by this instruction. The following attribute bits can be selected:

  - Hex 02=Instruction wait access state control Bit 1
    - 0 = Access state modification is not allowed
    - 1 = Access state modification is allowed if specified

    The machine recognizes the new value at the next instruction wait by the process.

  - Hex 03=Time slice end access state control Bit 2
    - 0 = Access state modification is not allowed
    - 1 = Access state modification is allowed if specified

    The machine recognizes the new value at the next time slice end for the process.

- Hex 04=Time slice event option — Bit 3
  - 0 = No event is signaled if time slice end occurred without a long wait during the time slice
  - 1 = An event is signaled if time slice end occurred without a long wait during the time slice

  The machine recognizes the new value at the next time slice end.

- Hex 05=Exception event option — Bit 4
  - 0 = No event is signaled upon exception occurrence
  - 1 = An event is signaled upon exception occurrence

- Hex 07=Problem phase program option — Bit 6
  - 0 = No problem phase program specified (do not enter the problem phase)
  - 1 = Problem phase program specified (enter the problem phase)

- Hex 08=Termination phase program option — Bit 8
  - 0 = No termination phase program specified (do not enter termination phase)
  - 1 = Termination phase program specified (enter the termination phase)

- Hex 09=Process default exception handler option — Bit 8
  - 0 = No process default exception handler specified
  - 1 = Process default exception handler specified

- Hex 0A=Process name resolution list option — Bit 9
  - 0 = No process name resolution list
  - 1 = Process name resolution list specified

- Hex 27=Process adopted user profile list option — Bit 11
  - 0 = No process adopted user profile list
  - 1 = Process adopted user profile list specified

  Bit 11 of the scalar modification value replaces the current process value.

- Signal event control mask

  The modification of the control mask is:

  – Hex 0C=Signal event control mask Char(2)

    The machine recognizes the change on the next conditional Signal Event instruction that is encountered.

The resource management attributes and data types are as follows:

- Hex 0E=Process priority Char(1)

  The scalar modification value replaces the current process priority. If the process is active, its position relative to other processes contending for the same resource is immediately adjusted.

- Hex 0F=Process storage pool identification Char(1)

  The scalar modification value replaces the current process value. If the process is active, subsequent main storage requirements are satisfied from the new storage pool. The release of main storage acquired from other storage pools is unpredictable.

- Hex 10=Maximum temporary auxiliary storage allowed Bin(4)

  The scalar modification value replaces the current process value. The new value is checked the next time auxiliary storage is required to determine if the scalar modification value has been exceeded.

- Hex 11=Time slice interval Char(8)

  The scalar modification value replaces the current process time slice value. The new time slice value takes effect the next time the process is dispatched.

- Hex 12=Default time-out interval Char(8)

  The scalar modification value replaces the current process value. The new new value is used the next time the process executes a Dequeue Lock or Wait On Event instruction that specifies a zero time-out value.

- Hex 13=Maximum processor time allowed Char(8)

  The scalar modification value replaces the current process value. The new value is used at the end of the next time slice to determine if the maximum allowed processor time has been exceeded.

- Hex 15=Process multiprogramming level class ID Char(1)

  The effect of the modification is immediate, but the MPL rules are not applied until the next instruction wait or time slice end.

The process pointer attributes and data types are as follows:

- Hex 16=User profile pointer System pointer

  Modification of this attribute is reflected in the next authority verification for the process or upon creation of a permanent system object by the process.

- Hex 17=Process communication object pointer Space pointer system pointer, data pointer, or scalar

  Modification of this attribute is reflected only upon the next Materialize Process Attributes instruction.

- Hex 18=Process name resolution list pointer — Space pointer

  The machine references this list for subsequent address resolutions.

- Hex 1A=Termination phase program pointer — System pointer

  The new program is to be used when the process enters the termination phase. The system pointer must address a program.

- Hex 1B=Problem phase program pointer — System pointer

  The new program is to be used when the process enters the problem phase. The system pointer must address a program.

- Hex 1C=Process default exception pointer — System pointer

  The program is to be activated and invoked if an exception is not handled at the program invocation level. The system pointer must address a program.

- Hex 26=Process adopted user profile list pointer — System pointer

  The machine uses this list for the next authority verification. The list has the format as described in the Initiate Process instruction. This list of user profiles completely replaces a previously provided list.

The modification control indicators can not be modified through the Modify Process Attributes instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Process control special authorization

- Retrieve
  - Contexts referenced for address resolution

- Object Management
  - User profile (new) of the process if the process user profile is to be changed

- Operational
  - Process adopted user profile (new) of the process if the process adopted user profile list is to be changed

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Object management
  - User profile (new) of the process if the process user profile is to be changed
  - User profile of process is locked LSRD when the process user profile is being modified
  - The previous process user profile has its LSRD lock removed

Events

0002 Authorization
    0301 Special authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	4	Other
06 Addressing					
01 Space addressing violation	X	X	X		
02 Boundary alignment	X	X	X		
03 Range	X	X	X		
06 Optimized addressability invalid	X	X	X		
08 Argument/Parameter					
01 Parameter reference violation	X	X	X		
0A Authorization					
01 Unauthorized for operation	X				
04 Unauthorized for process control	X				
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state					X
1C Machine-Dependent Exception					
04 Object storage limit exceeded					X
06 Machine lock limit					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X		
02 Object destroyed	X	X	X		
03 Object suspended	X	X	X		
24 Pointer Specification					
01 Pointer does not exist	X	X	X		
02 Pointer type invalid	X	X	X		
03 Pointer addressing invalid object	X				
28 Process State					
02 Process control space not associated with a process	X				
0A Process attribute modification not allowed			X		
2A Program Creation					
06 Invalid operand type	X	X	X		
07 Invalid operand attribute	X	X	X		
08 Invalid operand value range	X	X	X		
0A Invalid operand length	X	X	X		
0C Invalid operand ODT reference	X	X	X		
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
03 Scalar value invalid			X		
38 Template Specification					
01 Template value invalid		X			
02 Template size invalid		X			

RESUME PROCESS (RESPR)

Op Code (Hex)	Operand 1	Operand 2
0386	Process control space	Option template

Operand 1: System pointer or null.

Operand 2: Character(1) scalar (fixed-length).

Description: The designated process or processes are made eligible for the processor resource. The affected processes are denoted by the operand 1 and operand 2 values.

If operand 1 is a system pointer, it must identify the process control space associated with a process to be resumed. If operand 1 is null, the executing process is identified and its subordinate processes are resumed.

The process issuing the Resume Process instruction requires no authority if the resuming process is the initiator of the target process. If this condition is not met, the resuming process must carry the process control special authorization in its process user profile or any current adopted user profile(s).

Operand 2 is a character scalar designating the resume process option. The format is:

- Resume option — Char(1)
  - Resume domain — Bits 0-1
    - 01 = Root process only
    - 10 = All subordinate processes only
    - 11 = Root process and all subordinate processes
  - Reserved (binary 0) — Bits 2-7

If operand 1 identifies the issuing process, the resume option must designate all subordinate processes only; otherwise, the scalar value invalid exception is signaled.

The suspended process or processes are resumed in the same internal processing phase as they existed in when they were suspended. The phases may be initiation, problem, or termination.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Process control special authorization
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
 0101 Object authorization violation
 0301 Special authorization violation

000C Machine resource
 0201 Machine auxiliary storage threshold exceeded

0010 Process
 0402 Process resumed
  (signaled to initiating process)
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X			
02 Boundary alignment	X	X			
03 Range	X	X			
06 Optimized addressability invalid	X	X			
08 Argument/Parameter					
01 Parameter reference violation	X	X			
0A Authorization					
01 Unauthorized for operation	X				
04 Unauthorized for process control	X				
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state					X
1C Machine-Dependent Exception					
04 Object storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X			
02 Object destroyed	X	X			
03 Object suspended	X	X			
24 Pointer Specification					
01 Pointer does not exist	X	X			
02 Pointer type invalid	X	X			
03 Pointer addressing invalid object	X				
28 Process State					
02 Process control space not associated with a process	X				
2A Program Creation					
06 Invalid operand type	X	X			
07 Invalid operand attribute	X	X			
08 Invalid operand value range	X	X			
0A Invalid operand length	X	X			
0C Invalid operand ODT reference	X	X			
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
01 Scalar type invalid	X	X			
03 Scalar value invalid		X			

SUSPEND PROCESS (SUSPR)

Op Code (Hex)	Operand 1	Operand 2
0392	Process control space	Option template

Operand 1: System pointer or null.

Operand 2: Character(1) scalar.

Description: Designated processes are suspended based on the process or processes identified by operand 1 and the suspend options specified in operand 2.

Operand 1 identifies the process to be suspended. The operand 1 system pointer addresses the process control space associated with the process to be suspended. If operand 1 is null, the process issuing the instruction is considered the process to be suspended.

No authorization is required if one of the following conditions exists:

- The suspending process is the initiator of the target process.
- The process is suspending itself.

If neither condition exists, the suspending process must carry the process control special authorization in its process user profile or currently adopted user profile(s).

Operand 2 is a character(1) scalar designating the suspend option. The format is:

• Suspend option	Char(1)
– Suspend domain	Bits 0-1
01 = Suspend root process only	
10 = Suspend all subordinate processes only	
11 = Suspend root process and all subordinates	
– Access state control	Bit 2
0 = Access state is not modified	
1 = Access state is modified	
– Reserved (binary 0)	Bits 3-7

A process can be suspended in any internal processing phase: initiation, problem, or termination.

If any process designated to be suspended has already been suspended, no operation is performed on the process, and no exception is signaled. If the suspend option specifies subordinate processes and the referenced process has no subordinates, no exception is signaled.

If the access state control parameter specifies modify access state and the process's or processes' instruction wait access state control specifies allow access state modification, then the access state of the process's access group is modified.

Suspended processes retain locks. Processes in the suspended state can be operated on with the Materialize Process Attributes, Modify Process Attributes, Resume Process Attributes, and Terminate Process Attributes instructions.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Process control special authorization
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0302 Process suspended
        (signaled to initiating process)
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X			
02 Boundary alignment	X	X			
03 Range	X	X			
06 Optimized addressability invalid	X	X			
08 Argument/Parameter					
01 Parameter reference violation	X	X			
0A Authorization					
01 Unauthorized for operation	X				
04 Unauthorized for process control	X				
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
1A Lock State					
01 Invalid lock state					X
1C Machine-Dependent Exception					
04 Object storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X			
02 Object destroyed	X	X			
03 Object suspended	X	X			
24 Pointer Specification					
01 Pointer does not exist	X	X			
02 Pointer type invalid	X	X			
03 Pointer addressing invalid object	X				
28 Process State					
02 Process control space not associated with a process	X				
2A Program Creation					
06 Invalid operand type	X	X			
07 Invalid operand attribute	X	X			
08 Invalid operand value range	X	X			
0A Invalid operand length	X	X			
0C Invalid operand ODT reference	X	X			
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
03 Scalar value invalid		X			

TERMINATE INSTRUCTION (TERMINST)

Op Code (Hex)	Operand 1	Operand 2
0342	Process control space	Termination option

Operand 1: System pointer.

Operand 2: Character(3) scalar.

Description: The instruction causes the currently executing instruction within the process specified by operand 1 to be terminated according to the options specified by operand 2. Operand 1 is a system pointer addressing the process control space associated with the process for which the currently executing instruction is to be terminated. If the process control space specified by operand 1 is not associated with a process, then an exception is signaled.

Operand 2 is a character (3) scalar specifying the termination options. The format of the character scalar is:

• Termination type (Must be hex 00)	Char(1)
• Reserved	Char(2)

The termination type of hex 00 specifies that only instructions which require a relatively long time for their execution are to be terminated. These instructions are the following:

Activate cursor
Apply journaled changed
Copy data space entries
Create program
Create data space index
Data base maintenance
Insert sequential data space entries
Modify data space index attributes
Request I/O
Retrieve journal entries
Retrieve sequential data space entries
Set cursor

These instructions are only subject to termination at certain points within their execution. If the instruction is beyond the last point at which it can be terminated when this instruction is executed, the target instruction will execute to completion. Additionally, if the specified process is not currently executing one of these instructions, the termination request is ignored.

Termination of the currently executing instruction in the specified process results in the signaling of an exception by the terminated instruction to indicate that it terminated rather than completed execution.

Substring operand references which allow for a null substring reference (a length value of zero) may not be specified for this instruction.

It is anticipated that this instruction will be used in cases where a target process is to have the current instruction processing terminated, if necessary, to provide for timely processing of an event sent to the process. Without the capability to terminate certain instructions, processing of the event could be delayed until instruction completion, which in extreme cases, takes hours. The target process must be prepared to provide for this case of instruction termination, since the function of the instruction will not have been performed if it is terminated.

Authorization Required

- Process control special authorization
  - If not initiating process
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution

Events

0002 Authorization
0101 Object violation

000C Machine resources
0201 Machine auxiliary storage exceeded

000D Machine status
0101 Machine check

0010 Process
0601 Exception signaled to process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference
0201 Object location reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
04 Unauthorized for process control	X		
10 Damage Encountered			
04 System object damage	X		X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state			X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer address invalid object	X		
28 Process State			
01 Process ineligible for operation	X		
02 Process control space not associated with a process	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
03 Scalar value invalid		X	

TERMINATE PROCESS (TERMPR)

Op Code (Hex)	Operand 1	Operand 2
0332	Process control space	Termination option

Operand 1: System pointer or null.

Operand 2: Character(3) scalar (fixed-length).

Description: The instruction causes the termination of one or more processes. Because this instruction may require one process to act upon another process, a portion of the function is controlled by the issuing process, and the remainder of the function is controlled by the target process. When control is returned to the issuing process, the function may not have been performed in its entirety.

Operand 1 identifies the process that is to be terminated. Operand 1 can be a system pointer that addresses the process control space associated with the process to be terminated, or it can be null. If operand 1 is null, the process issuing the instruction is considered the process to be terminated.

Operand 2 is a character(3) scalar specifying the termination option. The format of the termination option is as follows:

• Termination specifications	Char(1)
– Termination action	Bit 0
0 = Initiate process destruction against the subordinate processes of the designated process.	
1 = Initiate process destruction against the designated process and all subordinate processes.	
– Conditional termination action	Bit 1
0 = Place process in termination phase if not already there. If the process is in the termination phase, the request is ignored (conditional).	
1 = Place process in termination phase if not already there. If in termination phase, immediate process destruction results (unconditional).	
– Reserved (binary 0)	Bits 2-7
• Termination code	Char(2)

A process can apply the terminate function to any process in the machine except for a superordinate process in whose domain the issuing process resides.

No authorization is required in the following circumstances:

- The process issuing the instruction initiated the process identified by operand 1.

- The process referenced by operand 1 is the process issuing the instruction.

In all other cases, the process issuing the instruction must be currently governed by a user profile having the process control special authorization. The user profile can be either the process's assigned user profile or a currently adopted user profile.

The key element that dictates the function of Terminate Process instruction is the subject process's process status indicators. This attribute of a process supplies information relative to the current state of the process and the actions occurring both within and without that have caused the process to be in the current state. These indicators contain the following major categories of information:

- Process states
  - External existence state
    a. Active
    b. Suspended
  - Internal processing phase
    a. Initiation phase
    b. Problem phase
    c. Termination phase

- Process interrupt status

- Process initial internal termination status

- Process initial external termination status

- Process final termination status

The process initial internal termination status is generated when a process takes termination action upon itself. For example, this status is generated when the Terminate Process instruction is executed with the process itself as the target. The process and its subordinate processes are then placed in the termination phase. A subprocess's process initial external termination status is generated, and it contains the same information supplied in the superordinate process's process initial internal termination status.

Subprocesses are not placed in the termination phase when the superordinate process enters termination phase as a result of a RETURN from the first invocation in the initiation or problem phase, or when it is returned as a result of an unhandled exception.

The process initial external termination status is generated when action is taken against the process by another process; for example, this status is generated when the Terminate Process instruction is issued by one process with another process as the target. This action conditionally places the process in the termination phase if the process is not already in that phase. The status is also placed in the subprocess's process initial external termination status.

The process is placed in the termination phase only if the termination phase option process attribute is set to enter the termination phase. The process can be conditionally removed from the termination phase based on the conditional termination action option. This option allows orderly return from a termination phase. An unconditional termination request results in an immediate process destruction if the process is already in the termination phase. A conditional request results in the instruction not being performed.

The process final termination status either is generated internally by the process's own termination action while in the termination phase or is supplied by another process while the target process is in the termination phase.

All three termination status fields are supplied as event-related data for the process terminate event.

When the Terminate Process instruction is executed by a process itself, and the process is in the initiation or problem phase, the machine stores the termination status in the process initial internal termination status. This status field is also filled in when returning from the first invocation in the problem phase and upon an exception not being handled by the process. The initial internal termination status is propagated to any established subprocess's initial external status indicators only during Terminate Process instruction action. Refer to the Materialize Process Attributes instruction, earlier in this chapter, for the detailed format of the attribute. The following information is recorded:

- Initial internal termination reason
  - Return from first invocation in problem phase
  - Return from first invocation in initiation phase and no first program phase program supplied
  - Terminate Process instruction issued by process itself
  - Exception not handled by the process

- Initial internal termination code

The process's internal processing phase attribute is set to indicate that the process is in the termination phase if the process termination phase option specifies enter termination phase. If the process's current attributes indicate that a termination phase program is to be given control, the process status indicators are set to the active-termination state, an activation of the designated program is established (if not already existing), an invocation is created, and control is transferred to the program's entry point. All program invocations are destroyed prior to giving the process termination phase program control. If no termination phase program is defined, the machine sets the final termination status field equal to the initial internal termination status field. This indicates that a termination phase program was not executed and the instruction proceeds immediately with destruction of the process.

If a Return External instruction is executed in the highest level invocation in the problem phase or an exception is not handled in either the problem phase or initiation phase, the same functions are applied as for the explicitly specified terminate instruction described in the previous paragraph. When control is returned from the highest invocation, the initial internal termination code is set to 0 or to the exception type for an exception that is not handled.

When the Terminate Process instruction is issued by a process to itself while it is in the termination phase, the instruction stores information relative to the termination in the process's final termination status field. All subprocesses are destroyed regardless of their current internal processing phase.

The stored information is contained in the process status indicators attribute materialized through the Materialize Process Attributes instruction. The information made available includes:

- Final termination reason
  - Return from first invocation
  - Terminate Process instruction issued by the process itself
  - Terminate Process instruction issued to this process by another process
  - Exception not handled by the process

- Final termination code

The machine immediately proceeds with the destruction of the process.

If the Terminate Process instruction is executed in an external process, the target process's initial external termination code is supplied by the instruction's termination option. If the target process is in the initiation or problem phase, termination action proceeds as described earlier; that is, the process internal processing phase is set to the termination phase, and the termination phase program is invoked.

If the initial external termination status had been previously supplied; that is, the process has already been the target of an external Terminate Process instruction, immediate process destruction takes place with the later termination option recorded as the final termination status. If the status was not previously supplied, then it is recorded in the initial external termination status and the process is placed in the termination phase.

The following information is recorded in the initial external termination status:

- Initial external termination reason
  - Terminate Process instruction issued explicitly to the process from another process
  - Terminate Process instruction issued to superordinate process of this process

- Initial external termination code

If the process returns from the highest invocation or receives an exception that is not handled during the termination phase and if the process has active or suspended subprocesses, the process and its subprocesses are destroyed.

The same action occurs if the process that has active or suspended subprocesses attempts to terminate itself during the termination phase.

The functions performed by the instruction are determined by the setting of the termination action operand field in the Terminate Process instruction and are described in the following paragraphs.

The first option (binary 0) specifies that all the designated process's subordinates are to be destroyed. No exception is signaled if there are no subordinate processes.

The second option (binary 1) specifies that the designated process and all subordinates are to be destroyed.

If a process phase is terminated and programs are currently invoked, these invocations are terminated. If conditional termination was specified or the process was not in termination phase for an unconditional termination, then invocation exit programs set for the terminated invocations will be allowed to run.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

For more information about the invocation exit program, see *Program Execution* in the *Functional Concepts Manual.*

Authorization Required

- Process control special authorization
  - If not initiating process or process not terminating itself

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0202 Process terminated (to initiating process)
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0011 Program
  0301 Invocation exit bypassed due to a RTNEXCP or a SIGEXCP instruction
  0302 Invocation exit bypassed due to process termination
  0304 Failure to invoke program (invocation exit or exception handler)

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	4	Other
06 Addressing					
01 Space addressing violation	X	X			
02 Boundary alignment	X	X			
03 Range	X	X			
06 Optimized addressability invalid	X	X			
08 Argument/Parameter					
01 Parameter reference violation	X	X			
0A Authorization					
01 Unauthorized for operation	X				
04 Unauthorized for process control	X				
10 Damage Encountered					
04 System object damage state	X				X
44 Partial system object damage					X
1A Lock State					
01 Invalid lock state					X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X			
02 Object destroyed	X	X			
03 Object suspended	X	X			
24 Pointer Specification					
01 Pointer does not exist	X	X			
02 Pointer type invalid	X	X			
03 Pointer addressing invalid object	X				
28 Process State					
01 Process ineligible for operation	X				
02 Process control space not associated with a process	X				
2A Program Creation					
06 Invalid operand type	X	X			
07 Invalid operand attribute	X	X			
08 Invalid operand value range	X	X			
0C Invalid operand ODT reference	X	X			
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
03 Scalar value invalid		X			

WAIT ON TIME (WAITTIME)

Op Code (Hex)	Operand 1
0349	Wait template

Operand 1: Character(16) scalar.

Description: This instruction causes the process to wait for a specified time interval. The current process is placed in wait state for the amount of time specified by the wait template in accordance with the specified wait options.

The format of the wait template for operand 1 is:

• Wait time interval	Char(8)
• Wait options	Char(2)
– Access state control for entering wait 0 = Do not modify access state 1 = Modify access state	Bit 0
– Access state control for leaving wait 0 = Do not modify access state 1 = Modify access state	Bit 1
– MPL (multiprogramming level) control during wait 0 = Do not remain in current MPL set 1 = Remain in current MPL set	Bit 2
– Reserved	Bits 3-15
• Reserved	Char(6)

The format of the wait time interval value is that of a 64-bit unsigned binary value where bit 41 is equal to 1024 microseconds, assuming the bits are numbered from 0 to 63.

The access state control options control whether the process access group (PAG) will be explicitly transferred between main and auxiliary storage when entering and leaving a wait as a result of execution of this instruction. Specification of modify access state requests that the PAG be purged from main to auxiliary storage for entering a wait and requests that the PAG be transferred from auxiliary to main storage for leaving a wait. Specification of do not modify access state requests that the PAG not be explicitly transferred between main and auxiliary storage as a result of executing this instruction.

The access state of the PAG is modified when entering the wait if the process' instruction wait initiation access state control attribute specifies allow access state modification, if the access state control for entering wait option specifies modify access state, and if the MPL control during wait option specifies do not remain in current MPL set.

The MPL control during wait option controls whether the process will be removed from the current MPL (multiprogramming level) set or remain in the current MPL set when the process enters a wait as a result of executing this instruction.

When the MPL control during wait option specifies remain in current MPL set and the access state control for entering wait option specifies do not modify access state, the process will remain in the current MPL set for a maximum of 2 seconds. After 2 seconds, the process will automatically be removed from the current MPL set. The automatic removal does not change or affect the total wait time specified for the process in the wait time interval.

While the process is in wait state it may be interrupted for events unless the process is masked.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resources
  0201 Machine auxiliary storage exceeded

000D Machine status
  0101 Machine check

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X				
02 Boundary alignment	X				
03 Range	X				
06 Optimized addressability invalid	X				
08 Argument/Parameter					
01 Parameter reference violation	X				
10 Damage Encountered					
04 System object damage					X
44 Partial system object damage					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
02 Object destroyed	X				
03 Object suspended	X				
24 Pointer Specification					
01 Pointer does not exist	X				
02 Pointer type invalid	X				
2A Program Creation					
06 Invalid operand type	X				
07 Invalid operand attribute	X				
0A Invalid operand length	X				
0C Invalid operand ODT reference	X				
0D Reserved bits are not zero	X				X
32 Scalar Specification					
01 Scalar type invalid	X				
02 Scalar attributes invalid	X				
03 Scalar value invalid	X				

Chapter 12. Queue Management Instructions

This chapter describes the instructions used for queue management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CREATE QUEUE (CRTQ)

Op Code (Hex)	**Operand 1**	**Operand 2**
0316	Address-ability to created queue	Queue template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: The instruction creates a queue based on the parameters specified in the queue template (operand 2) and returns a system pointer in the pointer object (operand 1) that addresses the created object.

The queue template (operand 2) has the following format:

• Template size specification	Char(8)
– Number of bytes provided	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context 0 = Addressability is not inserted in context 1 = Addressability is inserted in context	Bit 2
– Access group 0 = Member of access group is not created 1 = Member of access group is created	Bit 3
– Reserved (binary 0)	Bits 4-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Access group	System pointer

• Queue attributes	Char(1)
– Message content	Bit 0
0 = Contains scalar data only	
1 = Contains pointers and scalar data	
– Queue type	Bits 1-2
00 = Keyed	
01 = Last in first out (LIFO)	
10 = First in first out (FIFO)	
11 = Reserved	
– Queue overflow action	Bit 3
0 = Signal exception	
1 = Extend queue	
– Reserved (binary 0)	Bits 4-7
• Maximum number of messages	Bin(4)
• Current number of messages	Bin(4)*
• Extension value	Bin(4)
• Key length (maximum key length = 256)	Bin(2)
• Maximum size of messages to be enqueued (The maximum allowable size of a queue message is 64 000 bytes.)	Bin(4)

Note: The values of the parameters annotated with an asterisk (*) are ignored by this instruction.

The template identified by operand 2 must be 16-byte aligned.

If the created object is permanent, it is owned by the user profile governing process execution. The owning user profile is implicitly assigned all private authority states for the object. The storage occupied by the created object is charged to this owning user profile. If the created object is temporary, no owning user profile exists, and all authority states are assigned as public. Storage occupied by the created context is charged to the creating process.

The object identification specifies the symbolic name that identifies the queue within the machine. A type code of hex 0A is implicitly supplied by the machine. The object identification is used to identify the object on materialize instructions as well as to locate the object in a context that addresses the object.

The existence attribute specifies that the queue is to be created as temporary. A temporary queue, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated.

A space may be associated with the created object. The space may be fixed or variable in size. The initial allocation is as specified in the size of space entry. The machine allocates a space of at least the size specified; the actual size allocated depends on an algorithm defined by a specific implementation. Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended, this byte value is also used to initialize the new allocation. If no space is allocated, this byte value is ignored.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must be a system pointer that identifies a context where addressability to the newly created queue is to be placed. If the initial context indicates that addressability is not to be inserted into a context, the context entry is ignored.

If the access group creation attribute entry indicates that the object is to be created in an access group, the access group entry must be a system pointer that identifies an access group in which the object is to be created. Only temporary queues may be created in an access group. If the object is not to be created in the access group, the access group entry is ignored.

The message content attribute specifies whether the messages to be enqueued will contain pointers and scalar data, or scalar data only. If the messages are to contain pointers the message text operand on Enqueue and Dequeue instructions must be aligned on 16-byte boundaries.

The queue type parameter establishes the basic sequence in which messages are dequeued from the queue.

The queue overflow action parameter establishes the machine action when the number of messages resident on the queue (enqueued and not yet dequeued) exceeds the current maximum capacity of the queue. This value is initially established by the value specified in the maximum number of messages parameter. The queue message limit exceeded exception and the queue message limit exceeded event are signaled when the number of resident messages exceeds this parameter unless the extend queue option is specified. When the extend queue option is specified, the value of the maximum number of messages parameter is increased by the amount specified by the extension value parameter each time the number of enqueued messages exceeds the current value of the maximum number of messages parameter. When the extend queue option is specified, the extension value parameter must contain a value greater than 0. If the signal exception option is specified, the extension value parameter is ignored and the maximum number of messages parameter must contain a value greater than zero.

The current number of messages entry is reported in the materialization of the queue's attribute, and the value of the entry is ignored in the creation template.

The key length parameter establishes the size of the queue's key. If the queue type parameter keyed is specified, the value must be greater than 0. The key can contain pointers, but the pointers are considered to be scalar data when they are placed on the queue by an Enqueue instruction. If the queue type parameter specifies LIFO or FIFO, the key length can be equal to or greater than 0; however, the queue is not treated as a keyed queue.

The size of all messages to be enqueued is established by the maximum size of messages to be enqueued parameter. The Enqueue instruction may specify a size (in the message prefix) that is greater than this value, but the message is truncated to this length. The maximum size of messages to be enqueued parameter must have a value of 0 or greater, up to a maximum value of 64 000 bytes. The maximum size of a queue, excluding its associated space, cannot exceed 16 megabytes. This value includes machine overhead associated with the queue.

Authorization Required

- Insert
  - User profile of creating process
  - Context identified by operand 2
- Retrieve
  - Contexts referenced for address resolution
- Object control
  - Operand 1 if replace option requested

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Access group identified by operand 2
  - Context identified by operand 2
  - User profile of creating process
- Object control
  - Operand 1 if replace option requested

Events

0002 Authorization
 0101 Object authorization violation

000C Machine resource
 0201 Machine auxiliary storage threshold exceeded
 0501 Machine address threshold exceeded

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	X
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded	X		
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute		X	
08 Invalid operand value range		X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
38 Template Specification			
01 Template value invalid		X	

DEQUEUE (DEQ, DEQB, or DEQI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3	Operand 4-5
1033		Message prefix	Message text	Queue	
1C33	Branch options				Branch target
1833	Indicator options				Indicator target

Operand 1: Character variable scalar (fixed-length).

Operand 2: Space pointer.

Operand 3: System pointer.

Operand 4-5:

- *Branch Form*—Branch point, instruction pointer, relative instruction number, or absolute instruction number.
- *Indicator Form*—Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

If the branch or indicator option is indicated in the op code, the extender field is required along with one or two branch operands (for branch option) or one or two indicator operands (for indicator option). See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The instruction retrieves a queue message based on the queue type (FIFO, LIFO, or keyed) specified during the queue's creation. If the queue was created with the keyed option, messages can be retrieved by any of the following relationships between an enqueued message key and a selection key specified in operand 1 of the Dequeue instruction: ≠, >, <, ≤, and ≥. If the queue was created with either the LIFO or FIFO attribute, then only the next message can be retrieved from the queue.

If a message is not found that satisfies the dequeue selection criterion and the branch or options are not specified, the process is put into the wait state until a message arrives to satisfy the dequeue or until the dequeue wait time-out expires. If branch or indicator options are specified, the process is not placed in the dequeue wait state and either the control flow is altered according to the branch options, or indicator values are set based on the presence or absence of a message to be dequeued.

A nonzero dequeue wait time-out value overrides any dequeue wait time-out value specified as the current process attribute. A zero wait time-out value causes the wait time-out value to be taken from the current process attribute. If all wait time-out values are 0 (from the Dequeue instruction and the current process attribute), an immediate wait time-out exception is signaled. The bits in this field are numbered from 0 to 63, and bit 41 is defined as 1024 microseconds. The maximum wait time-out interval allowed is a value equal to (2^{48} - 1) microseconds. Any value that indicates more time than the maximum wait time-out causes the maximum wait time-out to be used.

A message is dequeued from the queue specified by operand 3. The criteria for message selection are given in the message prefix specified by operand 1. The message text is returned in the space specified by operand 2, and the message prefix is returned in the scalar specified by operand 1. The size of the message text retrieved is returned in the message prefix. The size of the message text can be less than or equal to the maximum size of message specified when the queue was created. When dequeuing from a keyed queue, the length of the search key field and the length of the message key field (in the message key prefix specified in operand 1) are determined implicitly by the attributes of the queue being accessed. If the message text on the queue contains pointers, the message text operand must be 16-byte aligned. Improper alignment results in an exception being signaled. The format of the message prefix is as follows:

• Timestamp of enqueue of message	Char(8)**
• Dequeue wait time-out value (ignored if branch options specified)	Char(8)*
• Size of message dequeued (The maximum allowable size of a queue message is 65 000 bytes.)	Bin(4)**
• Access state modification option indicator and message selection criteria	Char(1)*
– Access state modification option	Bit 0-1*
When entering Dequeue wait 0 = Access state is not modified 1 = Access state is modified	Bit 0*
When leaving Dequeue wait 0 = Access state is not modified 1 = Access state is modified	Bit 1*
– Multiprogramming level option 0 = Leave current MPL set at Dequeue wait 1 = Remain in current MPL set at Dequeue wait	Bit 2*
– Time-out option 0 = Wait for specified time, then signal time-out exception 1 = Wait indefinitely	Bit 3*
– Actual key to input key relationship (for keyed queue) 0010: Greater than 0100: Less than 0110: Not equal 1000: Equal 1010: Greater than or equal 1100: Less than or equal	Bits 4-7*
• Search key (ignored for FIFO/LIFO queues but must be present for FIFO/LIFO queues with nonzero key length values)	Char(key length)*
• Message key	Char(key length)**

Note: Fields shown here with one asterisk indicate input to the instruction, and fields shown here with two asterisks are returned by the machine.

The access state of the process access group is modified when a Dequeue instruction results in a wait and the following conditions exist: the process' instruction wait initiation access state control attribute specifies allow access state modification, the dequeue access state modification option specifies modify access state, and the multiprogramming level option specifies leave MPL set during wait.

The process will remain in the current MPL set for a maximum of two seconds when a Dequeue instruction results in a wait if the multiprogramming level option specifies remain in current MPL set at Dequeue wait and the access state modification when entering Dequeue wait option specifies do not modify access state. After two seconds, the process will automatically be removed from the current MPL set. The automatic removal does not change or affect the total wait time specified for the process by the Dequeue wait time-out value.

Operand 3 is a system pointer addressing the queue from which the message is to be dequeued.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Message dequeued (equal), message not dequeued (not equal).

Authorization Required

- Retrieve
  - Operand 3
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation			X	X
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state			X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X		X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object			X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X			
08 Invalid operand value range	X			
09 Invalid branch target operand				X
0A Invalid operand length	X			
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid	X			
3A Wait Time-out				
01 Dequeue				X

DESTROY QUEUE (DESQ)

Op Code (Hex)	Operand 1
0325	Queue

Operand 1: System pointer.

Description: This instruction destroys the specified queue and all currently enqueued messages. All processes currently in the dequeue wait state for this queue are removed from the dequeue wait state and an object destroyed exception is signaled to the waiting processes. Addressability is deleted from the context (if any) that addresses the object. The system pointer identified by operand 1 is not modified by the instruction, and a subsequent reference to the destroyed queue through the pointer results in an object destroyed exception.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Object control
  - Operand 1

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Context which addresses operand 1
  - User profile which owns operand 1
  - Access group which contains operand 1

- Object control
  - Operand 1

Events

0002 Authorization
- 0101 Object authorization violation

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	X
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

ENQUEUE (ENQ)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
036B	Queue	Message prefix	Message text

Operand 1: System pointer.

Operand 2: Character scalar.

Operand 3: Space pointer.

Description: A message is enqueued according to the queue type attribute specified during the queue's creation.

If keyed sequence is specified, enqueued messages are sequenced in ascending binary collating order according to the key value. If a message to be enqueued has a key value equal to an existing enqueued key value, the message being added is enqueued following the existing message.

If the queue was defined with either last in, first out (LIFO) or first in, first out (FIFO) sequencing, then enqueued messages are ordered chronologically with the latest enqueued message being either first on the queue or last on the queue, respectively. A key can be provided and associated with messages enqueued in a LIFO or FIFO queue; however, the key does not establish a message's position in the queue. The key can contain pointers, but the pointers are not considered to be pointers when they are placed on the queue by an Enqueue instruction.

Operand 1 specifies the queue to which a message is to be enqueued. Operand 2 specifies the message prefix, and operand 3 specifies the message text.

The format of the message prefix is as follows:

• Size of message to be enqueued	Bin(4)*
• Enqueue key value (Ignored for FIFO/LIFO queues with key lengths equal to 0. Must be present for all other queues.)	Char(key length)*

Note: Fields annotated with an asterisk indicate input to the instruction.

The size of the message to be enqueued is supplied to inform the machine of the number of bytes in the space that are to be considered message text. The size of the message is then considered the lesser of the size of the message to be enqueued attribute and the maximum message size specified on queue creation. The message text can contain pointers. When pointers are in message text, the operand 3 space pointer must be 16-byte aligned. Improper alignment will result in an exception being signaled.

If the enqueued message causes the number of messages to exceed the maximum number of messages attribute of the queue, one of the following occurs:

- If the queue is not extendable, the queue message limit exceeded exception and the queue message limit exceeded event are signaled. The message is not enqueued.

- If the queue is extendable, the queue is implicitly extended by the extension value attribute. The message is enqueued. No exception is signaled, but the queue extended event is signaled.

The maximum allowable queue size, including all messages currently enqueued and the machine overhead, is 16 megabytes.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Insert
  - Operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0012 Queue
    0301 Queue message limit exceeded
    0401 Queue extended

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation	X			X
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state	X			X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded	X			X
04 Object storage limit exceeded	X			
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
26 Process Management				
02 Queue message limit exceeded	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2E Resource Control Limit				
01 User profile storage limit exceeded				X

MATERIALIZE QUEUE ATTRIBUTES (MATQAT)

Op Code (Hex)	Operand 1	Operand 2
0336	Receiver	Queue

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The attributes of the queue specified by operand 2 are materialized into the object specified by operand 1. The format of the materialized queue attributes must be aligned on a 16-byte multiple. The format is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context 0 = Addressability not in context 1 = Addressability in context	Bit 2
– Access group 0 = Not a member of access group 1 = Member of access group	Bit 3
– Reserved (binary 0)	Bits 4-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

- Performance class Char(4)
  - Space alignment Bit 0
    - 0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    - 1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.
  - Reserved (binary 0) Bits 1-4
  - Main storage pool selection Bit 5
    - 0 = Process default main storage pool is used for object.
    - 1 = Machine default main storage pool is used for object.
  - Reserved (binary 0) Bit 6
  - Block transfer on implicit access state modification Bit 7
    - 0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
    - 1 = Transfer the machine default storage transfer size. This value is 8 storage units.
  - Reserved (binary 0) Bits 8-31

- Reserved (binary 0) Char(7)

- Context System pointer

- Access group System pointer

- Queue attributes Char(1)
  - Message content Bit 0
    - 0 = Contains scalar data only
    - 1 = Contains pointers and scalar data
  - Queue type Bits 1-2
    - 00 = Keyed
    - 01 = Last in, first out
    - 10 = First in, first out
  - Queue overflow action Bit 3
    - 0 = Signal exception
    - 1 = Extend queue
  - Reserved (binary 0) Bits 4-7

- Current maximum number of messages Bin(4)

- Current number of messages enqueued Bin(4)

- Extension value Bin(4)

- Key length Bin(2)

- Maximum size of message to be enqueued Bin(4)

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled when the receiver contains insufficient area for the materialization.

Authorization Required

- Operational
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	X
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X		
08 Invalid operand value range	X		
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE QUEUE MESSAGES (MATQMSG)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
033B	Receiver	Queue	Message selection template

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(16) scalar.

Description: This instruction materializes selected messages on a queue. One or more messages on the queue specified by operand 2 is selected according to information provided in operand 3 and materialized into operand 1. The number of messages materialized and the amount of key and message text data materialized for each message is governed by the message selection template.

Note that the list of messages on a queue is a dynamic attribute and may be changing on a continual basis. The materialization of messages provided by this instruction is just a picture of the status of the queue at the point of interrogation by this instruction. As such, the actual status of the queue may differ from that described in the materialization when subsequent instructions use the information in the template as a basis for operations against the queue.

Operand 1 specifies a space that is to receive the materialized attribute values.

Operand 2 is a system pointer identifying the queue from which the messages are to be materialized.

Operand 3 is a character (16) scalar specifying which messages are to be materialized.

The operand 1 space pointer must address a 16-byte boundary. The materialization template has the following format:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Materialization data	Char(4)
– Count of messages materialized	Bin(4)
• Queue data	Char(12)
– Count of messages on the queue	Bin(4)
– Maximum message size	Bin(4)
– Key size	Bin(4)
• Reserved	Char(8)
• Message data (repeated for each message)	Char(*)
– Message attributes	Char(16)
– – Message enqueue time	Char(8)
– – Message length	Bin(4)
– – Reserved	Char(4)
– Message key	Char(*)
– Message text	Char(*)

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The maximum message size and key size are values specified when the queue was created. If the queue is not a keyed queue, the value materialized for the key size is zero.

The length of the message key and message text fields is determined by values supplied in operand 3, message selection data. If the length supplied in operand 3 exceeds the actual data length, the remaining space will be padded with binary zeros.

The message selection template identified by operand 3 must be at least 16 bytes and must be on a 16-byte boundary. The format of the message selection template is as follows:

• Message selection	Char(2)
– Type	Bits 0-3
0001–All messages	
0010–First	
0100–Last	
1000–Keyed	
All other values are reserved	
– Key relationship (if needed)	Bits 4-7
0010–Greater than	
0100–Less than	
0110–Not equal	
1000–Equal	
1010–Greater than or equal	
1100–Less than or equal	
All other values are reserved	
– Reserved	Bits 8-15
• Lengths	Char(8)
– Number of key bytes to materialize	Bin(4)
– Number of message text bytes to materialize	Bin(4)
• Reserved	Char(6)
• Key (if needed)	Char(*)

The message selection type must not specify keyed if the queue was not created as a keyed queue.

Both of the fields specified under lengths must be zero or an integer multiple of 16. The maximum value allowed for the key length is 256. The maximum value allowed for the message text is 65536.

Authorization Required

- Retrieve
  - Operand 2
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X		
02 Boundary alignment	X	X	X		
03 Range	X	X	X		
06 Optimized addressability invalid	X	X	X		
08 Argument/Parameter					
01 Parameter reference violation	X	X	X		
0A Authorization					
01 Unauthorized for operation		X			
10 Damage Encountered					
04 System object damage		X			X
44 Partial system object damage					X
1A Lock State					
01 Invalid lock state				X	
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X		
02 Object destroyed	X	X	X		
03 Object suspended	X	X	X		
24 Pointer Specification					
01 Pointer does not exist	X	X	X		
02 Pointer type invalid	X	X	X		
03 Pointer addressing invalid object		X			
28 Process State					
02 Process control space not associated with a process		X			
2A Program Creation					
06 Invalid operand type	X	X	X		
07 Invalid operand attribute	X	X	X		
08 Invalid operand value range	X	X	X		
0A Invalid operand length			X		
0C Invalid operand ODT reference	X	X	X		
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
01 Scalar type invalid	X	X	X		
02 Scalar attributes invalid	X	X	X		
03 Scalar value invalid			X		
38 Template Specification					
03 Materialization length exception	X				

Chapter 13. Resource Management Instructions

This chapter describes the storage and resource management instructions. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CREATE ACCESS GROUP (CRTAG)

Op Code (Hex)	**Operand 1**	**Operand 2**
0366	Addressability to created access group	Access group template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: An access group with the attributes of the template identified by operand 2 is created, and a system pointer to the access group is returned in the pointer identified by operand 1.

The access group template specified by operand 2 must be 16-byte aligned and must have the following format:

• Template size specification	Char(8)*
– Number of bytes provided in template	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes	Bit 0
0 = Temporary (required)	
– Space attribute	Bit 1
0 = Fixed-length	
1 = Variable-length	
– Initial context	Bit 2
0 = Addressability is not inserted into context	
1 = Addressability is inserted into context	
– Reserved (binary 0)	Bits 3-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Context	System pointer

Note: The value associated with each entry shown here with an asterisk (*) is ignored.

The storage occupied by the created access group is charged to the creating process.

The object identification specifies the symbolic name that identifies the access group within the machine. A type code of hex 03 is implicitly supplied by the machine. The object identification is used to identify the access group on materialize instructions as well as to locate the access group in a context that addresses the access group.

The existence attribute specifies that the access group is to be created as temporary. An access group, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated. An access group can contain only other temporary objects and not another access group.

A space may be associated with the created access group. The space may be fixed or variable in size. The initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified; the actual size allocated depends on an algorithm defined by a specific implementation. A fixed size space entry of 0 causes no space to be allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended, the byte space entry value is also used to initialize the new allocation. If no space is allocated, this entry is ignored.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context in which addressability to the newly created object is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

The performance class parameter provides information that allows the machine to manage the access group with consideration for the overall performance objectives of operations involving the access group.

Access groups are implicitly extended by the machine to a size large enough to contain any objects inserted into them. The maximum size of an access group is 4 megabytes.

Authorization Required

- Insert
  - Context identified by operand 2

Lock Enforcement

- Modify
  - Context identified by operand 2

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
0E Context Operation			
01 Duplicate object identification			X
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
01 Template value invalid		X	

CREATE DUPLICATE OBJECT (CRTDOBJ)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0327	Addressability to new object	Create duplicate object template	Object to be duplicated

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: System pointer.

Description: A copy of the object identified by operand 3 is created. The object may be a cursor or a space.

The new object is identical to the source object except as modified by the creation template.

- A resolved pointer in the space portion of the source object that has an address to an interior element in the same space is not resolved to address the same functional address in the new version of the object; that is, pointers are not relocated.
- Any authorization established for the source object is not duplicated into the new object.
- A cursor addressed by the instruction is duplicated in its unactivated form. Any modifications that have been made to the cursor after it was originally created are not reflected in the new object.

A system pointer addressing the new object is returned in the pointer specified by operand 1.

The Create Duplicate Object instruction template specified by operand 2 must be aligned on a 16-byte boundary. The format is:

• Template size specification	Char(8)*
– Number of bytes provided	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context 0 = Addressability is not inserted in context 1 = Addressability is inserted in context	Bit 2
– Access group 0 = Member of access group is not created 1 = Member of access group is created	Bit 3
– Replace option 0 = Create as new object 1 = Replace existing object	Bit 4
– Return space size option 0 = Do not return space size 1 = Return actual space size in space size field	Bit 5
– Reserved (binary 0)	Bits 6-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)@
• Initial value of space	Char(1)

- Performance class Char(4)
  - Space alignment Bit 0
    - 0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    - 1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.
  - Reserved (binary 0) Bits 1-4
  - Main storage pool selection Bit 5
    - 0 = Process default main storage pool is used for object.
    - 1 = Machine default main storage pool is used for object.
  - Transient storage pool selection Bit 6
    - 0 = Default main storage pool (as specified for main storage pool selection)
    - 1 = Transient storage pool is used for object.
  - Block transfer on implicit access state modification Bit 7
    - 0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
    - 1 = Transfer the machine default storage transfer size. This value is 8 storage units.
  - Unit number Bits 8-15
  - Reserved (binary 0) Bits 16-31

- Reserved (binary 0) Char(7)

- Context System pointer

- Access group System pointer

Note: The value associated with each entry shown here with an asterisk (*) is ignored. The value associated with each entry shown here with an at sign (@) may be altered by the instruction.

If the created object is permanent, it is owned by the user profile governing process execution. The owning user profile is implicitly assigned all private authority states for the object. The storage occupied by the created object is charged to this owning user profile. If the created object is temporary, there is no owning user profile and all authority states are assigned as public. Storage occupied by the created context is charged to the creating process.

The object identification specifies the symbolic name that identifies the object within the machine. A type code identical to that of the source object is implicitly supplied by the machine. The object identification is used to identify the object on materialize instructions as well as to locate the object in a context that addresses the object.

The subtype code and name can be the same as or different from the object being duplicated. If both names and subtypes are the same, the new object cannot be placed in the same context as the original object. If the names or subtypes are different, the new object may be placed in the same context.

The existence attribute specifies whether the duplicate is to be a temporary object or a permanent object. The temporary and the permanent object creation attributes are supported for both the original object and the duplicate object.

A temporary object, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated. A permanent object exists in the machine until explicitly destroyed by the user.

A space may be associated with the created object. The space may be fixed or variable. The initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified; the actual size allocated depends on an algorithm defined by a specific implementation and is optionally returned in the space size entry. A fixed size space entry of 0 causes no space to be allocated.

The contents of the original space (if any) are copied into the duplicate space without modification. If the duplicate space is shorter than the original space, the information is truncated. If the duplicate space is longer, each byte beyond that copied from the original is initialized to a value specified by the initial value of space entry. When the space is extended, this byte value is also used to initialize the new allocation.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context in which addressability to the newly created object is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

If the access group creation attribute entry indicates that the object is to be created in an access group, the access group entry must be a system pointer that identifies the access group in which the object is to be created. Because access groups may only be created as temporary objects, the existence attribute entry must be temporary (bit 0 equals 0). If the object is not to be created in an access group, the access group entry is ignored.

Performance class parameters provide information that allows the machine to manage the duplicate object with consideration for the overall performance objectives of operations involving the duplicate object.

The unit number field, which can be specified for space objects only, indicates the auxiliary storage unit on which the space should be located if possible.

Operand 3 identifies a system pointer addressing the object to be duplicated.

Authorization Required

- Insert
  - User profile of creating process
  - Context referenced by operand 2

- Retrieve
  - Operand 3 (object to be duplicated)
  - Contexts referenced for address resolution

- Space authority
  - Operand 3 (only if the object to be duplicated has an associated space to be duplicated)

Lock Enforcement

- Materialize
  - Operand 3 (object to be duplicated)
  - Contexts referenced for address resolution

- Modify
  - User profile of creating process
  - Context referenced by operand 2
  - Access group referenced by operand 2

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	1	2	3	Other
02 Access Group				
01 Object ineligible for access group			X	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X	X	
0E Context Operation				
01 Duplicate object identification		X		
10 Damage Encountered				
04 System object damage state		X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state		X	X	
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
04 Object not eligible for operation			X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
03 Pointer addressing invalid object		X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2E Resource Control Limit				
01 User profile storage limit exceeded		X		
38 Template Specification				
01 Template value invalid		X		

DESTROY ACCESS GROUP (DESAG)

Op Code (Hex)	Operand 1
0351	Access group

Operand 1: System pointer.

Description: The access group identified by the system pointer (operand 1) is destroyed, and addressability is deleted from any context that addresses the access group. The system pointer is not modified. Any attempted reference to the destroyed access group through the pointer causes the object destroyed exception to be signaled.

If objects exist within the designated access group, the access group is not destroyed, and an object not eligible for destruction exception is signaled.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Context that addresses access group
- Object control
  - Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attributes	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

ENSURE OBJECT (ENSOBJ)

Op Code (Hex)	Operand 1
0381	Object to be ensured

Operand 1: System pointer.

Description: The object identified by operand 1 is protected from volatile storage loss. The machine ensures that any changes made to the specified object are recorded on nonvolatile storage media. The access state of the object is not changed by this instruction. If operand 1 addresses a temporary object, no operation is performed because temporary objects are not preserved during a machine failure. No exception is signaled if temporary objects are referenced.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0401 Journal space attached to a journal port became unusable

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attributes	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

MATERIALIZE ACCESS GROUP ATTRIBUTES (MATAGAT)

Op Code (Hex)	Operand 1	Operand 2
03A2	Receiver	Access group

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The attributes of the access group and the identification of objects currently contained in the access group are materialized into the receiving object specified by operand 1.

The materialization must be aligned on a 16-byte boundary. The format is:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temporary 1 = Reserved	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Context 0 = Addressability not in context 1 = Addressability in context	Bit 2
– Reserved (binary 0)	Bits 3-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.	
– Reserved (binary 0)	Bits 1-4
– Default main storage pool	Bit 5
0 = Process main storage pool is used for this object.	
1 = Machine default main storage pool is used for this object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Minimum storage transfer size for this object is transferred. This value is 1 storage unit.	
1 = Machine default storage transfer size is transferred. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Reserved (binary 0)	Char(16)
• Access group size	Bin(4)
• Reserved (binary 0)	Bin(4)
• Number of objects in the access group	Bin(4)
• Reserved (binary 0)	Char(4)
• Access group object system pointer (repeated for each object currently contained in the access group)	System pointer

The receiver space contains the access group's attributes (as defined by the Create Access Group instruction), the current status of the access group, and a system pointer to each object assigned to the access group.

The access group size represents the total amount of space that has been allocated to the access group. The amount of available space represents the amount of space that is available in the access group for additional objects.

There is one access group object system pointer for each object currently assigned to the access group. The authorization field within each system pointer is not set.

Authorization Required

- Retrieve
  - Operand 2
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE RESOURCE MANAGEMENT DATA (MATRMD)

Op Code (Hex)	Operand 1	Operand 2
0352	Receiver	Control data

Operand 1: Space pointer.

Operand 2: Character(8) scalar (fixed-length).

Description: The data items requested by operand 2 are materialized into the receiving object specified by operand 1. Operand 2 is an 8-byte character scalar. The first byte identifies the generic type of information being materialized, and the remaining 7 bytes further qualify the information desired.

Operand 1 contains the materialization and has the following format:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Time of day	Char(8)
• Resource management data	Char(*)

The remainder of the materialization depends on operand 2 and on the machine implementation.

The following values are allowed for operand 2:

• Selection option	Char(1)

Hex 01=Materialize process utilization data
Hex 02=Materialize auxiliary storage information
Hex 03=Materialize extended storage management counters
Hex 04=Materialize storage transient pool information
Hex 05=Materialize storage pool information
Hex 06=Storage management counters
Hex 07=Materialize extended storage pool information
Hex 08=Materialize machine address threshold data
Hex 0A=Materialize MPL control information
Hex 0B=Materialize machine reserved storage pool information

• Reserved (binary 0)	Char(7)

The following defines the formats and values associated with each of the above materializations of resource management data.

Processor Utilization (Hex 01):

• Processor time since IPL (initial program load)	Char(8)

Processor time since IPL is the total amount of processor time used, both by instruction processes and internal machine functions, since IPL. The significance of bits within the field is the same as that defined for the time-of-day clock.

Auxiliary Storage Information (Hex 02):

• Number of auxiliary storage units	Bin(2)
• Auxiliary storage capacity	Bin(8)
• Auxiliary storage space available	Bin(8)
• Auxiliary storage event threshold	Bin(8)
• Auxiliary storage control flags	Char(1)
– Error logging control flag	Bit 1
– Reserved	Bit 3
• Reserved	Char(5)
• Auxiliary storage unit utilization (repeated once for each auxiliary storage unit)	Char(64)
– Device type	Bin(2)
– Reserved	Char(1)
– Unit number	Bin(1)
– Reserved	Char(4)
– Capacity	Bin(8)
– Space available	Bin(8)
– Device dependent information	Char(40)
Bytes transferred to main storage	Bin(4)
Bytes transferred from main storage	Bin(4)
Requests for data transfer to main storage	Bin(4)
Requests for data transfer from main storage	Bin(4)
Reserved	Char(24)

Number of auxiliary units is the number of logical and physical devices that comprise the secondary store.

Auxiliary storage capacity is the total number of bytes of auxiliary storage attached to the machine.

Auxiliary storage space available is the number of bytes of space on secondary storage available for allocation; that is, not currently assigned to objects or internal machine functions.

Auxiliary storage event threshold is a number which, should it exceed secondary storage space available, will cause the event secondary storage threshold exceeded to be signaled. When the event is signaled, the machine resets this value to 0.

Error logging control flag bit, when set to 1, specifies that any temporary errors subject to threshold control are logged on every occurrence. When set to 0, such errors are logged only when the device specific thresholds are reached.

Auxiliary storage unit utilization data is repeated once for each logical device of the auxiliary storage. The relationship of logical to physical devices, and portions of the materialized utilization data, are device-dependent. Data is associated with a device by virtue of its logical position on the array.

Extended Storage Management Counters (Hex 03):

• Access pending	Bin(2)
• Storage pool delays	Bin(2)
• Directory look-up operations	Bin(4)
• Directory page faults	Bin(4)
• Access group member page faults	Bin(4)
• Microcode page faults	Bin(4)
• Microtask read operations	Bin(4)
• Microtask write operations	Bin(4)
• Reserved	Bin(4)

The definition of the fields materialized for this option is the same as that provided under the storage management counters option, hex 06. The extended fields under this option allow for materialization of larger values.

Storage Transient Pool Information (Hex 04):

• Storage pool to be used for the transient pool	Bin(2)

The pool number materialized is the number of the main storage pool, which is being used as the transient storage pool. A value of 0 indicates that the transient pool attribute is being ignored.

Storage Pool Information (Hex 05):

• Machine minimum transfer size	Bin(2)
• Maximum number of pools	Bin(2)
• Current number of pools	Bin(2)
• Main storage size	Bin(2)
• Minimum size–pool 1	Bin(2)
• Reserved (binary 0)	Char(6)
• Individual main storage pool information (repeated once for each pool, up to the current number of pools)	Char(16)
– Pool size	Bin(2)
– Pool maintenance	Bin(2)
– Process interruptions (data base)	Bin(2)
– Process interruptions (nondata base)	Bin(2)
– Data transferred to pool (data base)	Bin(4)
– Data transferred to pool (nondata base)	Bin(4)

Machine minimum transfer size is the smallest number of bytes that may be transferred as a block to or from main storage.

Maximum number of pools is the maximum number of storage pools into which main storage may be partitioned. These pools will be assigned the logical identification beginning with 1 and continuing to the maximum number of pools.

Current number of pools is a user-specified value for the number of storage pools the user wishes to utilize. These are assumed to be numbered from 1 to the number specified. This number is fixed by the machine to be equal to the maximum number of pools.

Main storage size is the amount of main storage, in units equal to the machine minimum transfer size, which may be apportioned among main storage pools.

Minimum size–Pool 1 is the amount of main storage, in units equal to the machine minimum transfer size, which must remain in pool 1. This amount is machine and configuration dependent.

Individual main storage pool information is data in an array that is associated with a main storage pool by virtue of its ordinal position within the array. In the descriptions below, data base refers to all other data, including internal machine fields. Pool size, pool maintenance, and data transferred information is expressed in units equal to the machine minimum transfer size described above.

Pool size is the amount of main storage assigned to the pool.

Pool maintenance is the amount of data written from a pool to secondary storage by the machine to satisfy demand for resources from the pool. It does not represent total transfers from the pool to secondary storage, but rather is an indication of machine overhead required to provide primary storage within a pool to requesting processes.

Process interruptions (data base and nondata base) is the total number of interruptions to processes (not necessarily assigned to this pool) which were required to transfer data into the pool to permit instruction execution.

Data transferred to pool (data base and nondata base) is the amount of data transferred from auxiliary storage to the pool to permit instruction execution and as a consequence of set access state, implicit access group movement, and internal machine actions.

Storage Management Counters (Hex 06):

• Access pending	Bin(2)
• Storage pool delays	Bin(2)
• Directory look-up operations	Bin(2)
• Directory page faults	Bin(2)
• Access group member page faults	Bin(2)
• Microcode page faults	Bin(2)
• Microtask read operations	Bin(2)
• Microtask write operations	Bin(2)

Access pending is a count of the number of times that a paging request must wait for the completion of a different request for the same page.

Storage pool delays is a count of the number of times that processes have been momentarily delayed by the unavailability of a main storage frame in the proper pool.

Directory look-up operations is a count of the number of times that auxiliary storage directories were interrogated, exclusive of storage allocation or deallocation.

Directory page faults is a count of the number of times that a page of the auxiliary storage directory was transferred to main storage, to perform either a look-up or an allocation operation.

Access group member page faults is a count of the number of times that a page of an object contained in an access group was transferred to main storage independently of the containing access group. This occurs when the containing access group has been purged or because portions of the containing access group have been displaced from main storage.

Microcode page faults is a count of the number of times a page of microcode was transferred to main storage.

Microtask read operations is a count of the number of transfers of one or more pages of data from auxiliary main storage on behalf of a microtask rather than a process.

Microtask write operations is a count of the number of transfers of one or more pages of data from main storage to auxiliary storage on behalf of a microtask, rather than a process.

Extended Storage Pool Information (Hex 07):

• Machine minimum transfer size	Bin(2)
• Maximum number of pools	Bin(2)
• Current number of pools	Bin(2)
• Main storage size	Bin(2)
• Minimum size—pool 1	Bin(2)
• Reserved (binary 0)	Char(6)
• Individual main storage pool information (repeated once for each pool, up to the current number of pools)	Char(32)
– Pool size	Bin(2)
– Pool maintenance	Bin(4)
– Process interruptions (data base)	Bin(4)
– Process interruptions (nondata base)	Bin(4)
– Data transferred to pool (data base)	Bin(4)
– Data transferred to pool (nondata base)	Bin(4)
– Reserved (binary 0)	Char(10)

The meanings for the fields of this selection option are the same as the meanings described for selection option hex 05.

Machine Address Threshold Data (Hex 08):

• Total permanent addresses possible	Char(8)
• Total temporary addresses possible	Char(8)
• Permanent addresses remaining	Char(8)
• Temporary addresses remaining	Char(8)
• Permanent addresses threshold	Char(8)
• Temporary addresses threshold	Char(8)

Total permanent addresses possible is the maximum number of permanent addresses that can exist on the machine.

Total temporary addresses possible is the maximum number of temporary addresses that can exist on the machine.

Permanent addresses remaining is the number of permanent addresses that can still be created before address regeneration must be run.

Temporary addresses remaining is the number of temporary addresses that can still be created before address regeneration must be run.

Permanent addresses threshold is a number that, when it exceeds the number of permanent addresses remaining, causes the event machine address threshold exceeded to be signaled. When the event is signaled, the threshold is reset to 0.

Temporary addresses threshold is a number that, when it exceeds the number of temporary addresses remaining, causes the event machine address threshold exceeded to be signaled. When the event is signaled, the threshold is reset to 0.

Multiprogramming Level Control Information (Hex 0A):

• Machine-wide MPL control	Char(16)
– Machine maximum number of MPL classes	Bin(2)
– Machine current number of MPL classes	Bin(2)
– MPL (max)	Bin(2)
– Ineligible event threshold	Bin(2)
– MPL (current)	Bin(2)
– Number of processes in ineligible state	Bin(2)
– Reserved	Char(4)
• MPL class information (repeated for each MPL class, from 1 to the current number of MPL classes)	Char(16)
– MPL (max)	Bin(2)
– Ineligible event threshold	Bin(2)
– Current MPL	Bin(2)
– Number of processes ineligible state	Bin(2)
– Number of processes assigned to class	Bin(2)
– Transitions (active to ineligible)	Bin(2)
– Transitions (active to MI wait)	Bin(2)
– Transitions (MI wait to ineligible)	Bin(2)

Machine-Wide MPL Control:

Maximum number of MPL classes is the largest number of MPL classes allowed in the machine. These are assumed to be numbered from 1 to the maximum.

Current number of MPL classes is a user-specified value for the number of MPL classes in use. They are assumed to be numbered from 1 to the current number.

MPL (max) is the maximum number of processes which may concurrently be in the active state in the machine.

Ineligible event threshold is a number which, if exceeded by the machine number of ineligible processes defined below, will cause the machine ineligible threshold exceeded event to be signaled. When the event is signaled, this value is set by the machine to 65 535.

MPL (current) is the current number of processes in the active state.

Number of processes in the ineligible state is the number of processes not currently active because of enforcement of both the machine and class MPL rules.

MPL Class Information:

MPL class controls is data in an array that is associated with an MPL class by virtue of its ordinal position within the array.

MPL (max) is the number of processes assigned to the class which may be concurrently active.

Ineligible event threshold, MPL (current), and number of processes in ineligible state are as defined above but apply only to processes assigned to the class.

Number of processes assigned to class is the total number of processes, in any state, assigned to the pool.

Transitions count is the total number of transitions by processes assigned to a class as follows:

1. Active state to ineligible state

2. Active state to wait

3. Wait state to ineligible state

Note that transitions from wait state to active state can be derived as (2 - 3) and transitions from ineligible state to active state as (1 + 3). These numbers are unsigned Bin(2) and are maintained by the machine without regard to overflow conditions.

Machine Reserved Storage Pool Information (Hex 0B):

• Current number of pools	Bin(2)
• Reserved	Char(6)
• Individual main storage pool information (repeated once for each pool, up to the current number of pools)	Char(8)
– Pool size	Bin(2)
– Machine portion of the pool	Bin(2)
– Reserved	Char(4)

Pool size is the amount of main storage assigned to the pool (including the machine reserved portion).

Machine portion of the pool specifies the amount of storage from the pool that is dedicated to machine functions.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

0002 Authorization
0101 Object authorization violation

000A Lock
0301 Object lock transferred

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X		X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
02 Scalar attribute invalid		X	
03 Scalar value invalid	X		
38 Template Specification			
03 Materialization length exception	X		

MODIFY RESOURCE MANAGEMENT CONTROLS (MODRMC)

Op Code (Hex)	Operand 1	Operand 2
0326	Receiver	Control data

Operand 1: Space pointer.

Operand 2: Character(8) scalar (fixed-length).

Description: The control fields implied by operand 2 are modified according to the template specified in operand 1. Operand 2 is an 8-byte character scalar. The first byte generically identifies the type of controls being modified, and the remaining 7 bytes further qualify these controls. The allowable values for operand 2 are machine-dependent.

Operand 1 specifies the values to be used in the modification. The modification template is of the same size and layout as the corresponding materialize resource management data template. The instruction assumes that all values that may be modified under a given value for operand 2 are in fact being modified.

The values allowed for operand 2 and their interpretations are:

• Selection option	Char(1)

Hex 02=Modify auxiliary storage controls
Hex 04=Modify storage transient pool identification
Hex 05=Modify main storage pool controls
Hex 08=Modify machine address threshold
Hex 0A=Modify MPL controls

• Reserved (binary 0)	Char(7)

Associated with these values are the following modification templates, which are assumed to begin 16 bytes past the location specified by operand 1.

Auxiliary Storage Control (Hex 02):

• Reserved	Char(18)*
• Auxiliary storage event threshold	Bin(8)
• Auxiliary storage control flags	Char(1)
– Error logging control flag	Bit 1
– Reserved	Bit 7

Note: The value associated with each entry shown here with an asterisk (*) is ignored.

Auxiliary storage event threshold is a number that, if greater than the number of bytes of auxiliary storage space available, causes the auxiliary storage threshold exceeded event to be signaled. This number is set by the machine to 0 whenever the event is signaled.

Error logging control flag, when set to 1, specifies that any temporary errors subject to threshold control be logged on every occurrence. When set to 0, such errors are logged only when the device specific thresholds are reached.

Modify Storage Transient Pool Identification (Hex 04):

• Storage pool to be used as the transient pool	Bin(2)

The value specified identifies which of the main storage pools is to be used for the transient pool. A value of 0 indicates that the transient pool attribute is to be ignored.

Main Storage Pool Control (Hex 05):

• Machine-wide storage pool control	
– Reserved	Char(4)*
– Current number of pools	Bin(2)
– Reserved	Char(10)*
• Individual main storage pool controls (repeated once for each main storage pool, up to the current number of pools)	
– Pool size	Bin(2)
– Modify failure indicator	Char(1)
– Reserved	Char(13)*

Note: The value associated with each entry shown here with an asterisk (*) is ignored.

Current number of pools equals the maximum number of pools allowed.

Individual main storage pool controls are associated with main storage pools by virtue of their logical position in the array.

Pool size specifies the size of the pool. The unit assumed is the machine minimum transfer size. The sum of the values specified for all pools must equal main storage size, and the value specified for pool 1 must be greater than or equal to the pool 1 minimum size. This minimum value is machine and configuration dependent and the value for any given machine may be materialized using the Materialize Resource Management Data instruction. A value of 0 means that no storage is to be allocated for a pool. A nonzero value must be greater than 16.

Modify failure indicator indicates that the instruction has tried to modify the pool size to less than the machine required minimum size. The machine required minimum size is 16 plus the machine reserved portion of the pool. This field is set to hex FF by the instruction if the pool size is less than the machine required minimum size.

Machine Address Threshold (Hex 08):

• Reserved	Char(32)
• Permanent addresses threshold	Char(8)
• Temporary addresses threshold	Char(8)

Permanent addresses threshold is a number that, when it exceeds the number of permanent addresses remaining, causes the event machine address threshold exceeded to be signaled. When the event is signaled, the threshold is reset to 0.

Temporary addresses threshold is a number that, when it exceeds the number of temporary addresses remaining, causes the event machine address threshold exceeded to be signaled. When the event is signaled, the threshold is reset to 0.

Multiprogramming Level Control (Hex 0A):

• Machine-wide MPL control	
– Reserved	Char(2)*
– Current number of MPL classes	Bin(2)
– MPL (maximum)	Bin(2)
– Ineligible event threshold	Bin(2)
– Reserved	Char(8)*
• MPL class controls (repeated once for each MPL class, up to the current number of MPL classes)	
– MPL (maximum)	Bin(2)
– Ineligible event threshold	Bin(2)
– Reserved	Char(12)*

Note: The value associated with each entry shown here with an asterisk (*) is ignored.

Current number of MPL classes specifies the number of MPL classes required by the user. These are assumed to be numbered from 1. This value may not be modified and is set by the machine to be equal to the machine maximum number of MPL classes.

MPL (maximum) specifies the maximum number of processes which may concurrently be in the active state.

Ineligible event threshold is a number which, if exceeded by the number of processes in the machine in the ineligible state, causes the machine ineligible state threshold event to be signaled. When this event is signaled, the threshold is reset by the machine to 32 767.

MPL class controls are associated with an MPL class by virtue of their ordinal position in the array.

MPL (maximum) and ineligible event threshold are as defined for machine-wide MPL controls but apply only to processes applied to a particular MPL class.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Privileged instruction

Events

0002 Authorization
    0201 Privileged instruction violation

000A Lock
    0301 Object lock transferred

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation			X
02 Privileged instruction			X
10 Damage Encountered			
04 System object damage state	X		X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
07 Modify main storage pool controls invalid			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
02 Scalar attribute invalid		X	
03 Scalar value invalid		X	
38 Template Specification			
01 Template value invalid		X	

RESET ACCESS GROUP (RESAG)

Op Code (Hex)	Operand 1
0365	Access group

Operand 1: System pointer.

Description: This instruction resets the space used within an access group to remove the unused space and therefore reduce the overhead of transferring the access group from one storage media to another.

The access group identified by the system pointer specified by operand 1 is reset to remove unused space within the access group.

Because the access group is used as a storage area for the dynamic creation and destruction of objects, the internal storage space associated with the access group can be automatically extended to a size that is unnecessary for subsequent usage of the access group. This instruction provides a mechanism to reset the internal storage space associated with the access group back to a size which is adequate for its current usage.

Authorization Required

- Operational
  - Operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution

- Modify
  - Access group specified for operand 1

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage	X	X
44 Partial system object damage		X
1A Lock State		
01 Invalid lock state	X	
1C Machine Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attributes	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	

SET ACCESS STATE (SETACST)

Op Code (Hex)	Operand 1
0341	Access state template

Operand 1: Space pointer.

Description: The instruction specifies the access state (which specifies the desired speed of access) that the issuing process has for a set of objects or subobject elements in the execution interval following the execution of the instruction. The specification of an access state for an object momentarily preempts the machine's normal management of an object.

The Set Access State instruction template must be aligned on a 16-byte boundary. The format is:

• Number of objects to be acted upon	Bin(4)
• Reserved (binary 0)	Char(12)
• Access state specifications (repeated as many times as necessary)	Char(32)
– Pointer to object whose access state is to be changed	Space pointer or System pointer
– Access state code	Char(1)
– Reserved (binary 0)	Char(3)
– Access state parameter	Char(12)
Access pool ID	Char(4)
Space length	Bin(4)
Reserved (binary 0)	Char(4)

The number of objects entry specifies how many objects are potential candidates for access state modification. An access state specification entry is included for each object to be acted upon.

The pointer to object entry identifies the object or space which is to be acted upon. For the space associated with a system object, the space pointer may address any byte in the space. This pointer is followed by parameters that define in detail the action to be applied to the object.

The access state code designates the desired access state. The allowed values are as follows:

Access State Code (Hex)	Function and Required Parameter
00	No operations are performed.
01	Associated object is moved into main storage (if not already there) synchronously with the execution of the instruction.
02	Associated object is moved into main storage (if not already there) asynchronously with the execution of the instruction.
03	Associated object is placed in main storage without regard to the current contents of the object. This causes access to secondary storage to be reduced or eliminated.
20	Associated object attributes are moved into main storage synchronous with the instruction's execution. The associated attributes are the attributes that are common to all system objects. The associated pointer to object must be a resolved system pointer.
21	Associated object attributes are moved into main storage asynchronous with the instruction's execution. The associated attributes are the attributes that are common to all system objects. The associated pointer to object must be a resolved system pointer.
40	Perform no operation on the associated object. The main storage occupied by this object is to be used, if possible, to satisfy the request in the next access state specification entry.
80	Associated object not required in main storage by issuing process. Object is moved from main storage synchronously with the execution of the instruction.
81	Associated object not required in main storage by issuing process. Object is moved from main storage asynchronously with the execution of the instruction.

Access state code hex 03 may be used for spaces only. The pointer to the object in the access state specification must be a space pointer. Otherwise, the pointer type invalid exception is signaled.

Access state code hex 40 may be used in conjunction with access state codes hex 01, hex 02, or hex 03. The access state specification entry with access state code hex 40 must immediately precede the access state specification entry with access state code hex 01, hex 02, or hex 03 with which it is to be combined. The pointer to the object in both entries must be a space pointer. Otherwise, the pointer type invalid exception is signaled. The access state parameter field in the access state specification entry with code hex 40 is ignored. The access pool ID and the space length in the entry with access state code hex 01, hex 02, or hex 03 are used.

The access/pool ID entry indicates the desired main storage pool in which the object is to be placed (0000-0006). The storage pool ID entry is treated as a 4-byte logical binary value. When a 0000 storage pool ID is specified, the storage pool associated with the issuing process is used.

The space length entry designates the part of the space associated with the object to be operated on. If the pointer to the object entry is a system pointer, the operation begins with the first byte of the space. If the pointer to the object entry is a space pointer that specifies a location, the operation proceeds for the number of storage units that are designated. No exception is signaled when the number of referenced bytes of the space are not allocated. When operations on objects are designated by system pointers, this operation is performed in addition to the access state modification of the object. This entry is ignored for access state codes hex 20 and hex 21.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
04 Access State		
01 Access state specification invalid	X	
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
04 Pointer not resolved	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attributes	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
38 Template Specification		
01 Template value invalid	X	

SUSPEND OBJECT (SUSOBJ)

Op Code (Hex)	Operand 1
0361	Object to be suspended

Operand 1: System pointer.

Description: The object is truncated to the minimum size needed to maintain its existence in the machine.

After this instruction has been executed, the operational portion of the referenced object cannot be accessed. Ownership and addressability to the object may still be obtained, and some access to the object's attributes is possible. However, any operation that involves access to the operational part of the object results in an exception. This instruction makes space in the system available for other objects. The instruction should be used after an object dump function to save the object on a backup storage medium. An object load function can be used to restore a truncated object to its untruncated or normal state.

Only permanent objects may be suspended. The following objects may be suspended:

- Space object
- Data space
- Data space index
- Index (except those with pointers)
- Program
- Journal space

The following instructions can reference objects that have been suspended:

- Destroy (all suspendable objects)
- Grant Authority
- Journal Object
- Lock Object
- Materialize Authority
- Materialize Authorized Users
- Materialize Journal Object Attributes
- Materialize Object Lock
- Materialize System Object
- Modify Addressability
- Rename Object
- Request I/O (load and dump)
- Resolve System Pointer
- Restart Authority
- Transfer Object Lock
- Transfer Ownership
- Unlock Object

The object suspended exception is signaled if an attempt is made to suspend an object that already is suspended.

Authorization Required

- Suspend
  - Unrestricted (special authorization)

    or
  - Restricted (special authorization and object control authority on the object)

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Object control
  - Operand 1

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port became unusable

001C Journal space
    0301 Threshold reached

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
04 Special authorization required		X
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
04 Object not eligible for operation	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attributes	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

Chapter 14. Object Lock Management Instructions

This chapter describes the lock management instructions. The instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

LOCK OBJECT (LOCK)

Op Code (Hex)	**Operand 1**
03F5	Lock request template

Operand 1: Space pointer.

Description: The instruction requests that locks for system objects identified by system pointers in the space object (operand 1) be allocated to the issuing process. The lock state desired for each object is specified by a value associated with each system pointer in the lock template (operand 1).

The lock request template must be aligned on a 16-byte boundary. The format is as follows:

- Number of lock requests in template — Bin(4)
- Offset to lock state selection values — Bin(2)
- Wait time-out value for instruction — Char(8)
- Lock request options — Char(1)
  - – Lock request type — Bits 0-1
    - 00 = Immediate request
      - – If all locks cannot be immediately granted, signal exception.
    - 01 = Synchronous request
      - – Wait until all locks can be granted.
    - 10 = Asynchronous request
      - – Allow processing to continue and signal event when the object is available.
  - – Access state modifications — Bits 2-3
    - When the process is entering lock wait for synchronous request: — Bit 2
      - 0 = Access state should not be modified.
      - 1 = Access state should be modified.
    - When the process is leaving lock wait: — Bit 3
      - 0 = Access state should not be modified.
      - 1 = Access state should be modified.
  - – Reserved (binary 0) — Bits 4-5*
  - – Time-out option — Bit 6
    - 0 = Wait for specified time, then signal time-out exception.
    - 1 = Wait indefinitely.
  - – Reserved (binary 0) — Bit 7
- Reserved (binary 0) — Char(1)

• Object(s) to be locked	System pointer (one for each object to be locked)
• Lock state selection (repeated for each pointer in the template)	Char(1)
– Requested lock state (1 = lock requested, 0 = lock not requested) Only one state may be requested.	Bits 0-4
LSRD lock	Bit 0
LSRO lock	Bit 1
LSUP lock	Bit 2
LEAR lock	Bit 3
LENR lock	Bit 4
– Reserved (binary 0)	Bits 5-6*
– Entry active indicator	Bit 7
0 = Entry not active – This entry is not used.	
1 = Entry active – Obtain this lock.	

Note: Entries indicated with an asterisk are ignored by the instruction.

Lock Allocation Procedure

A single Lock instruction can request the allocation of one or more lock states on one or more objects. Locks are allocated sequentially until all locks requested are allocated.

When a requested lock state cannot be immediately granted, any locks already allocated by this Lock instruction are released, and the lock request option specified in the lock request template establishes the machine action. The lock request options are described in the following paragraphs.

- Immediate Request–If the requested locks cannot be granted immediately, this option causes the lock request not grantable exception to be signaled. No locks are granted, and the lock request is canceled.

- Synchronous Request–This option causes the process requesting the locks to be placed in the wait state until all requested locks can be granted. If the locks cannot be granted in the time interval established by the wait time-out parameter specified in the lock request template, the lock wait time-out exception is signaled to the requesting process at the end of the interval. No locks are granted, and the lock request is canceled.

- Asynchronous Request–This option allows the requesting process to proceed with execution while the machine asynchronously attempts to satisfy the lock request.

  When the synchronous request option is specified and the requested locks cannot be immediately allocated, the access state modification parameter in the lock request template specifies whether the access state of the process access group is to be modified on entering and/or returning from the lock wait. The parameter has no effect if the process instruction wait access state control attribute specifies that no access state modification is allowed. If the process attribute value specifies that access state modification is allowed and the wait on event access state modification option specifies modify access state, the machine modifies the access state for the specified process access group.

  If a synchronous lock wait is requested and the invocation containing the lock instruction is terminated, then the lock request is canceled.

  If the lock request is satisfied, then the object locked event is signaled to the requesting process. If the request is not satisfied in the time interval established by the wait time-out parameter specified in the lock request template, the wait time-out for pending lock event is signaled to the requesting process. No locks are granted, and the lock request is canceled. If an object is destroyed while a process has a pending request to lock the object, the object destroyed event is signaled to the waiting process.

  If an asynchronous lock wait is requested and the invocation containing the Lock instruction is terminated, then the lock request remains active.

The wait time-out parameter establishes the maximum amount of time that a process competes for the requested set of locks when either the synchronous or asynchronous wait options are specified. The bits in this field are numbered from 0 to 63, and bit 41 is defined as 1024 microseconds. The maximum wait time-out interval allowed is a value equal to (2^{48} - 1) microseconds. Any value that indicates more time than the maximum wait time-out causes the maximum wait time-out to be used. If the wait time-out parameter is specified with a value of binary 0, then the value associated with the default wait time-out parameter in the process definition template establishes the time interval.

When two or more processes are competing for a conflicting lock allocation on a system object, the machine attempts to first satisfy the lock allocation request of the process with the highest priority. Within that priority, the machine attempts to satisfy the request that has been waiting longest.

If any exception is identified during the instruction's execution, any locks already granted by the instruction are released, and the lock request is canceled.

For each system object lock counts are kept by lock state and by process. When a lock request is granted, the appropriate lock count(s) of each lock state specified is incremented by 1.

If a previously unsatisfied lock request is satisfied by the transfer of a lock from another process, the lock request and transfer lock are treated as independent events relative to lock accounting. The appropriate lock counts are incremented for both the lock request and the transfer lock function.

Authorization Required

- Some authority or ownership
  - Objects to be locked

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000A Lock
    0101 Object locked
    0201 Object destroyed
    0401 Asynchronous lock wait timeout

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state		X
02 Lock request not grantable	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
06 Machine lock limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
38 Template Specification		
01 Template value invalid	X	
3A Wait Time-out		
02 Lock		X

LOCK SPACE LOCATION (LOCKSL)

Op Code (Hex)	Operand 1	Operand 2
03F6	Space location	Lock type request

Operand 1: Space pointer data object.

Operand 2: Char(1) scalar.

Description: The space location identified by operand 1 is locked according to the request specified by operand 2. Locking the space location does not prevent any byte operation from referencing that location, nor does it prevent the space from being extended, truncated, or destroyed. Space location locks follow the normal locking rules with respect to conflicts and waits but are strictly symbolic in nature.

Following is the format of operand 2:

- Requested lock state Char(1)

  Hex 80=LSRD lock
  Hex 40=LSRO lock
  Hex 20=LSUP lock
  Hex 10=LEAR lock
  Hex 08=LENR lock
  All other values are reserved.

If the requested lock cannot be immediately granted, the process will enter a synchronous wait for the lock, for a period of up to the interval specified by the process default time-out value. If the wait exceeds this time limit, a space location lock wait exception is signaled, and the requested lock is not granted.

During the wait, the process access state may be modified. This can occur if the process' instruction wait access state control attribute is set to allow access state modification.

A space pointer machine object cannot be specified for operand 1.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object			X
44 Partial system object damage			X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
06 Machine lock limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
02 Object destroyed	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
03 Scalar value invalid		X	
3A Wait Time-Out			
04 Space location lock wait	X		

MATERIALIZE ALLOCATED OBJECT LOCKS (MATAOL)

Op Code (Hex)	Operand 1	Operand 2
03FA	Receiver	System object or space location

Operand 1: Space pointer.

Operand 2: System pointer or space pointer data object.

Description: This instruction materializes the current allocated locks on a designated object. If operand 2 is a system pointer, the current allocated locks on the object identified by the system pointer specified by operand 2 are materialized into the template specified by operand 1. If operand 2 is a space pointer, the current allocated locks on the specified space location are materialized into the template specified by operand 1. The materialization template identified by operand 1 must be 16-byte aligned. The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Current cumulative lock status	Char(3)
– Lock states currently allocated (1 = Yes)	Char(1)
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Locks implicitly set	Bit 5
Reserved (binary zero)	Bits 6-7
– Reserved (binary 0)	Char(2)
• Reserved (binary 0)	Char(1)
• Number of lock descriptions following	Bin(2)
• Reserved (binary 0)	Char(2)
• Lock state descriptors (repeated for each lock currently allocated)	Char(32)
– Process control space	System pointer
– Lock state being described Hex 80=LSRD lock request Hex 40=LSRO lock request Hex 20=LSUP lock request Hex 10=LEAR lock request Hex 08=LENR lock request All other values are reserved	Char(1)
– Status of lock request A value of 1 in the corresponding bit indicates the condition is true:	Char(1)
Reserved (binary 0)	Bits 0-5
Implicit lock (machine applied)	Bit 6
Lock held by process	Bit 7
– Reserved (binary 0)	Char(14)

Locks may be implicitly applied by the machine (status code = hex 02). If the implicit lock is held for a process, a pointer to the associated process control space is returned. Locks held by the machine but not related to a specific process, cause the process control space entry to be assigned a value of binary zero.

Only a single lock state is returned for each lock state descriptor entry.

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than eight causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions are signaled in the event the receiver contains insufficient area for the materialization, other than the materialization length exception.

A space pointer machine object cannot be specified for operand 2.

Events

0002 Authorization
0101 Authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

000D Machine status
0101 Machine check

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object
0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	X
10 Damage Encountered			
04 System object damage		X	X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE DATA SPACE RECORD LOCKS (MATDRECL)

Op Code (Hex)	Operand 1	Operand 2
032E	Receiver	Record selection template

Operand 1: Space pointer.

Operand 2: Space pointer.

Description: This instruction materializes the current allocated locks on the specified data space record.

The current lock status of the data space record identified by the template in operand 2 is materialized into the space identified by operand 1.

The record selection template identified by operand 2 must be 16-byte aligned. The format of the record selection template is as follows.

• Record selection	Char(24)
– Data space identification	System pointer
– Record number	Bin(4)
– Reserved	Char(4)
• Lock selection	Char(8)
– Materialize data space locks held 1 = Materialize 0 = Do not materialize	Bit 0
– Materialize data space locks waited for 1 = Materialize 0 = Do not materialize	Bit 1
– Reserved	Bits 2-7
– Reserved	Char(7)

The data space identification must be a system pointer to a data space.

The record number is a relative record number within that data space. If the record number is zero then all locks on the specified data space will be materialized. If the record number is not valid for the specified data space a template value invalid exception is signaled.

Both of the fields specified under lock selection are bits which determine the locks to be materialized. If the first bit is on, the current holders of the specified data space record lock are materialized. If the second bit is on, any process waiting to lock the specified data space record is materialized.

The materialization template identified by operand 1 must be 16-byte aligned. The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Materialization data	Char(8)
– Count of locks held	Bin(2)
– Count of locks waited for	Bin(2)
– Reserved	Char(4)
• Locks held identification (repeated for each lock held)	Char(32)
– Process identification	System pointer
– Record number	Bin(4)
– Reserved	Char(12)
• Locks waited for identification (repeated for each lock waited for)	Char(32)
– Process identification	System pointer
– Record number	Bin(4)
– Reserved	Char(12)

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The count of locks held contains the number of locks held. One system pointer to the PCS (process control space) of each process holding a lock and the relative record number which is locked are materialized in the area identified as locks held identification. These fields contain data only if held data space locks are selected for materialization.

The count of locks waited for contains the number of locks being waited for. One system pointer to the PCS (process control space) of each process waiting for a lock and the relative record number which the process is waiting for is materialized in the area identified as locks waited for identification. These fields contain data only if data space record locks waited for are selected for materialization.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	X
10 Damage Encountered			
04 System object damage		X	X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state		X	
1C Machine Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
01 Template value invalid		X	
03 Materialization length exception	X		

MATERIALIZE OBJECT LOCKS (MATOBJLK)

Op Code (Hex)	Operand 1	Operand 2
033A	Receiver	System object or space location

Operand 1: Space pointer.

Operand 2: System pointer or space pointer data object.

Description: If operand 2 is a system pointer, the current lock status of the object identified by the system pointer is materialized into the template specified by operand 1. If operand 2 is a space pointer, the current lock status of the specified space location is materialized into the template specified by operand 1. The materialization template identified by operand 1 must be aligned on a 16-byte boundary. The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Current cumulative lock status	Char(3)
– Lock states currently allocated (1 = yes)	Char(1)
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Locks implicitly set	Bit 5
Reserved (binary 0)	Bits 6-7
– Lock states for which processes are in synchronous wait (1 = yes)	Char(1)
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Implicit lock request	Bit 5
Reserved (binary 0)	Bits 6-7
– Lock states for which processes are in asynchronous wait (1 = yes)	Char(1)
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Reserved (binary 0)	Bits 5-7
• Reserved (binary 0)	Char(1)
• Number of lock descriptions that follow	Bin(2)
• Reserved (binary 0)	Char(2)
• Lock state descriptors (repeated for each lock currently allocated or waited for)	Char(32)
– Process control space	System pointer
– Lock state being described	Char(1)
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Reserved (binary 0)	Bits 5-7
– Status of lock request	Char(1)
Reserved	Bits 0-2
Waiting because this lock is not available	Bit 3
Process in asynchronous wait for lock	Bit 4
Process in synchronous wait for lock	Bit 5
Implicit lock (machine-applied)	Bit 6
Lock held by process	Bit 7
– Reserved (binary 0)	Char(14)

Locks may be applied by the machine (status code = hex 02). If the implicit lock is held for a process, a pointer to the associated process control space is returned. Locks held by the machine but not related to a specific process cause the process control space entry to be assigned a value of binary 0.

Only a single lock state is returned for each lock state descriptor entry.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This total is supplied as input to the instruction and is not modified by the instruction. A total of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled if the receiver contains insufficient area for the materialization.

A space pointer machine object cannot be specified for operand 2.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	X
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE PROCESS LOCKS (MATPRLK)

Op Code (Hex)	**Operand 1**	**Operand 2**
0312	Receiver	Process control space

Operand 1: Space pointer.

Operand 2: System pointer or null.

Description: The lock status of the process identified by operand 2 is materialized into the receiver specified by operand 1. If operand 2 is null, the lock status is materialized for the process issuing the instruction. The materialization identifies each object or space location for which the process has a lock allocated or for which the process is in a synchronous or asynchronous wait. The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Number of lock entries	Bin(2)
• Reserved (binary 0)	Char(6)
• Lock status (repeated for each lock currently allocated or waited for by the process)	Char(32)
– Object, space location, or binary 0 if no pointer exists	System pointer or space pointer
– Lock state	Char(1)
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Reserved (binary 0)	Bits 5-7
– Status of lock state for process	Char(1)
Reserved	Bits 0-1
Object or space location no longer exists	Bit 2
Waiting because this lock is not available	Bit 3
Process in asynchronous wait for lock	Bit 4
Process in synchronous wait for lock	Bit 5
Implicit lock (machine-applied)	Bit 6
Lock held by process	Bit 7
• Reserved (binary 0)	Char(14)

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled if the receiver contains insufficient area for the materialization.

Authorization Required

- Retrieve
  - Context referenced by address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
28 Process State			
02 Process control space not associated with a process		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE PROCESS RECORD LOCKS (MATPRECL)

Op Code (Hex)	Operand 1	Operand 2
031E	Receiver	Process selection template

Operand 1: Space pointer.

Operand 2: Space pointer.

Description: This instruction materializes the current allocated data space record locks held by the process. The current lock status of the process identified in the process selection template specified by operand 2 is materialized into the receiver identified by operand 1. The materialization identifies each data space record lock which the process has or the process is waiting to obtain.

If the PCS (process control space) pointer is null or all zeros, the lock activity for the process issuing the instruction is materialized.

The process selection template identified by operand 2 must be 16-byte aligned. The format of the process selection template is as follows:

• Process selection	Char(16)
– Process identification	System pointer
• Lock selection	Char(8)
– Materialize held locks 1 = Materialize 0 = Do not materialize	Bit 0
– Materialize locks waited for 1 = Materialize 0 = Do not materialize	Bit 1
– Reserved	Bits 2-7
– Reserved	Char(7)

The process identification must be a system pointer to a PCS (process control space) or null, all zeros.

Both of the fields specified under lock selection are bits which determine the locks to be materialized. If the first bit is on, any data base record lock held by the process is materialized. If the second bit is on, any data base record lock the process is waiting for is materialized.

The materialization template identified by operand 1 must be 16-byte aligned. The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Materialization data	Char(8)
– Count of locks held	Bin(2)
– Count of locks waited for	Bin(2)
– Reserved	Char(4)
• Locks held identification (repeated for each lock held)	Char(32)
– Data space identification	System pointer
– Relative record number	Bin(4)
– Reserved	Char(12)
• Locks waited for identification (repeated for each lock waited for)	Char(32)
– Data space identification	System pointer
– Relative record number	Bin(4)
– Reserved	Char(12)

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The count of locks held contains the number of locks held by the process. One system pointer to the data space and relative record number in the data space is materialized in the area identified as locks held identification for each lock. These fields contain data only if held locks are selected for materialization.

The count of locks waited for contains the number of locks that the process is waiting for. One system pointer to the data space and relative record number in the data space is materialized in the area identified as locks waited for identification for each lock waited for. These fields contain data only if locks waited for are selected for materialization.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
0101 Authorization violation

000C Machine resources
0201 Machine auxiliary storage exceeded

000D Machine status
0101 Machine check

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object
0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	X
10 Damage Encountered			
04 System object damage		X	X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state		X	
1C Machine Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
01 Template value invalid		X	
03 Materialization length exception	X		

MATERIALIZE SELECTED LOCKS (MATSELLK)

Op Code (Hex)	Operand 1	Operand 2
033E	Receiver	Object or space location template

Operand 1: Space pointer.

Operand 2: System pointer or space pointer data object.

Description: The locks held by the process issuing this instruction for the object or space location referenced by operand 2 are materialized into the template specified by operand 1. The format of the materialization template is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Cumulative lock status for all locks on operand 2	Char(1)
– Lock state	Char(1)
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Reserved (binary 0)	Bits 5-7
• Reserved	Char(3)
• Number of lock entries	Bin(2)
• Reserved	Char(2)
• Lock status (repeated for each lock currently allocated)	Char(2)
– Lock state Hex 80=LSRD lock request Hex 40=LSRO lock request Hex 20=LSUP lock request Hex 10=LEAR lock request Hex 08=LENR lock request All other values are reserved	Char(1)
– Status of lock	Char(1)
Reserved (binary 0)	Bits 0-5
Implicit lock 0 = Not implicit lock 1 = Is implicit lock	Bit 6
Reserved (binary 1)	Bit 7

The first 4 bytes of the materialization identifies the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identifies the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

A space pointer machine object cannot be specified for operand 2.

Authorization

- Retrieve
  - Context referenced by address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Authorization violation

000C Machine resources
  0201 Machine auxiliary storage exceeded

000D Machine status
  0101 Machine check

0010 Process
  0601 Exception signaled to process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference
  0201 Object location reference

0017 Damage set
  0401 System object
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
02 Unauthorized for operation		X	
10 Damage Encountered			
04 System object	X	X	X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
28 Process State			
02 Process control space not associated with a process		X	
2A Program creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
03 Materialization length exception	X		

TRANSFER OBJECT LOCK (XFRLOCK)

Op Code (Hex)	Operand 1	Operand 2
0382	Receiving process control space	Lock transfer template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: The receiving process (operand 1) is allocated the locks designated in the lock transfer template (operand 2). Upon completion of the transfer lock request, the current process no longer holds the transferred lock(s).

Operand 2 identifies the objects and the associated lock states that are to be transferred to the receiving process. The space contains a system pointer to each object that is to have a lock transferred and a byte which defines whether this entry is active. If the entry is active, the space also contains the lock states to be transferred. Operand 2 must be aligned on a 16-byte boundary. The format is as follows:

• Number of lock transfer requests in template	Bin(4)
• Offset to lock state selection bytes (1 byte for each lock transfer request)	Bin(2)
• Reserved (binary 0)	Char(8)*
• Reserved	Char(1)
– Reserved	Bits 0-6*
– Reserved (binary 0)	Bit 7
• Reserved (binary 0)	Char(1)
• Object lock(s) to be transferred	System pointer (one for each object lock to be transferred)
• Lock state selection (repeated for each pointer in the template)	Char(1)
– Lock state to transfer. Only one state may be requested. (1 = transfer)	Bits 0-4
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
– Reserved (binary 0)	Bit 5*
– Lock count	Bit 6
0 = The current lock count is transferred.	
1 = A lock count of 1 is transferred.	
– Entry active indicator	Bit 7
0 = Entry not active This entry is not used.	
1 = Entry active This lock is transferred.	

Note: Entries indicated by an asterisk are ignored by the instruction.

If the receiving process is issuing the instruction, then no operation is performed, and no exception is signaled. The lock count transferred is either the lock count held by the transferring process or a count of 1. If the receiving process already holds an identical lock, then the final lock count is the sum of the count originally held by the receiving process and the transferred count.

Only locks currently allocated to the process issuing the instruction can be transferred. If the transfer of an allocated lock would result in the violation of the lock allocation rules, then the lock cannot be transferred. An implicit lock may not be transferred.

No locks are transferred if an entry in the template is invalid.

The locks specified by operand 2 are transferred sequentially and individually. If one lock cannot be transferred because the process does not hold the indicated lock on the object, then exception data is saved to identify the lock that could not be transferred. Processing of the next lock to be transferred continues.

After all locks specified in operand 2 have been processed, the object lock transferred event is signaled to the process receiving the locks if any locks were transferred. If any lock was not transferred, the invalid object lock transfer request exception is signaled.

When an object lock is transferred, the transferring process synchronously loses the record of the lock, and the object is locked to the receiving process. However, the receiving process obtains the lock asynchronously after the instruction currently being executed is completed. If the transferring process holds multiple locks for the object, any lock states not transferred are retained in the process.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000A Lock
    0301 Object lock transferred

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state			X
04 Invalid object lock transfer request		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
28 Process State			
02 Process control space not associated with a process	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
38 Template Specification			
01 Template value invalid		X	

UNLOCK OBJECT (UNLOCK)

Op Code (Hex)	Operand 1
03F1	Unlock template

Operand 1: Space pointer.

Description: The instruction releases the object locks that are specified in the unlock template. The template specified by operand 1 identifies the system objects and the lock states (on those objects) that are to be released. The unlock template must be aligned on a 16-byte boundary. The format is as follows:

• Number of unlock requests in template	Bin(4)
• Offset to lock state selection bytes	Bin(2)
• Reserved (binary 0)	Char(8)*
• Unlock option	Char(1)
– Reserved (binary 0)	Bits 0-3*
– Unlock type	Bits 4-5
00 = Unlock specific locks now allocated to process	
01 = Cancel specific asynchronously waiting lock request or allocated locks	
10 = Cancel all asynchronously waiting lock requests	
11 = Invalid	
– Reserved (binary 0)	Bit 6*
– Reserved (binary 0)	Bit 7
– Reserved (binary 0)	Char(1)
• Object to unlock (one for each unlock request)	System pointer
• Unlock options (repeated for unlock request)	Char(1)
– Lock state to unlock (only one state can be selected) (1 = unlock)	Bits 0-4
LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
– Lock count option	Bit 5
0 = Lock count reduced by 1	
1 = All locks are unlocked – The set lock count = 0	
– Reserved (binary 0)	Bit 6*
– Entry active indicators	Bit 7
0 = Entry not active – This entry is not used.	
1 = Entry active – These locks are unlocked.	

Note: Entries indicated by an asterisk are ignored by the instruction.

If all asynchronous lock waits are being canceled (unlock type 10), then system pointers to the objects and unlock options for each object are not required. If the asynchronous lock fields are provided in the template, then the data is ignored.

Unlock type 01 attempts to cancel an asynchronous lock request that is identical to the one defined in the template. After the instruction attempts to cancel the specified request, program execution continues just as if unlock type 00 had been selected. A waiting lock request is canceled if the number of active requests in the template, the objects, the objects corresponding lock states, and the order of the active entries in the template all match.

When a lock is released, the lock count is reduced by 1 or set to 0 in the specified state. This option is specified by the lock count option parameter.

If unlock type 01 is specified and the unlock count option for an object lock is 0 (lock count reduced by 1), then a successful cancel satisfies this request, and no additional locks on the object are unlocked. If the unlock count option for an object lock is set to 1 (set lock count to 0), the results of the cancel are disregarded, and all held locks on the object are unlocked.

Specific locks can be unlocked only if they are allocated to the process issuing the unlock instruction. Implicit locks may not be unlocked with this instruction. No locks are unlocked if an entry in the template is invalid.

Object locks to unlock are processed sequentially and individually. If one specific object lock cannot be unlocked because the process does not hold the indicated lock on the object, then exception data is saved, but processing of the instruction continues.

After all requested object locks have been processed, the invalid unlock request exception is signaled if any object lock was not unlocked.

If unlock type 01 is selected and the cancel attempt is unsuccessful, an invalid unlock request exception is signaled when any object lock in the template is not unlocked.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state		X
03 Invalid unlock request	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
38 Template Specification		
01 Template value invalid	X	

UNLOCK SPACE LOCATION (UNLOCKSL)

Op Code (Hex)	Operand 1	Operand 2
03F2	Space location	Lock type

Operand 1: Space pointer data object.

Operand 2: Char(1) scalar.

Description: The lock type specified by operand 2 is removed from the space location identified by operand 1 (the lock must be held by the process that issues the instruction). The space location specified by operand 1 need not exist when this instruction is issued, although the space pointer must be a valid pointer as used to lock the space location. When multiple locks of the same lock state for the same space location need to be unlocked, this instruction must be issued for each lock held for the space location. If an attempt is made to unlock a space location lock not held by the process, an invalid space location unlock exception is signaled.

Following is the format of operand 2:

- Lock state to be unlocked Char(1)

  Hex 80=LSRD lock
  Hex 40=LSRO lock
  Hex 20=LSUP lock
  Hex 10=LEAR lock
  Hex 08=LENR lock
  All other values are reserved.

A space pointer machine object cannot be specified for operand 1.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

0000C Machine resources
  0201 Machine auxiliary storage exceeded

000D Machine status
  0101 Machine check

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X		
02 Boundary alignment	X		
03 Range	X		
06 Optimized addressability invalid	X		
08 Argument/Parameter			
01 Parameter reference violation	X		
10 Damage Encountered			
04 System object	X		X
44 Partial system object damage			X
1A Lock State			
05 Invalid space location unlock	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
06 Machine lock limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
02 Object destroyed	X	X	
24 Pointer Specification			
01 Pointer does not exist	X		
02 Pointer type invalid	X		
2A Program Creation			
06 Invalid operand type	X		
07 Invalid operand attribute	X		
08 Invalid operand value range	X		
0C Invalid operand ODT reference	X		
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
03 Scalar value invalid		X	

Chapter 15. Event Management Instructions

This chapter describes all instructions used for event management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instructions Summary*.

CANCEL EVENT MONITOR (CANEVTMN)

Op Code (Hex)	**Operand 1**
03D1	Event monitor template

Operand 1: Character(48) scalar (fixed-length).

Description: An event monitor having exactly the same qualifications as the template referenced by the operand 1 template is canceled, and the event monitor is disassociated from the currently executing process. The qualifications used to determine the event monitor are based on event identification, compare value length, and compare value. All event monitors currently associated with the process are examined until a matching monitor is located. If a monitor is not found within the process, the event monitor not present exception is signaled.

The Cancel Monitor Event instruction template identified by operand 1 must be 16-byte aligned and have the following format:

• Option indicators	Char(2)
– Compare value content	Bit 0
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 1-15
• Reserved (binary 0)	Char(8)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length	Bin(2)
• Compare value	Char(32)

If compare value content is set to system pointer present, compare value length must be at least 16 and the system pointer must be located in the first 16 bytes of the compare value.

If the compare value length entry is 0, the compare value entry is ignored. If the event monitor to be canceled has a compare value qualifier, the compare value length and compare values must be identical to that specified in the operand 1 template.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
14 Event Management		
02 Event monitor not present	X	
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
02 Scalar attributes invalid	X	
03 Scalar value invalid	X	

DISABLE EVENT MONITOR (DBLEVTMN)

Op Code (Hex)	Operand 1
0399	Event monitor template

Operand 1: Character(48) scalar (fixed-length).

Description: The event monitor with the same qualifications as the template referenced by operand 1 is placed in the disabled state. When an event monitor is disabled, the machine does not schedule execution of the event handling routine associated with the event monitor and a wait on event for the disabled monitor is not satisfied.

If the event monitor specifies that signals are to be held while the event monitor is disabled, the signals and event-related data are retained. The maximum number of signals to be retained is denoted by an event monitor attribute in the Monitor Event instruction. Signals and event-related data received by the event monitor in excess of the maximum number to be retained are lost.

If the event monitor specifies that signals are not to be held while the event monitor is disabled, the signals and event-related data are not recorded.

If an event monitor is signaled while it is in the disabled state, the signals are retained, and the monitor's event handler, if specified, is scheduled for execution when the event monitor is enabled.

The operand 1 Disable Monitor Event instruction template must be 16-byte aligned and have the following format:

• Option indicators	Char(2)
– Compare value content	Bit 0
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 1-15
• Reserved (binary 0)	Char(8)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)

• Compare value length	Bin(2)
• Compare value	Char(32)

If compare value content is set to system pointer present, compare value length must be at least 16 and the system pointer must be located in the first 16 bytes of the compare value.

If the compare value length is 0, the compare value entry is ignored by the instruction. The event monitor to be disabled must also have a zero length compare value.

If no event monitor with an identical event identification, compare value length, and compare value is found within the executing process, the event monitor not present exception is signaled.

If the event monitor is currently disabled, no operation takes place, and no exception is signaled.

An event monitor monitoring timer event (class 0014) cannot be disabled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
14 Event Management		
02 Event monitor not present	X	
05 Disable timer event monitor invalid	X	
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
02 Scalar attributes invalid	X	
03 Scalar value invalid	X	

ENABLE EVENT MONITOR (EBLEVTMN)

Op Code (Hex)	Operand 1
0369	Event monitor template

Operand 1: Character(48) scalar (fixed-length).

Description: The instruction places an event monitor in the enabled state. The event monitor may have been initially established in the disabled state or may have been disabled explicitly by the Disable Monitor Event instruction.

If the event monitor is for an interval timer event, the time interval begins when the monitor is enabled.

If the event monitor is currently enabled, no operation takes place, and no exception is signaled.

If the event monitor currently has any retained signals, the event handling program, if specified, is invoked.

The operand 1 template must be 16-byte aligned and have the following format:

• Option indicators	Char(2)
– Compare value content	Bit 0
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 1-15
• Reserved (binary 0)	Char(8)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length	Bin(2)
• Compare value	Char(32)

If compare value content is set to system pointer present, compare value length must be at least 16 and the system pointer must be located in the first 16 bytes of the compare value.

If the compare value length is 0, the instruction ignores the compare value entry. The event monitor to be enabled must have a zero length compare value.

If no event monitor with an identical event identification, compare value length, and compare value is currently associated with the executing process, the event monitor not present exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
14 Event Management		
02 Event monitor not present	X	
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
02 Scalar attributes invalid	X	
03 Scalar value invalid	X	

MATERIALIZE EVENT MONITORS (MATEVTMN)

Op Code (Hex)	Operand 1
0379	Receiver

Operand 1: Space pointer.

Description: This instruction materializes the event monitors for the current process. The event monitors for the current process are materialized into the materialization template specified by operand 1.

The materialization template identified by operand 1 must be 16-byte aligned and has the following format:

Template header	Char(16)
• Template size specification	Char(8)
– Number of bytes provided	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Materialization data	Char(8)
– Count of event monitors	Bin(4)
– Reserved (binary 0)	Char(4)
Monitor data (repeated for each monitor and identical to the template for monitor event)	Char(*)
• Reserved (binary 0)	Char(16)
• Event handler specification (system pointer to the program or all zeros if no event handler is specified)	System pointer
• Reserved (binary 0)	Char(2)

• Option indicators	Char(2)
– Monitor domain	Bit 0
0 = Machine-wide	
1 = Process-directed	
– Reserved (binary 0)	Bits 1-7
– Enabled/Disabled state	Bit 8
0 = Enabled state	
1 = Disabled state	
– Signal retention option	Bit 9
0 = Signals are retained while disabled	
1 = Signals are not retained while disabled	
– Short form option	Bit 10
0 = Include event-related data with the signal	
1 = Do not include event-related data with the signal	
– Event handler qualifier	Bit 11
0 = Event handler not present	
1 = Event handler present	
– Compare value content	Bit 12
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 13-15
• Maximum number of signals to be retained	Bin(4)
• Event priority (0-255; 0 = highest priority)	Bin(2)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length	Bin(2)
• Compare value (padded with binary 0 if less than 32 bytes)	Char(32)

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

All other values in the template are identical to those specified as input to the Monitor Event instruction.

Events

000C Machine resources
 0201 Machine auxiliary storage exceeded

000D Machine status
 0101 Machine check

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage		X
44 Partial system object damage		X
1C Machine Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	
38 Template Specification		
03 Materialization length exception	X	

MODIFY PROCESS EVENT MASK (MODPEVTM)

Op Code (Hex)	Operand 1	Operand 2
0372	Previous mask state	New mask state

Operand 1: Binary(2) scalar variable or null.

Operand 2: Binary(2) scalar or null.

Description: This instruction optionally modifies and retrieves the state of the event mask in the process executing this instruction. If the event mask is in the masked state, the machine does not schedule signaled event monitors in the process. The event monitors continue to be signaled by the machine or other processes. When the process is modified to the unmasked state, event handlers are scheduled to handle those events that occurred while the process was masked and those events occurring while in the unmasked state. The number of signals retained while the process is masked is specified by the attributes of the event monitor associated with the process.

The process is automatically masked by the machine when event handlers are invoked. If the process is unmasked in the event handler, other events can be handled if another enabled event monitor within that process is signaled. If the process is masked when it exits from the event handler, the machine explicitly unmasks the process.

Valid operand values are:

0	–	Masked
256	–	Unmasked

Other values are reserved and must not be specified. If any other values are specified, a scalar value invalid exception is signaled. If operand 1 is null, the current mask state is not returned. If operand 2 is null, the mask state is not modified. If both operands are null, an invalid operand type exception is signaled. If both operands are not null, the mask state is retrieved before the state is modified.

Events

0000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand ODT reference		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
02 Scalar attributes invalid	X	X	
03 Scalar value invalid		X	

MONITOR EVENT (MNEVT)

Op Codes (Hex)	Operand 1
0371	Event monitor template

Operand 1: Space pointer.

Description: This instruction specifies an intent to monitor for a specific event and defines a preliminary event handling mechanism within the executing process. It allows monitoring of both machine and user-signaled events.

The monitor is in effect until a Cancel Monitor Event instruction is issued or until the process terminates.

The event monitor template identified by operand 1 must be 16-byte aligned and have the following format:

• Template size specification	Char(8)
– Number of bytes provided	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Reserved (binary 0)	Char(8)
• Event handler specification (program)	System pointer
• Reserved (binary 0)	Char(2)

• Option indicators	Char(2)
– Monitor domain	Bit 0
0 = Machine-wide	
1 = Process-directed	
– Reserved (binary 0)	Bits 1-7
– Enabled/disabled option	Bit 8
0 = Enabled state	
1 = Disabled state	
– Signal retention option	Bit 9
0 = Signals are retained while disabled.	
1 = Signals are not retained while disabled.	
– Short form option	Bit 10
0 = Event-related data is included with the signal.	
1 = Event-related data is not included with the signal.	
– Event handler qualifier	Bit 11
0 = Event handler not present	
1 = Event handler present	
– Compare value content	Bit 12
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 13-15
• Maximum number of signals to be retained	Bin(4)
• Event priority (0-255; 0 = highest priority)	Bin(2)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length	Bin(2)
• Compare value	Char(32)

Note: This instruction ignores template entries annotated with an asterisk.

The attributes of the event monitor have the following meaning:

- Event handler specification–This entry is a system pointer with addressability to a program that is to be given control on the occurrence of the event. The pointer must reference a program and the currently adopted user profile or the process user profile must carry operational authority for the program. The entry is ignored if the event handler qualifier indicator is set to not present.

- Option indicators

  These indicators further describe the qualifications of the event monitor.
  - Monitor domain–This attribute denotes whether the event is to be monitored on a process-directed or a machine-wide basis. If the monitor domain is set to process-directed, the event monitor is signaled to monitor machine events occurring based on the execution of the monitoring process or to monitor user-signaled events that are specifically directed at the monitoring process. If the monitor domain is set to machine-wide, the event monitor is capable of receiving both process-directed or machine-wide signals.

    Most machine events are signaled machine-wide, which means that to monitor machine events, the monitor domain must be specified as machine-wide. However, a specific subset of machine events is signaled directly to a process because the event is associated with a function initiated by the process. The following machine events, for example, are signaled directly to a process:
    a. All timer types (time of day, interval, repetitive interval)
    b. REQIO complete (signaled to process issuing the REQIO instruction)
    c. Process initiated successfully/unsuccessfully (signaled to the initiator of the process)
    d. Process terminated (signaled to the initiator of the process)
    e. Pending lock granted (signaled to process receiving the lock)
    f. Object destroyed during asynchronous lock wait (signaled to the requesting process)
    g. Lock transferred (signaled to the receiving process)
    h. Asynchronous lock wait time-out (signaled to the requesting process)

Events signaled through the Signal Event instruction can be signaled to all processes in the machine (machine-wide) or to a specific process. The Signal Event instruction allows specification of the domain of the signal–machine-wide or process.

- Enabled/disabled initial state–This option specifies whether or not the event monitor is to be initially enabled for signals immediately. The state can be altered by the Enable Monitor Event and the Disable Monitor Event instructions.
- Signal retention option–This option specifies whether or not signals are to be retained while the event monitor is disabled. This option can be used to limit the maximum number of signals to be retained value.
- Short form option–This option specifies whether or not the specific event-related data is to be appended to the standard event data when the signal is presented. If the short form option is set to do not include event-related data with the signal, only the standard data is presented upon retrieval of the signal. This option has a performance advantage.
- Event handler qualifier–This indicator specifies whether or not the corresponding system pointer entry in the template is to be used. If this indicator denotes the presence of a system pointer, the pointer object must be a resolved or initial-valued system pointer addressing a program.
- Compare value content–This indicator specifies the presence or absence of a system pointer in the compare value. The indicator is ignored if the compare value length is 0. If the indicator is set to system pointer present and the compare value length is not equal to 0, the compare value length must be at least 16, and the system pointer is assumed to be located in the first 16 bytes of the compare value. The template must be 16-byte aligned. The instruction does not verify that the system pointer addresses an object type associated with the machine event specified in the event identification.

- Maximum number of signals to be retained
  - This attribute indicates the number of signals that the machine retains while the process is masked, while an event monitor is disabled, or while the event monitor is enabled with the events not being handled as rapidly as they are being signaled. The number must be greater than 0. While this number of signals is pending, any signals received are discarded.

- Event priority
  - This attribute specifies the relative importance of this event compared with other events to be monitored within a process. The event priority value establishes the order in which event handlers are scheduled if multiple events have occurred, and it determines the preemptability when a process is waiting for one event and another occurs.

    The duplicate event monitor exception (hex 1401) is signaled if an identical event monitor exists but it specifies a different event handling program. If an identical event monitor already exists with the same event handler specified, then no exception is signaled.

- Event identification–This attribute is an identification corresponding to a machine set of events or the identification specified for a user-signaled event. An event class value of hex 0000 is invalid. An event type value of hex 00 denotes generic monitoring by event class; that is, all types and subtypes within an event class are monitored. An event subtype value of hex 00 denotes generic monitoring by event class and type; that is, all subtypes within an event class and type are monitored. Timer events require the specification of class, type, and subtype; that is, there is no generic monitor capability for timer events. If an interval timer event monitor is created as disabled, the time interval is not started until the monitor is enabled. The event class for machine events is in the range of hex 0001 to hex 7FFF. User-defined events may be signaled from classes hex 8000 and above. See Chapter 24. *Event Specifications* for the event identifications.

- The compare value length entry is used when the machine event allows or requires a compare value, and it must be equal to the length specified for the event. The compare value length entry is also used for user-signaled event monitoring to further qualify a signal. For user events, the length cannot exceed 32 characters. A template value invalid exception is signaled if the compare value length is less than 0 or greater than 32 characters.

- The compare value entry is used to further qualify a signal. If a compare value length of 0 is specified, the compare value entry and the compare value content indicator are ignored. Certain machine events require a compare value to specify to the machine under what conditions the event is to be signaled. For example, the timer class machine events require the specification of the time interval to be monitored. For these events, the compare value length must contain the proper value, and the compare value must always be present. A template value invalid exception is signaled if an invalid compare value length or compare value is specified.

  If the compare value qualifies the event monitor and the length of the compare value specified in the event monitor is greater than the length specified in an event generated by the Signal Event instruction, the event monitor is not signaled. If the compare value length in the event monitor is less or equal to the compare value length in an event generated by the Signal Event instruction, the compare value length from the event monitor is used as the comparison length for the compare value. If the compare value content is set to a system pointer present, the compare value length must be at least 16, and the system pointer must be located in the first 16 bytes of the compare value. The operand must be 16-byte aligned. If the compare value is not required and is not present, the event monitor receives signals regardless of the signaled compare value. Chapter 24. *Event Specifications* defines the appropriate compare value length for machine events.

Authorization Required

- Operational
  - Contexts referenced for address resolution
  - Program referenced as event handler

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
14 Event Management		
01 Duplicate event monitor	X	
03 Machine event requires compare value	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
38 Template Specification		
01 Template value invalid	X	

RETRIEVE EVENT DATA (RETEVTD)

Op Code (Hex)	Operand 1
0375	Receiver

Operand 1: Space pointer.

Description: The instruction retrieves the event-related data associated with a signaled event monitor and places it in the specified space object.

If an event handling program does not retrieve the event-related data before it returns or terminates, the signal and event-related data are lost. This instruction causes the event-related data to be purged and decrements the signals pending count.

If the instruction is issued from a program that is not an event handler, the number of bytes available for retrieval entry is set to binary 8.

Operand 1 defines a template in which the event-related data is to be placed. The receiver must be 16-byte aligned.

The following data is placed in the template by the instruction:

• Template size specification	Char(8)
– Number of bytes provided for retrieval	Bin(4)
– Number of bytes available for retrieval	Bin(4)
• Reserved (binary 0)	Char(2)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length (value of 0 denotes the absence of a compare value)	Bin(2)
• Compare value	Char(32)

- Origin of signal Char(1)
  - Hex 80=Signal by machine
  - Hex 00=Signal by Signal Event instruction

- Reserved (binary 0) Char(1)

- Event-specific data length Bin(2)

For short form event monitors, the event-specific data length value is 0 and the following attributes are not supplied:

- Signals pending count Bin(4)

- Time of event signal Char(8)

  This is a 64-bit field representing an unsigned binary value where bit 41 is equal to 1024 microseconds.

- Process (causing signal—denoted by process control space pointer) System pointer

  This entry is set to binary 0 if the event signal is not related to a process action. For example, this attribute is set to binary 0 for a timer event.

  This entry is set to binary 0 if the signaling PCS (process control space) does not exist when the data is retrieved.

- Event-specific data Char(*)

The first 4 bytes of the retrieved output identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. If fewer than 8 bytes are available in the space identified as the receiver operand, a materialization length exception is signaled. The second 4 bytes of the retrieved output identify the total number of bytes available to be retrieved. The instruction retrieves as many bytes as can be contained in the area specified as the receiver. If the byte space identified by the receiver is greater than that required to contain the information requested for retrieval, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled in the event that the receiver contains insufficient area for the retrieval.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

If the short form option is selected, the signals pending count, time of event signal, process control space pointer, size of event-specific data, and event-specific data entries are not made available.

When the compare value length is less than 32, the excess bytes in the 32-byte compare value are unpredictable.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
38 Template Specification		
03 Materialization length exception	X	

SIGNAL EVENT (SIGEVT)

Op Code (Hex)	Operand 1
0345	Signal event template

Operand 1: Space pointer.

Description: The instruction causes an event to be signaled. The instruction also causes any event monitor currently associated with existing processes to be located, signals these event monitors, and passes the event-related data to them.

Operand 1 specifies the event qualifications, the process to be signaled, the conditional signal mask, and the event-related data. The format is as follows:

• Template size specification	Char(8)
– Number of bytes provided	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Reserved (binary 0)	Char(8)
• Process to signal	System pointer
• Option indicators	Char(2)
– Signal domain	Bit 0
0 = Machine-wide domain	
1 = Process domain	
– Compare value content	Bit 1
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 2-15
• Conditional signal mask	Char(2)
• Reserved (binary 0)	Char(4)
• Size of event-specific data	Bin(2)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length (value of 0 denotes the absence of a compare value)	Bin(2)
• Compare value	Char(32)
• Event-specific data	Char(*)

An event class value of hex 0000 is invalid.

An event type value of hex 00 is invalid.

An event subtype value of hex 00 is invalid.

Events can be signaled directly to a process by providing addressability to the process control space as the process to signal attribute of the Signal Event instruction template. If the event is to be signaled directly to a process, the signal domain must be set to process, and the system pointer addressing the process control space must be supplied. If the process control space is not currently associated with a process, the process control space not associated with a process exception is signaled. If the signal domain is machine-wide, then the process to signal entry is ignored.

A value of binary 0 in the conditional signal mask results in the event being unconditionally signaled. If the value is nonzero, the conditional signal mask is ANDed with the process signal event control mask with a nonzero result causing the event to be signaled. If the result is 0, the event is not signaled. (See the Initiate Process instruction in Chapter 11. *Process Management Instructions*, for a description of the signal event control mask.)

If no compare value is specified on the signal, then only event monitors monitoring the event identification without a compare value will be signaled. The compare value presence is denoted by the compare value length greater than 0 and less than or equal to 32 characters. If a compare value is specified, then event monitors monitoring the event will be signaled if the compare value length in the signaled event is greater than or equal to the compare value length in the event monitor and the compare values match for as many bytes as specified in the event monitor. The event monitor is also signaled when it does not specify a compare value if the event IDs match. If the compare value content is set to system pointer present, the compare value length must be at least 16 bytes, and the system pointer must be located in the first 16 bytes of the compare value.

Since this instruction deals with one process acting upon another process, a portion of the function is performed under control of the issuing process and the remainder of the function is performed under control of the target process. When control is returned to the issuing process, the function may not have been performed in its entirety.

A timer event (class 0014) cannot be signaled explicitly through the use of this instruction.

Authorization Required

- Retrieve
  - Context referenced for address resolution

Lock Enforcement

- Materialize
  - Context referenced for address resolution

Events

nnnn Any machine or user-signaled event

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference
0301 Invocation reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
14 Event Management		
06 Signal time event invalid	X	
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
28 Process State		
02 Process control space not associated with a process	X	
2A Program Creation		
05 Invalid op code extender field		X
06 Invalid operand type	X	
07 Invalid operand attribute	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
38 Template Specification		
01 Template value invalid	X	

TEST EVENT (TESTEVT, TESTEVTB, or TESTEVTI)

Op Code (Hex)	Extender	Operand 1	Operand 2
10FA	None		
1CFA	Branch option	Event-related data	Event monitor template
18FA	Indicator option		

Operand 1: Space pointer.

Operand 2: Character(48) scalar or null (fixed-length).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. *Introduction* for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The instruction tests the signaled flag of the event monitor that matches the event identification, compare value length, and compare value specified by the operand 2 template. If the event monitor has been signaled, the instruction materializes the event-related data into the area specified by operand 1.

If operand 2 is null, the instruction locates the highest priority signaled event monitor associated with the process.

If operand 2 is null and no event monitors are currently active, the not signaled condition is returned and no event data is returned.

The template addressed by operand 2 must be 16-byte aligned and have the following format:

• Option indicators	Char(2)
– Compare value content	Bit 0
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 1-15
• Reserved (binary 0)	Char(8)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length	Bin(2)
• Compare value	Char(32)

If compare value content is set to system pointer present, the compare value length must be at least 16 and the system pointer must be located in the first 16 bytes of the compare value.

If no event monitor associated with the process has the matching attributes of event identification, compare value length, and compare value, the event monitor not present exception is signaled.

If the compare value length entry is 0, the instruction ignores the compare value entry. The requirement of the instruction is then met by a corresponding event identification.

If an event monitor in the signaled state is found, the instruction causes the event-related data to be moved to the area located by operand 1 and decrements the signals pending count by 1. Operand 1 is unchanged if no event monitors are in the signaled state.

If branch options are specified, control flow may be modified depending on whether the specified event monitor is in the signaled or not signaled state. If branch options are not specified for the instruction, control is returned to the next sequential instruction.

The operation is independent of the enabled/disabled state of the referenced event monitor or the masked/unmasked state of the process.

The receiver must be 16-byte aligned.

The following data is placed in the operand 1 space when the instruction is executed:

• Template size specification	Char(8)
– Number of bytes provided for retrieval	Bin(4)
– Number of bytes available for retrieval	Bin(4)
• Reserved (binary 0)	Char(2)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length (value of 0 denotes the absence of a compare value)	Bin(2)
• Compare value	Char(32)
• Indicators	Char(2)
– Origin of signal	Char(1)
0 = Signaled by machine	
1 = Signaled by Signal Event instruction	
– Compare value content	Bit 1
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 2-15
• Reserved (binary 0)	Char(7)
• Event-specific data length	Bin(2)

For short form event monitors, the event-specific data length value is 0 and the following attributes are not supplied:

• Signals pending count	Bin(4)
• Time of event signal	Char(8)

This is a 64-bit field representing an unsigned binary value where bit 41 is equal to 1024 microseconds.

- Process (causing signal–denoted by process control space pointer) System pointer

  This entry is set to binary 0 if the event signal is not related to a process action. For example, this attribute is set to binary 0 for a timer event.

  This entry is set to binary 0 if the signaling PCS (process control space) does not exist when the data is retrieved.

- Event-specific data Char(*)

The first 4 bytes of the retrieved output identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. If fewer than 8 bytes are available in the space identified as the receiver operand, a materialization length exception is signaled. The second 4 bytes of the retrieved output identifies the total number of bytes available to be retrieved. The instruction retrieves as many bytes as can be contained in the area specified as the receiver. If the area identified by the receiver is greater than that required to contain the information requested for retrieval, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled in the event that the receiver contains insufficient area for the retrieval.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Event monitor is in the signaled or not signaled state.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
14 Event Management			
02 Event monitor not present		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
• 20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
05 Invalid op code extender field			X
06 Invalid operand type	X	X	X
07 Invalid operand attribute	X		X
08 Invalid operand value range	X	X	X
09 Invalid branch target operand			X
0C Invalid operand ODT reference	X	X	X
0D Reserved bits are not zero	X	X	X
2C Program Execution			
04 Invalid branch target			X
32 Scalar Specification			
02 Scalar attributes invalid	X	X	
03 Scalar value invalid		X	
38 Template Specification			
03 Materialization length exception	X		

WAIT ON EVENT (WAITEVT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0344	Event-related data	Event monitor template	Time-out value	Wait options

Operand 1: Space pointer.

Operand 2: Character scalar (fixed-length).

Operand 3: Character(8) scalar (fixed-length).

Operand 4: Character(3) scalar (fixed-length).

Description: The executing process is placed in the wait state until an event is signaled to an event monitor identified by operand 2 or until the time-out value elapses. By waiting for an event to occur, the instruction allows synchronization of the process with an external source.

The instruction can specify a time-out value (operand 3) which, when exceeded, causes the waiting process to be made eligible for the processor resource and has an exception signaled to the instruction. A default time out value is alternatively supplied at process initiation time.

Event monitors have a priority associated with them. The priority defines if the waiting process should be made eligible for the processor in order to handle events of equal or higher priority than the event that the process is waiting for. If the waiting process is monitoring events of lower priority than the event that it is waiting for, the process remains in the wait state until the event that it is waiting for occurs or the time-out value is reached. All of the event monitors present in the operand 2 template must be of equal priority. If the event monitors do not have the same priority associated with them, a scalar value invalid exception is signaled. The exception data returned contains the first event monitor specification from operand 2 with a different priority.

All the event monitors in the operand 2 template must be of equal priority. If the event monitors in operand 2 do not have the same priority, a scalar value invalid exception is signaled and the exception data returned contains the first event monitor specification from operand 2 that has a different priority.

The event monitor template addressed by operand 2 is used to locate an event monitor that is associated with the process and has matching event ID, compare value length, and compare value. If a matching event monitor is not found, the event monitor not present exception is signaled.

The format of the operand 2 template must be 16-byte aligned and have the following format:

• Option indicators	Char(2)
– Compare value content	Bit 0
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 1-15
• Reserved (binary 0)	Char(8)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length	Bin(2)
• Compare value	Char(32)

If the compare value content is set to system pointer present, the compare value length must be at least 16, and the system pointer must be located in the first 16 bytes of the compare value.

If the compare value length entry is 0, the instruction ignores the compare value.

If the number of event monitors entry is 0, the wait is completed by the signaling of any event monitor that has no event handler. The signaling of an event monitor which has an event handler causes the event handler to be invoked, but the wait is not completed. Either of the following conditions causes the wait to be completed, and control is passed to the instruction following the Wait On Event instruction:

- If one or more event monitors (each having no event handlers) are in the signaled state, the highest priority event monitor completes the wait.

- If no event monitors are in the signaled state, the first event monitor (having no event handler) to be signaled completes the wait.

If the number of event monitors entry is 1, the wait is completed only by the signaling of the specified event monitor. The signaling of any other event monitor does not complete the wait but does cause the action specified by the event monitor to be performed (invoking an event handler if the event is of higher priority or recording the signal), and the wait is resumed. If the signaled event monitor has an event handler specification, the event handler is given control. The wait is completed when the event handler returns control and control is passed to the instruction following the Wait On Event instruction. Operand 1 is not modified by the instruction. If the signaled event monitor has no event handler, the data associated with the occurrence of the event is stored in the area designated by operand 1, the wait is completed, and control is returned to the instruction following the Wait On Event instruction.

Unless the short form option is used by the event monitor, the receiver must be 16-byte aligned.

The following data is placed in the space object when the wait is completed by an event and no event handler is present:

• Template size specification	Char(8)
– Number of bytes provided for retrieval	Bin(4)
– Number of bytes available for retrieval	Bin(4)
• Reserved (binary 0)	Char(2)
• Event identification	Char(4)
– Event class	Char(2)
Bit 0 = 0–Machine event	
Bit 0 = 1–User event	
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length	Bin(2)
• Compare value	Char(32)

• Indicators	Char(2)
– Origin of signal	Bit 0
0 = Signaled by machine	
1 = Signaled by Signal Event instruction	
– Compare value content	Bit 1
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 2-15
• Event-specific data length	Bin(2)

For short form event monitors, the event-specific data length value is 0 and the following attributes are not supplied:

• Reserved (binary 0)	Char(4)
• Signals pending count	Bin(4)
• Time of event signal	Char(8)
• Process (causing signal–denoted by process control space pointer)	System pointer

This entry is set to binary 0 if the event signal is not related to a process action. For example, this attribute is set to binary 0 for a timer event.

This entry is set to binary 0 if the signaling PCS (process control space) does not exist when the data is retrieved.

• Event-specific data	Char(*)

Operand 3 is a character(8) scalar specifying a realtime interval that the process will wait for the event to occur. If the event does not occur within the interval, a wait time-out exception is signaled, and the process is taken out of the wait. The machine uses operand 3 as a 64-bit unsigned binary field (bit 41 has a value of 1024 microseconds). If time interval is 0, the process default wait time-out value is used. If the wait time-out value is also 0, a wait time-out exception is signaled immediately. The maximum wait time-out allowed is a value equal to (2^{48} - 1) microseconds. Any value that indicates more time than the maximum wait time-out causes the maximum wait time-out to be used. If the wait is timed and the wait time limit is exceeded, then operand 1 is not modified by the instruction.

Operand 4 is a character(3) scalar specifying the access state modification option. The operand has the following values and meaning:

• Number of event monitors	Bin(2)
• Wait options	Char(1)
– When entering event wait	Bit 0
0 = Access state is not modified.	
1 = Access state is modified.	
– When leaving event wait	Bit 1
0 = Access state is not modified.	
1 = Access state is modified.	
– Time-out option	Bit 2
0 = Wait for specified time, then signal time-out exception.	
1 = Wait indefinitely	
– Reserved (binary 0)	Bits 3-7

The number of event monitors denotes the number of event monitor specifications in the operand 2 template. If this value is 0, operand 2 is ignored. The value specified for number of event monitors can be from 0 to 16. If any other value is specified, an exception is signaled.

Operand 4 has no effect if the process instruction wait access state control attribute specifies that access state modification is not allowed. If the process attribute value specifies that access state modification is allowed and the wait on event access state modification option is modify access state, the process access group defined for the process has its access state modification performed by the machine.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	**Operands** **1**	 **2**	 **3**	 **4**	 **Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
14 Event Management					
02 Event monitor not present		X			
04 Wait on event attempted while masked					X
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	
03 Object suspended	X	X	X	X	
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X		X	X	
0A Invalid operand length	X	X	X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
32 Scalar Specification					
02 Scalar attributes invalid		X	X	X	
03 Scalar value invalid			X	X	
38 Template Specification					
03 Materialization length exception	X				
3A Wait Time-out					
03 Event				X	

Chapter 16. Data Base Management Instructions

This chapter describes the instructions used for data base management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. (instruction Summary.

ACTIVATE CURSOR (ACTCR)

Op Code (Hex)	Operand 1	Operand 2
0402	Cursor	Activation template

Operand 1: System pointer.

Operand 2: Space pointer or null.

Description: This instruction connects a previously created cursor to a process, allowing data base operations to be performed with that cursor. The cursor identified by operand 1 is temporarily modified with the replacement values as specified by operand 2.

The data spaces specified in operand 2 are implicitly locked LSRD (lock shared read) or as indicated in the activation template. The data space index specified in operand 2 is implicitly locked LSRD. The secondary data space indexes visible beneath the join cursor are implicity locked LSRD.

The cursor is implicitly locked LEAR (lock exclusive allow read) by the machine. Locking the cursor, data spaces, and data space index prevents them from being destroyed while in use.

An activated cursor can be operated on only by the process that activated it. Activating a cursor prevents any data base operations (except Create Duplicate Object and Materialize Cursor Attributes instructions for the creation template) from accessing the cursor unless they are issued by the activating process.

Locking the data spaces and data space index(es) prevents them from being destroyed while the cursor is activated.

The cursor may be either a permanent or a temporary object and must not be currently activated. The resulting activated cursor does not address an entry for retrieval and has no locked entries associated with it.

The format of the cursor activation template is as follows:

• Data space list pointer	Space pointer
• Length of data space list	Bin(2)
• Cursor attributes	Bin(2)
– Reserved (binary 0)	Bit 0
– Data space index	Bit 1*
– Replace values	Bit 2
0 = Use original cursor values.	
1 = Use replacement cursor values for the activation.	
– Disregard data space index	Bit 3
0 = Activation of the cursor uses the data space index over which it was created.	
1 = Activation of the cursor does not use the data space index over which it was created.	
– Reserved (binary 0)	Bit 4
– Second activation template extension indicator	Bit 5
– Extended activation functions indicator	Bit 6
– Return activation statistics	Bit 7
– Processing mode	Bits 8-9
Index indicator	Bit 8
0 = Random (or no index)	
1 = Sequential	
Data indicator	Bit 9
0 = Random	
1 = Sequential	
– Ensure activity	Bit 10
0 = Ensure data space entries instruction will not be used.	
1 = Ensure data space entries instruction will be used.	
– Reserved (binary 0)	Bits 11-15

• Unit of transfer	Bin(2)
• Locked entry wait time	Char(8)
• Extended activation functions	Char(2)
– Retrieve indicator	Bit 0
– Update indicator	Bit 1
– Delete indicator	Bit 2
– Insert indicator	Bit 3
– Reserved (binary 0)	Bits 4-7
– Implicit lock indicator	Bits 8-10
Hex 00 = LSRD lock	
Hex 01 = LSRO lock	
Hex 02 = LSUP lock	
Hex 03 = LEAR lock	
Hex 04 = LENR lock	
All other values are reserved.	
– Reserved (binary 0)	Bits 11-15
• Activation statistics	Char(8)
– Number of entries in data spaces	Bin(4)*
– Number of entries in data space index	Bin(4)*
• Second activation template extension functions	Char(2)
– Join default values options	
Default values prohibited	Bit 0
Default values allowed	Bit 1
Default values required	Bit 2
Reserved (binary 0)	Bit 3-15
• Data space number to start providing default values	Bin(2)
• Reserved (binary 0)	Char(20)

Notes:

1. The cursor activation template and data space list must each be aligned on a multiple of 16 bytes.
2. The value of the entry shown here with an asterisk (*) is ignored by this instruction.
3. This template is a subset of the create cursor template.

The entry identified as data space list pointer must provide a space pointer to a list of system pointers. Each of these system pointers must address a data space. The length of data space list indicates the number of bytes in the data space list and must be a multiple of 16 bytes.

The Activate Cursor instruction allows the user to specify a subset of the data spaces that are associated with the cursor to be selected for activation. This is not allowed when the cursor type is join. Each system pointer that identifies a data space that is to be put in use under this cursor must occupy the same position in the list that it occupied when the cursor was created by the Create Cursor instruction. To identify data spaces that are not to be used in this cursor activation, 16 bytes of 0's must be placed into the list in place of that data space's system pointer. If the entire data space list contains 0's, then a pointer does not exist exception is signaled.

A zero value in the length of data space list entry indicates all data spaces associated with the cursor are to be put in use and the pointer to the data space list is ignored. If operand 2 is null, all data spaces associated with the cursor are put in use and no replacement values are applied.

If the replace values entry is 1, the new values for processing mode, ensure activity, unit of transfer, and locked entry wait time replace those in the cursor during this activation. See the Create Cursor instruction for definitions of these fields.

A disregard data space index value of binary 1 indicates that this activation of the cursor does not result in the use of the data space index over which the cursor was created. No check is made to ensure the validity, damage, or suspended state of the data space index. The only operations allowed for the activation of the cursor are those which would be allowed if the cursor had been created directly over the data spaces identified for use through this cursor. A value of binary 0 causes the cursor to use the data space index over which it was created. This field does not apply to the data space indexes defined over secondary data spaces under a join cursor. If the cursor was not created over a data space index, this field is ignored.

A value of binary 1 in the second activation template extension field indicates that the second extension of the activation template has been provided and will be processed. The presence of the second activation template extension requires the extended activation function indicator to be binary one.

A value of binary 1 in the extended activation functions indicator field indicates that the extended activation functions are to be processed for this activation of the cursor. If this field contains a binary 0, the extended activation functions are ignored.

A value of binary 1 in the return activation statistics field indicates that the system should return the proper values in the activation statistics fields. If this field contains a value of binary 0, the activation statistics are not returned and the values provided in the input template are not modified. If this field contains a binary 1 but the extended activation functions indicator field has a value of binary 0, a template value invalid exception is signaled on the return activation statistics field.

The extended activation functions are as follows:

- A retrieve indicator value of binary 1 indicates that data space entries are retrieved using this activation of the cursor. This includes use of this cursor as the source operand for the Copy Data Space Entries instruction. A value of binary 0 indicates that data space entries are not retrieved through this activation of the cursor. If an attempt is made to retrieve data space entries through this activation of the cursor, an invalid data base operation exception is signaled. The update, delete, and insert indicators may not be specified if the cursor is a join type.

- An update indicator value of binary 1 indicates that data space entries are updated in data spaces using this activation of the cursor. A value of binary 0 indicates that data space entries are not updated through this activation of the cursor. If an attempt is made to update data space entries through this activation of the cursor, an invalid data base operation exception is signaled.

- A delete indicator value of binary 1 indicates that data space entries are deleted from data spaces using this activation of the cursor. A value of binary 0 indicates that data space entries are not deleted through this activation of the cursor. If an attempt is made to delete data space entries through this activation of the cursor, an invalid data base operation exception is signaled.

- An insert indicator value of binary 1 indicates that data space entries are inserted into data spaces using this activation of the cursor. This includes use of this cursor as the receiver operand for the Copy Data Space Entries instruction. A value of binary 0 indicates that data space entries are not inserted through this activation of the cursor. If an attempt is made to insert data space entries through this activation of the cursor, an invalid data base operation exception is signaled. The update, delete, and insert indicators may not be specified if the curosr is a join type.

Note: If the extended activation functions field has a value of binary 0, the retrieve indicator, insert indicator, delete indicator, and update indicator fields are ignored and all operations are allowed through this activation of the cursor.

- The implicit lock indicator value indicates the level of the implicit locks to be obtained on the data spaces identified for use through this cursor. If the extended activation functions indicator has a value of binary 0, this field is ignored and the data spaces identified for use through this cursor will be implicitly locked LSRD. A LSRD lock or LSRO lock remains with the data spaces until a data space entry in the data space is locked through this activation of the cursor (the lock is changed to LSUP until all entries are removed from the cursors' locked entry queue at which time it is returned to its initial lock state).

- If the activation statistics have been requested for this activation, the number of entries in data spaces field contains the number of data space entries, exclusive of deleted data space entries, in the data spaces identified for use through this cursor. When the cursor type is join, only the data space entries from the primary data space under the join cursor will be included. If the cursor is activated over a data space index, this field contains a value of 0. The number of entries in the data space index field contains the total number of data space entries referenced by the data space index that the cursor is activated over. Only those data space entries in data spaces identified for use through this cursor are included. When the cursor type is join, only the data space entries in the primary data space will be included. If the cursor was not created over a data space index or the disregard data space index option has a value of binary 1, this field contains a value of 0. If the activation statistics are not requested, these fields are not modified.

When performing a join operation and the join is not satisfied due to the absence of a qualifying entry in the joined-to data space, the field values of the default entry of the joined-to data space may be substituted. All mapping/conversion, intermediate buffer mapping, selection, and group-by processing will be performed using the default entry values.

A value of binary one in the default values prohibited field specifies the default entry values may not be substituted.

A value of binary one in the default values allowed field specifies the default entry values may be substituted.

A value of binary one in the default values required field specifies the resulting joined record may not be used unless default values have been substituted for at least one data space participating in the joined record.

Default values allowed may not be specified if default values prohibited has been specified. Default values required may not be specified unless default values allowed has been specified. These options are ignored for non-join type cursors.

The data space number to start providing default values field specifies which data space (as defined by the data space list on Create Cursor may provide default values (as defined by the join default values options). All succeeding data spaces in the list may likewise provide default values. The preceding data space in the list may not provide default values. A value of binary zero or binary one in this field is not allowed. This field is ignored when the cursor is not allowed. This field is ignored when the cursor is not a join type cursor or default values prohibited is specified.

The authority available to the process for each data space to be referenced by a cursor is determined at the time the cursor is activated. After activation of the cursor, references through the activated cursor do not take into consideration any further changes in the authority environment (adopted user profiles, granting or retracting authority). The authority stored at activate time is the sum of the following authority sources:

- Authority stored in a system pointer (in data space list)
- Public authorization
- Authority of the process user profile
- Authority of the current adopted and/or propagated user profiles

Authorization Required

- Operational
  - Operand 1
  - Data spaces referenced by operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Implicit locks
  - Cursor is implicitly locked LEAR.
  - Data spaces referenced are implicitly locked according to the extended activation function options. This is either LSRD, if the cursor is not activated with the extended activation function option, or the level indicated in the activation template.
  - Data space index referenced is implicitly locked LSRD.
  - Secondary data space indexes visible beneath a join cursor are implicitly locked LSRD.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port because unusable

001C Journal space
    0301 Threshold reached

Exceptions

Exception	**Operands** 1	**Operands** 2	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X

Exceptions

Exception	**Operands** 1	**Operands** 2	**Other**
12 Data Base Management			
07 Data space index invalid			X
16 Data space not addressed by cursor		X	
22 Data space index with selection routine build termination			X
33 Data space index with non-user exit S/O routine build termination			X
39 Derived field operation error during build index			X
1A Lock State			
01 Invalid lock state	X	X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
06 Machine lock limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
04 Object not eligible for operation	X		
05 Object not available to process	X	X	X
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
06 Instruction cancellation			X
07 Instruction termination			X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
30 Journal Management			
02 Entry not journaled			X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
01 Template value invalid		X	

COPY DATA SPACE ENTRIES (CPYDSE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
048F	Cursor (receiver)	Option template	Cursor (source)

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: System pointer.

Description: All or part of the entries in the data space referenced through the operand 3 cursor are copied into the data space referenced through the operand 1 cursor according to the specifications provided in the options template (operand 2). Operands 1 and 3 may indicate that the same data space is to be used as both source and receiver. In this case, the result of the copy is placed in the data space at the completion of the operation. The data space entries and data space referenced through the operand 3 cursor are left unchanged. If a data space index is specified in the options template, the data space entries are copied into the receiving data space in the order they are referenced by the data space index. Otherwise, the entries are copied in ordinal entry sequence into the receiver. The template can also specify both start and stop relative entries or keys. The copy can be limited to a number of entries to be copied. The copy can optionally skip deleted entries. The copied data space entries can be added to the end of the receiver data space or the receiver data space may be optionally reset by the copy. No input or output cursor mapping can be performed.

The format of the copy options template is as follows:

• Copy options	Char(2)
– Remove deleted entries	Bit 0
– Data space index retrieval	Bit 1
– Reset receiving data space	Bit 2
– Reserved (binary 0)	Bits 3-15
• Copy specifications	Char(2)
– Starting entry specified	Bit 0
– Ending entry specified	Bit 1
– Entry limit specified	Bit 2
– User entry buffer specified	Bit 3
– Reserved (binary 0)	Bits 4-15
• Number of entries copied	Bin(4)
• Data space entry last processed	Bin(4)
• Maximum number of entries	Bin(4)
• Source data space number	Bin(2)
• Receiver data space number	Bin(2)
• Starting ordinal entry number	Bin(4)
• Ending ordinal entry number	Bin(4)
• Starting key field count	Bin(2)
• Ending key field count	Bin(2)
• Reserved (binary 0)	Char(12)
• Starting key	Space pointer
• Ending key	Space pointer
• User buffer entry	Space pointer

The copy options template must be aligned on a 16-byte boundary.

If the remove deleted entries field has a value of binary 1, deleted entries are not copied into the receiving data space. This field is ignored if the data space index retrieval option is used for retrieving the entries.

A data space index retrieval field value of binary 1 indicates the data space index referenced through the operand 3 cursor is to be used to order the retrieval of entries from the designated source data space. If the data space index has a selection routine, those entries omitted from the data space index are not copied to the receiver. As the entries are placed into the receiving data space (if the same as the source data space), the data space index is updated to reflect the new organization of the data space entries. The data space index must be valid. If this field has a value of binary 0, the data space entries are retrieved in ordinal number sequence.

A reset receiving data space value of binary 1 indicates the data space to receive the entries is to be reset to an empty status before any of the copied entries are added. If the receiving data space and the source data space are the same, this field must be binary 1. See the Data Base Maintenance instruction in this chapter for details of the operation. If a value of binary 0 is specified, the copied entries will be added to the end of the receiving data space.

If the starting entry specified field has a value of binary 0, the copy retrieves entries from the source data space beginning with the ordinal entry number equal to 1. If the data space index is to be used for retrieval, the data space entry identified by the first entry in the data space index becomes the first entry retrieved. In either case, the copy continues through the data space or data space index sequentially until terminated. If this field contains a binary 1 and entry retrieval is not through a data space index, the copy begins with the data space ordinal entry number specified in the starting ordinal entry number field. If the field contains a binary 1 and entry retrieval is through a data space index, the starting key and the starting key field count are used to determine the first entry. The data space index is searched for the first data space entry that has a key that is equal to or after the specified argument key. If the field contains a binary 1, retrieval is through a data space index, and the starting key pointer has a value of binary 0's, the key of the data space entry designated by the starting ordinal entry number will be used as the first entry. Subsequent retrievals are performed sequentially through the data space index. In this case, if the designated entry has been omitted from the data space index or is deleted, an exception is signaled.

If the ending entry specified field has a value of binary 1, the copy attempts to retrieve entries until end of path is encountered. If this field is a binary 1 and the retrieval of entries is not through a data space index, the copy does not retrieve any entries that have an ordinal number that is greater than the ending ordinal entry number. If this field is binary 1 and entries are being retrieved through a data space index, the copy terminates when an entry is retrieved with a key that collates after the key defined by the ending key and the ending key field count. If this field has a value of binary 1 and the ending entry logically precedes the starting entry, a template value invalid exception is signaled for the ending key field.

Note: If an error is incurred while creating either of the argument keys, a key mapping error exception is signaled, and the instruction is terminated before any entries are copied. If either key field count contains a value of 0, only leading fork characters are used to determine the key. If either key field count specifies fewer than the actual number of fields represented in the data space index, then a truncated generic key is generated. Trailing fork characters are always used to generate the key.

If the entry limit specified field has a value of binary 1, the copy inserts up to the number of entries specified in the maximum number of entries field and then terminate the copy. If the field has a value of binary 0, no limit is placed on the number of entries to copy. If the source data space and receiving data space are the same, then this field must contain a binary 0.

If the entries retrieved from the source data space are shorter than entries inserted into the receiver data space, the remainder of each inserted entry is filled with data acquired from the corresponding positions of the user buffer entry (if provided). When no such user buffer entry has been provided, the remaining portion of each inserted entry is padded with binary 0's.

Upon completion of the instruction, the number of entries copied field contains the total number of data space entries that were inserted into the receiving data space. This count includes deleted entries if they were copied to the receiver. The data space entry last processed field contains the ordinal entry number of the last data space entry successfully referenced in the source data space before the instruction was completed.

The maximum number of entries indicates the upper limit on the number of entries to be inserted into the receiving data space. If the entry limit specified field is binary 1, this field must not contain a negative value, otherwise, it is ignored.

The source data space number designates which data space referenced by the operand 3 cursor is to be the source data space. This entry corresponds to the position of the data space in the corresponding data space pointer list associated with the cursor. This data space must be in the cursor's data space list after cursor activation. A value of 2 indicates the second data space in the list, for example.

The receiver data space number designates which data space referenced by the operand 1 cursor is to be the receiving data space. This entry corresponds to the position of the cursor's data space in the corresponding data space pointer list associated with the cursor. A value of 2 indicates the second data space in the list, for example. The receiver data space must not be in use by any process. If the data space is in use by any process (including the same process), an object not eligible for operation is signaled.

The starting ordinal entry number indicates which entry is to be retrieved from the source data space first. It corresponds to the ordinal entry number of the desired entry in the source data space. If the starting entry specified field is binary 1 and the retrieval of entries is not through a data space index, this field must contain a value greater than 0. If the retrieval of the designated entry would result in an end of path condition, no entries are copied. This field is ignored if the starting entry specified field is binary 0 or the retrieval is through a data space index and a key is to be used.

The ending ordinal entry number indicates which entry is to be retrieved last from the source data space. It corresponds to the ordinal entry number of the desired entry in the source data space. If the ending entry specified field is binary 1 and the retrieval of entries is not through a data space index, this field must contain a value greater than 0. This field is ignored if the ending entry specified field is binary 0 or the retrieval is through a data space index.

The starting key field count and ending key field count indicate the number of fields assumed to be in the starting key and ending key values. The key field counts include only data fields supplied by the user of the instruction and do not include fork characters. If the starting entry specified field has a binary 1 value and the retrieval of entries is through a data space index, the starting key field count must be greater than or equal to 0. Otherwise, the starting key and the starting key field count are ignored. If the ending entry specified field has a binary 1 value and the retrieval of entries is through a data space index, the ending key field count must be greater than or equal to 0. Otherwise, the ending key and the ending key field count are ignored.

The user buffer entry, if specified, is used to obtain default values in the physical (not logical) representation for the receiving data space when the source data space does not provide them. It is assumed to be as large as the physical (not logical) representation of each entry that resides in the receiving data space.

The cursors identified by operands 1 and 3 must have been activated to the issuing process. They may not be positioned to an entry in any data space and they may not hold entry locks. If either of the cursors is set or holds entry locks, an object not eligible for operation is signaled. At the completion of the instruction, the cursors are not positioned to any data space entries.

If either of the cursors are of a join type, an operation not valid with join cursor exception is signaled.

If the source cursor has output selection, an object not eligible for operation exception is signaled.

If either of the indicated cursors is under commitment control, an operation not valid under commitment control exception is signaled and the operation is terminated.

If the receiver data space is being journaled, an object not eligible for operation exception is signaled and the operation is terminated.

If this instruction does not complete normally, the entries already copied may be placed into the receiving data space. In the case of a system-generated exception, entries already copied may appear in the receiver (except when the source and receiving data spaces are the same). In the case of a system failure, the normal system recovery facilities control the entries which appear in the data space used as a receiver (except when the source and the receiving data spaces are the same). If the source and receiving data spaces are the same, the data space remains unchanged unless the instruction terminates normally. If a data space index was specified in the template and the source and receiving data spaces are the same, the data space index may be marked invalid if the instruction terminates abnormally. When this instruction completes, the changed system objects are not ensured.

If a data space index was specified in the input template and the source and receiver data spaces are the same, the data space index may be defined and valid over the receiver data space only once.

If the operand 1 cursor has not been activated with an operation intent of insert, or the operand 3 cursor has not been activated with an operation intent of retrieve, then an invalid data base operation exception is signaled.

Authorization Required

- Insert
  - The data space referenced by operand 1
- Retrieve
  - The data space referenced by operand 3
  - Contexts referenced for address resolution
- Delete
  - The data space referenced by operand 1 (reset option)
- Object management
  - The data space referenced by operand 1 (reset option)
  - The data space referenced by operand 3 (if no cursor mapping is specified and the cursor does not have the direct map attribute)

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Implicit locks
  - Data space referenced by operand 3 is locked LEAR
  - Data space referenced by operand 1 is locked LENR

Events

0002 Authorization
    0101 Authorization violation

0008 Data space index
    0301 Data space index invalidated

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port became unusable

001C Journal space
    0301 Threshold reached

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation	X		X	X
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage		X		X
12 Data Base Management				
01 Conversion mapping error				X
02 Key mapping error			X	
04 Data space entry limit exceeded				X
07 Data space index invalid				X
08 Incomplete key description			X	
09 Duplicate key value				X
0F Duplicate key value in uncommitted data space entry				X
21 Unable to maintain unique key DSI				X
23 Data space index select routine failure				X
25 Invalid data base operation	X			
27 Data space index key with invalid floating-point field				X
37 Operation not valid with join cursor	X			
38 Derived field operation error				X
1A Lock State				
01 Invalid lock state	X		X	
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
06 Machine lock limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	X
02 Object destroyed	X	X	X	X
03 Object suspended	X	X	X	X
04 Object not eligible for operation	X		X	X
05 Object not available to process	X		X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X		X	
03 Pointer addressing invalid object	X		X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
06 Instruction cancellation				X
07 Instruction termination				X
2E Resource Control Limit				
02 User profile storage limit exceeded				X
30 Journal management				
02 Entry not journaled				
38 Template Specification				
01 Template value invalid		X		
3E Commitment Control				
10 Operation not valid under commitment control	X		X	

CREATE CURSOR (CRTCR)

Op Code (Hex)	Operand 1	Operand 2
044A	Cursor	Cursor template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: A cursor object is created according to the definition given in the cursor template specified by operand 2, and addressability to the cursor is returned in the system pointer identified by operand 1.

Upon successful completion of the instruction, the created cursor contains addressability to the data space(s) and the primary data space index (if defined) specified in the cursor template.

The format of the cursor template is as follows:

• Template size	Char(8)
– Number of bytes provided by user	Bin(4)*
– Number of bytes that can be materialized	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes	Bit 0
0 = Temporary	
1 = Permanent	
– Space attribute	Bit 1
0 = Fixed-length	
1 = Variable-length	
– Initial context	Bit 2
0 = Addressability is not inserted in context	
1 = Addressability is inserted in context	
– Access group	Bit 3
0 = Not created as a member of an access group	
1 = Created as a member of an access group	
– Reserved (binary 0)	Bits 4-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Transient storage pool selection	Bit 6
0 = Default main storage pool (process default or machine default as specified for main storage pool selection) is used for object.	
1 = Transient storage pool is used for object.	
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Context	System pointer

• Access group	System pointer
• Data space index pointer	System pointer
• Data space mapping templates list pointer	Space pointer
• Data space list pointer	Space pointer
• Length of data space list	Bin(2)
• Cursor attributes	Char(2)
– Second template extension	Bit 0
0 = No second extension	
1 = Second extension of creation template	
– Data space index	Bit 1
0 = No data space index provided	
1 = Access through a data space index	
– Ignored	Bits 2-3*
– First template extension	Bit 4
0 = No extended creation functions	
1 = Extended creation functions	
– Floating-point attributes specified	Bit 5
0 = Use default floating-point attributes	
1 = Use specified floating-point attributes	
– Ignored	Bits 6-7*
– Processing mode	Bits 8-9
Index indicator	Bit 8
0 = Random (or no index)	
1 = Sequential	
Data indicator	Bit 9
0 = Random	
1 = Sequential	
– Ensure activity	Bit 10
0 = Ensure data space entries instruction will not be used.	
1 = Ensure data space entries instruction will be used.	
– Cursor type	Bit 11
0 = Non-join cursor	
1 = Join cursor	
– Group-by processing	Bit 12
0 = Group-by processing not specified	
1 = Group-by processing specified	
– Materialize of cursor not allowed	Bit 13
– Reserved (binary 0)	Bit 14-15
• Unit of transfer	Bin(2)
• Locked entry wait time	Char(8)*
• Extended creation functions	Char(18)
– Floating-point attributes	Char(1)
Floating-point overflow mask	Bit 0
0 = Masked	
1 = Unmasked	
Floating-point underflow mask	Bit 1
0 = Masked	
1 = Unmasked	
Floating-point zero divide mask	Bit 2
0 = Masked	
1 = Unmasked	
Floating-point inexact result	Bit 3
0 = Masked	
1 = Unmasked	
Floating-point invalid operand	Bit 4
0 = Masked	
1 = Unmasked	
Reserved (binary 0)	Bit 5
Floating-point rounding mode	Bits 6-7
00 = Round to positive infinity	
01 = Round to negative infinity	
10 = Round to zero (truncate)	
11 = Round to nearest or even	
– Reserved (binary 0)	Char(17)

• Second creation template extension	Char(192)
– Length of group-by mapping template list	Bin(2)
– Number of translate tables	Bin(2)
– Reserved (binary 0)	Char(12)
– Join definition template pointer	Space pointer
– Group-by processing definition template	Space pointer
– Output selection templates list pointer	Space pointer
– Group-by selection template pointer	Space pointer
– Group-by mapping templates pointerpointer	Space pointer
– Translate table list pointer	Space pointer
– Reserved (binary 0)	Char(48)
– Reserved (binary 0)	Char(32)

Notes:

1. The cursor template, data space list, and mapping templates list must each be aligned on a multiple of 16 bytes.
2. The values of the entries shown here with an asterisk (*) are ignored by this instruction.

The object identification specifies the symbolic name that identifies the cursor within the machine. A type code of hex 0D is implicitly supplied by the machine. The object identification is used to identify the object on materialize instructions as well as to locate the object in a context that addresses the object.

The existence attribute specifies whether the cursor is to be created as temporary or permanent. A temporary cursor, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated. A permanent cursor exists in the machine until explicitly destroyed by the user. A permanent cursor may not be created over a temporary, primary, or secondary index.

If the created object is permanent, it is owned by the user profile governing process execution. The owning user profile is implicitly assigned all private authority states for the object. The storage occupied by the created cursor is charged to this owning user profile. If the created cursor is temporary, there is no owning user profile and all authority states are assigned as public. The storage occupied by the created cursor is charged to the creating process.

A space may be associated with the cursor. The space may be fixed or variable in size. The initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified; the actual size allocated depends on an algorithm defined by a specific implementation. A fixed size space of zero length causes no space to be allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended, this byte value is also used to initialize the new allocation. If no space is allocated, this value is ignored.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created cursor is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

If the access group creation attribute entry indicates that the cursor is to be created in an access group, the access group entry must be a system pointer that identifies the access group in which the cursor is to be created. Since access groups may only be created as temporary objects, the existence attribute entry must be temporary (bit 0 equals 0) when a cursor is created in an access group. If the cursor is not to be created in an access group, the access group entry is ignored.

The performance class parameter provides information that allows the machine to manage the cursor with consideration for the overall performance objectives of operations involving the cursor.

If the data space index attribute specifies that the cursor is to be created over a data space index, the data space index pointer entry must be a system pointer. It must address a data space index that is used in accessing the data spaces through the cursor. If the data space index attribute specifies that the cursor is not to be created over an index, the data space index pointer entry is ignored.

The mapping templates list pointer must address a list of mapping template space pointers, one for each data space system pointer. The data space list pointer must address a list of system pointers, each addressing a data space. The length of data space list entry (which must be a multiple of 16 bytes) specifies the length of each of these lists.

A binary zero in the data space index attribute indicates that the cursor is to be built directly over the data spaces and that the data space index pointer entry is to be ignored. A binary one in the attribute indicates that the cursor is to access the data spaces through the data space index addressed by the data space index pointer.

A value of binary one in the first template extension field indicates that the extended creation functions are to be processed for the creation of this cursor. If this field contains a binary zero, the extended creation functions are ignored.

A value of binary one in the second template extension field indicates that the second creation template extension is to be processed for the creation of this cursor. If this field is binary zero, the second creation template extension is ignored. This field is ignored unless the first template extension field contains a binary one.

A value of binary one in the floating-point attributes specified field indicates that the floating-point attributes field contains the floating-point attributes for this cursor. If this field contains a binary zero, the default floating-point attributes are used. The default floating-point attributes are as follows:

- The floating-point overflow mask field contains a binary one indicating that this exception is unmasked.
- The floating-point underflow mask field contains a binary one indicating that this exception is unmasked.
- The floating-point zero divide mask field contains a binary one indicating that this exception is unmasked.
- The floating-point inexact result mask field contains a binary zero indicating that this exception is masked.
- The floating-point invalid operand mask field contains a binary one indicating that this exception is unmasked.
- The floating-point rounding mode field contains a binary 11 indicating that round to nearest or even is enabled.

If the floating-point attributes are specified field contains a value of binary one and the first extension field contains a value of binary zero then a template value invalid exception is signaled.

The processing mode entry identifies the type of processing to be accomplished with the cursor. This entry indicates whether the access to the data and/or index is random or sequential. This information is used to optimize the internal method for transferring information between main and auxiliary storage for both data spaces and the data space index. The index indicator indicates whether the index (independent of the data) is accessed randomly or sequentially and must be binary 0 if no index is specified in the cursor. The index indicator is used to optimize usage of the index. The data indicator indicates whether the entries (independent of the index) are accessed randomly or sequentially by arrival sequence and is used to optimize their transferral to and from auxiliary storage. If the type of processing is not known, binary 0's should be specified for both.

When the cursor type field indicates a join cursor, the processing mode refers only to the primary data space in the join definition and the data space index addressed by the data space pointer.

If the data indicator field is specified as a binary 1, the unit of transfer argument specifies the minimum number of data space entries that are to be transferred between auxiliary and main storage. The transfer takes place any time an entry residing outside the current transfer block is referenced by the Set Cursor instruction, the Retrieve Sequential Data Space Entries instruction, or the Retrieve Data Space Entry instruction. If the unit of transfer is binary 0 or the data indicator is binary 0, the machine establishes the unit of transfer of 1.

The ensure active attribute allows the cursor user to indicate at creation or activation of the cursor his intent to subsequently use the Ensure Data Space Entries instruction. Performance improvements may be obtained by specifying the intended use of this instruction at these times.

When the cursor type field indicates a join cursor, the unit of transfer argument is meaningful only for the primary data space in the data space list provided. The machine establishes the unit of transfer for the secondary data spaces.

A cursor type value of binary one indicates a join cursor is to be created. When this field contains a binary one, the second extended template field must contain a value of binary one. The join definition is given in the join definition template.

A group-by processing field value of binary one indicates group-by processing is specified. The group-by processing definition template pointer locates the group-by processing definition template. This definition will be associated with this cursor and performed optionally during retrieve sequential DSE.

If the materialize of cursor not allowed cursor attribute contains a binary one, materialize cursor will not allow the creation template to be materialized. This will allow pieces of the cursor creation templates necessary only for materialize cursor to be dropped from the cursor object, therefore, reducing the required space to store the object. This option is required when the second extension of the template is supplied or intermediate mapping table is supplied.

Locked entry wait time is the amount of elapsed time that a Set Cursor instruction is allowed to wait for an entry that is already locked before signaling an exception. Bit 41 of the value is equivalent to 1024 microseconds. If the field is 0, a machine default data base lock wait time-out value of approximately 60 seconds is used.

The system pointers in the data space list identify the data spaces the cursor is to reference. When a cursor is used over multiple data spaces, the data spaces are identified by a data space number derived from their position in the data space list. This data space number is used to uniquely identify each data space whenever the cursor is referenced. The first data space in the list is assigned the number 1, and the nth data space in the list is assigned the number n. For a join cursor, each position in the data space list must contain a system pointer to a data space.

When the cursor type indicates a join cursor, the data space list identifies the data spaces taking part in the join operation.

The join definition array identifies the order of the primary and secondary spaces in the join. The primary will be the first entry in the array, the subsequent entries define the order of the secondaries.

If the cursor is created over a data space index, only a subset of those data spaces visible through the index that are intended to be referenced through the cursor need to be specified in the data space list. For a non-join cursor, the data spaces must appear in the same position in the data space list as they did when the data space index was created. If a data space that is covered by the data space index is not to be referenced through the cursor, 16 bytes of binary 1 must be placed in the data space list in place of the data space system pointer. In the event that the entire data space list contains 0's, a pointer does not exist exception is signaled.

The extended creation functions are as follows:

The floating-point attributes are the floating-point computational attributes that are placed in effect whenever the from data space entry or the from key mapping is executed. The floating-point attributes are as follows:

- A value of binary one for the floating-point overflow mask field indicates that the floating-point overflow exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point underflow mask field indicates that the floating-point underflow exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- **A value of binary one for the floating-point zero divide mask field indicates that the floating-point zero divide exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.**

- **A value of binary one for the floating-point inexact result mask field indicates that the floating-point inexact result exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.**

- **A value of binary one for the floating-point invalid operand mask field indicates that the floating-point invalid operand exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.**

- **The floating-point rounding mode controls how floating-point values are rounded. The Store and Set Computational Attributes instruction provides a definition of the floating-point rounding modes.**

The second creation template extension functions are as follows:

The join definition template pointer locates the join definition template. This pointer is required if the cursor type indicates a join cursor and is otherwise ignored.

The group-by processing definition template pointer locates the group-by processing definition template. This pointer is required if the group-by processing cursor attribute field contains a binary one and is otherwise ignored.

When the group-by selection template pointer contains a null pointer, no selection on group-by results is requested. When this space pointer contains a non-null value, it addresses a selection template to be used on group-by results. This space pointer is ignored if the group-by processing attribute contains a value of binary zero.

When the output selection template list pointer contains a null value, this variety of selection is not requested. Otherwise, for each data space in the data space list, a corresponding space pointer to a selection template exists in the same position in the list of output selection templates. A null pointer in this list indicates no selection template exists for this data space.

Unused positions in the input and output selection template lists are ignored.

The translate table list pointer addresses an array of translate tables, each table 256 bytes in length. These tables are available for use by intermediate mapping when performing the translate operation. The number of tables in the array is designated by the number of translate tables field. A value of binary 0 in this field designates no translate tables are specified and the translate table list pointer is ignored.

The translation table is a 256-byte table of replacement values. The replacement value for a specific byte is located in the table at an offset equal to the bytes' binary value. Refer to the *Create Data Space Index* instruction in this chapter for an example of a translate table.

The group-by mapping templates list pointer must contain a non-null pointer if group-by processing is specified. This field is ignored if group-by processing is not specified.

The length of group-by mapping template list designates the byte length of that list. This byte length must be a multiple to 16 bytes. The field is ignored if group-by processing is not specified.

For each data space referenced by the data space list, there must be a corresponding mapping template space pointer in the same position in the mapping templates pointer list. Each mapping template space pointer must point to a mapping template that defines the view the user is to have of the data space entries that reside in that data space. Unused positions in the mapping templates pointer list are ignored.

The format of the mapping template for data space mapping and group-by mapping is as follows:

• Number of bytes in the mapping template	Bin(4)
• Mapping type	Char(2)
– Input mapping type	Char(1)
– Output mapping type	Char(1)
• Reserved (only specified when optional intermediate mapping is specified–contents ignored)	Char(10)
• Intermediate mapping (optional)	
– Number of intermediate mapping specifications	Bin(2)
– Reserved (binary 0)	Char(12)
– Number of data pointers	Bin(2)
– Data pointers (repeated)	Data pointers
– Intermediate mapping table	Char(0-n)
– Intermediate mapping specification (repeated)	Char(32)
Field description of operand 1	Char(8)
— Operand location type	Bin(2)
Hex 0000 = Data space	
Hex 0001 = Cursor intermediate buffer	
Hex 0002 = Group-by intermediate buffer	
Hex 0005 = Null (type)	
Hex 0006 = Literal (type)	
Remaining codes reserved	
— Field number	Bin(2)
— Start character	Bin(2)
— End character	Bin(2)
or	
— Array position of data pointer	Bin(2)
— Reserved (binary 0)	Char(4)
or	
Reserved (binary 0)	Char(6)
– Result field attributes	Char(5)
— Field type	Bin(2)
— Field length	Bin(2)
— Rounding mode	Char(1)
Hex 02 = Round to zero (truncate)	
Hex 04 = Decimal round remaining codes reserved	
Hex 05 = Not applicable	
Operation	Char(1)
Field description of operand 2	Char(18)
— Operand location/type	Bin(2)
Hex 0000=Data space–location	
Hex 0001=Cursor intermediate buffer–location	
Hex 0002=Group-by intermediate buffer–location	
Hex 0003=Reserved	
Hex 0004=Reserved	
Hex 0005=Null (type)	
Hex 0006=Literal (type)	
Hex 0007=Translate table-type	
Remaining codes reserved	
— Operand description	Char(16)
Field number	Bin(2)
Start character	Bin(2)
End character	Bin(2)
Reserved (binary 0)	Char(10)
or	
Array position of data pointer to literal	Bin(2)
Reserved	Char(14)
or	
Array position of translate table	Bin(2)
Reserved (binary 0)	Char(14)

• Input mapping table (optional)	Char(0-n)
– Number of fields described	Bin(2)
– Field specification (repeated for each field in template)	Char(6)
Field number	Bin(2)
Field attributes	Char(4)
Field type	Bin(2)
Field length	Bin(2)
• Output mapping table (optional)	Char(0-n)
– Number of fields described	Bin(2)
– Field specification (repeated for each field in template)	Char(6)
Field number	Bin(2)
Field attributes	Char(4)
Field type	Bin(2)
Field length	Bin(2)

The mapping template must be aligned on a 16-byte boundary when intermediate mapping is specified.

The number of bytes in the mapping template indicates the total number of bytes included in the number of bytes field, the mapping type field, the optional intermediate mapping tables, the input mapping table, and the output mapping table for this data space.

The input mapping type entry specifies the type of mapping to be used during the mapping of the data from the interface buffer to the data space during insert and update operations. Conversely, the output mapping type entry specifies the mapping type to be used during the mapping of the data from the data space to interface buffer for the retrieve operation. The values that can be associated with the mapping type entries are as follows:

Input Mapping Type

Hex 00 = Direct mapping
Hex 01 = Mapping table provided
Hex 04 = Not applicable

Output Mapping Type

Hex 00 = Direct mapping
Hex 01 = Mapping table provided
Hex 02 = Same as input mapping
Hex 03 = Intermediate mapping table provided

The input and output mapping types allowable with the mapping template are shown below:

Mapping template types:

	Group-by mapping	Data space mapping
Output mapping types:		
Direct mapping	X	X
Mapping table provided	X	X
Same as input mapping		X
Intermediate mapping	X	X
Input mapping types:		
Direct mapping		X
Mapping table provided		X

X = Allowable mapping types

AAC001-0

Direct mapping signifies that the data space entry is to be moved directly to or from the machine interface buffer without conversion or field repositioning. The mapping table provided or intermediate mapping table provided specifies that conversion and/or field repositioning are to be performed as designated by an associated mapping table defined in the mapping template. When same as input mapping is specified, the specifications for input mapping (input mapping type specification and the input mapping table, if specified) are also used for the output mapping function (this does not include intermediate mapping).

The input mapping table is present only if the input mapping type code specifies mapping table provided. Similarly, the output mapping table is present only if the output mapping type code specified mapping table provided. Also, the output intermediate mapping table is present only if the intermediate mapping table provided implies both the output mapping table and the intermediate mapping table are provided.

Whenever output intermediate mapping is specified, the output mapping table must also be supplied and its existence will be implied by the machine.

Whenever the group-by mapping template is specified, the input mapping type must equal hex 04, not applicable. Also, the only allowable output ampping type is intermediate mapping table provided or mapping table provided or direct mapping.

When creating a join cursor, the intermediate mapping table must be specified for each DS in the DS list. The intermediate mapping table associated with the first DS in the DS list will define the first N fields of the join intermediate buffer. The intermediate mapping tables associated with the subsequent data spaces in the DS list define the remaining fields in the join intermediate buffer in the order of the DS's in the DS list.

The output mapping tables of a join cursor define the mapping to occur from the intermediate buffer to the user interface buffer. The field number specified as the source in the intermediate buffer must have been defined by the intermediate mapping table of either the corresponding data space in the data space list or any preceding data space in the DS list.

Each output mapping table describes the receiving user interface buffer beginning at offset zero (first byte of user interface buffer). Output mapping performed on behalf of a join cursor begins with the mapping associated with the first data space in the list and proceeds sequentially through the DS list, performing the output mapping associated with each data space. The offset in the user buffer to which the results are placed can be controlled by use of dummy mapping to skip over result fields associated with previous data spaces in the list.

When creating a non-join cursor, the intermediate mapping table may be specified for a subset of the data spaces in the DS list. There is no relationship between the intermediate mapping tables specified. Each intermediate mapping table defines the fields in the intermediate buffer for that data space.

The field numbers in the intermediate buffer assigned to the corresponding intermediate mapping table may be used as a source field number by that mapping table, provided that field number had been defined by a preceding intermediate mapping specification.

The allowable source field numbers in the group-by intermediate buffer may be a previously defined field number for the results of group-by processing (the fields preceding the fields defined by the group-by intermediate mapping table).

The number of fields entry specifies the number of fields that are to be mapped between the intermediate buffers, user interface buffers, and/or the data space, and must equal the number of field specification entries in the associated mapping table. A value of zero in this field is allowable for input and output mapping tables. An exception is signaled by Insert Data Space Entry, Insert Sequential Data Space Entry, and Update Data Space Entry when attempting to perform those instructions with a number of fields specified as zero on input mapping table. Likewise, an exception is signaled by Retrieve DSE and Retrieve Sequential DSEs when attempting to perform those instructions with number of fields value of zero specified on the output mapping table. It is allowable with a join cursor to specify zero fields on a subset of the output mapping tables supplied. However, at least one output mapping table must specify non-zero fields to map, otherwise, an exception will be signalled by Retrieve DSE and Retrieve Sequential DSE.

The field specification entry is repeated for each field in the template. The order of the field specification entries on the mapping table implicitly specifies the order of the fields in the user interface buffer.

The order of the intermediate mapping specification entries in the intermediate mapping table implicitly specifies the order of the resulting fields in the associated intermediate buffer.

The field number entry is the relative location of the associated field in the data space or intermediate buffer as established by the Create Data Space and Create Cursor instructions. A value of 1 identifies the first field, and a value of n identifies the nth field. When the intermediate buffer is present, it is the source location during output mapping for each field specification entry associated with the output mapping table. The field number must reference a previosusly defined field.

The allowable operand location for operand 1 and operand 2 with the intermediate mapping types follows:

Intermediate mapping types:

Buffer sources:	Group-by	Cursor (output)
DS entry		X
User interface		
Cursor intermediate (output)		X
Group-by intermediate	X	

X = Buffer is allowable source for the associated intermediate buffer mapping type.

AAC002-0

The start character and end character fields are meaningful only for character type fields and specify the byte positions in the designated operand 1 or operand 2 fields. A value of the binary zero in these fields designates the entire field will be used. The end character field must contain a value of binary zero when the start character field contains a binary zero. When the start character field contains a non-zero positive value, the end character field must contain a value at least as great. Negative values in these fields are not allowed. A value of 1 specifies the first byte position and so on. A value of binary zero must be specified for non-character type result fields.

When the operand type is a literal, the operand description references a data pointer to the associated literal.

The result field attributes for intermediate buffer mapping describe the result field in the intermediate buffer associated with this intermediate mapping table. For arithmetic operators, this requires both fields to be defined identically (whether it was defined in the data space at Create Data Space, or previously defined in the intermediate buffer).

Fields are mapped from the machine interface buffer to the data space in the order they are specified in the input mapping template. If more than one field in the machine interface buffer is mapped to the same field in the data space, each field is mapped (and validity checking performed) to the data space entry as it is encountered. After all fields have been mapped, the data space entry fields with duplicate mapping will contain only the value contained in the last machine interface buffer field specified.

The following field types and specification codes are allowed:

Field Type	Specification Code (Hex)
Binary	0000
Floating-point	0001
Zoned decimal	0002
Packed decimal	0003
Character	0004
Dummy	0005

The permissible values for the field length entry vary based on the value of the associated field type entry as follows:

Field Type	Allowed Field Length Values	
Binary	Bytes 1-2	– Length in bytes = Binary 2 or 4
Floating-point	Bytes 1-2	– Length in bytes = Binary 4 or 8
Zoned decimal	Byte 1	– Fractional digits[1] = Binary 0 to total number of digits
	Byte 2	– Total number of digits = Binary 1 to 31
Packed decimal	Byte 1	– Fractional digits[1] = Binary 0 to total number of digits
	Byte 2	– Total number of digits = Binary 1 to 31
Character	Bytes 1-2	– Length in bytes = Binary 1 to 32 767
Dummy	Bytes 1-2	– Length in bytes = Binary 1 to 32 767

The allowable field operations and their definition in the intermediate mapping table follow:

Hex 01 = Concatenate
Hex 02 = Map
Hex 03 = Direct map
Hex 04 = Addition
Hex 05 = Subtraction
Hex 06 = Multiplication
Hex 07 = Division (/)
Hex 08 = Minimum
Hex 09 = Maximum
Hex 0A = Absolute value
Hex 0B = Translate
Hex 0C = Natural logarithm
Hex 0D = Exponential
Hex 0E = Sine
Hex 0F = Cosine
Hex 10 = Tangent
Hex 11 = Cotangent
Hex 12 = Arc sine
Hex 13 = Arc cosine
Hex 14 = Arc tangent
Hex 15 = Hyperbolic sine
Hex 16 = Hyperbolic cosine
Hex 17 = Hyperbolic tangent
Hex 18 = Arc hyperbolic tangent
Hex 19 = Square root
Hex 1A = X taken to the Y power
Hex 1B = Binary OR
Hex 1C = Binary XOR
Hex 1D = Binary AND
Hex 1E = NOT
Hex 1F = Negate

Remaining codes are reserved.

The concatenation operator is valid only for character type fields.

[1] The number of fractional digits to the right of the decimal point.

Refer to the specific computation instruction for restrictions on the use of these field operations (except MAP).

Character fields may not be specified as being mapped to or from any of the numeric field types with the following exception: when defining the intermediate mapping buffer, character fields may be specified as being mapped to or from a zoned field of equal length. Character fields are truncated or padded with blanks on the right when needed. Numeric fields are truncated or padded with zeros on the left or right as necessary.

The dummy field type indicates the number of bytes to be skipped in the interface buffer or intermediate buffer when a data space entry is being mapped to or from that buffer. The dummy type allows the alignment of fields in a buffer for reservation of space in the buffer for performing join operations, or other like functions. When dummy is specified, the field number entry must be zero. The operand 1 and operand 2 field description must be null type when dummy result field type is specified.

The floating point attributes of the cursor are used when performing floating point computations during output intermediate mapping and group-by intermediate mapping.

The rounding mode is meaningful only for decimal field operations and is otherwise ignored.

If the cursor is over a data space index, the keys provided in the interface buffer to set cursor and materialized to the interface buffer by materialize cursor attributes, have the key fields ordered as specified in the data space index (minus the fork characters).

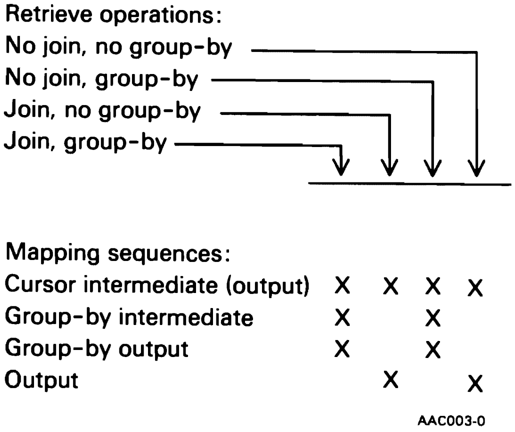
If the data space index does not have intermediate key mapping specified, the key field attributes will have those as specified in the cursor output mapping table. If a key field is referenced more than once in the output mapping template, the first occurrence of the field in the template determines the attributes of the field for those instructions. If the cursor has intermediate mapping associated with the data space, the key field must be defined as a result field in an intermediate buffer field defined by the intermediate mapping table for that data space. The result field must have been defined as the result of the map operator with the data space entry field (used as the key field for the key in the index) as the source of the map. The key field does not undergo the map transformation which may have occurred when mapping from the data space entry to the intermediate buffer and then to the user interface buffer. But rather, the key field undergoes the transformation from the attributes of the data space entry field to the attributes of the result field in the user interface buffer.

If the data space index has intermediate key mapping specified, the key field attributes will have those as specified by the data space index. When providing keys in the user interface buffer to Set Cursor, the key field contents are used with no further alternation (other than the alterations performed internally for ordering, alternate collating and zone/digit force options and, insertion of fork characters).

Fork characters are never present in the interface buffer and are inserted by the machine during construction and maintenance of the index.

The mapping sequences follow:

Output mapping:

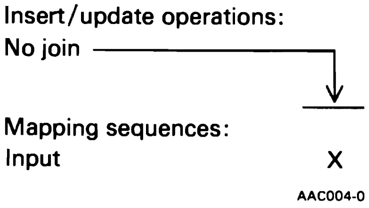


Retrieve operations:

Mapping sequences:	Join, group-by	Join, no group-by	No join, group-by	No join, no group-by
Cursor intermediate (output)	X	X	X	X
Group-by intermediate	X		X	
Group-by output	X		X	
Output		X		X

AAC003-0

Input mapping:



Insert/update operations:

Mapping sequences:	No join
Input	X

AAC004-0

The group-by intermediate buffer field numbers are assigned the number(s) immediately following the number(s) assigned to the group-by result fields. Refer to the group-by template for the possible result fields.

If any of the key fields for a data space does not appear in the output mapping, output intermediate mapping template or the key fields are defined (in the output intermediate template) with a field operation other than map with source field attributes identical to the result field attributes, then the Set Cursor instruction with the rule = key attribute results in an exception, the Materialize Cursor Attributes instruction will not materialize a key and the Set Cursor instruction will not materialize a key.

The format of the join definition template is as follows:

• Number of bytes in template	Bin(4)
• Number of elements in template	Bin(2)
• Reserved (binary 0)	Char(10)
• Definition array	Char(*)
– Join definition (repeated) for each DS	
— Index pointer (current DS)	System pointer
— Offset to join field identities template	Bin(4)
— Data space number in the index data space list (current DS)	Bin(2)
— Data space number of the previous data space (in the cursor list)	Bin(2)
— Reserved (binary 0)	Char(4)
— Reserved (binary 0)	Char(20)

The join definition template must be aligned on a 16-byte boundary.

The format of the join field identities template is as follows:

• Number of field identities	Bin(2)
• Relationship to be applied	Char(1)
– Hex 01 = Equal	
• Reserved (binary 0)	Char(1)
• Field identities (repeated)	Char(*)
– Field number	Bin(2)
– Reserved (binary 0)	Char(2)

The order of the join definitions dictate the order of the join operation (that is, joining is performed in the order the join definitions are provided, the first definition describes the primary data space, the subsequent entries describe the secondary data spaces).

A join definition must be provided for the primary data space under a join cursor as well as all secondary data spaces. When the data space associated with the template is the primary data space, the field data space numbered in the index data space list must equal 1 (if an index is over the primary) and the remaining unreserved fields are ignored.

The current data space is the data space being joined (target DS of a join operation). The previous data space is the source of the joining field value(s). The index pointer addresses a data space index containing composite keys comprised of join fields in the current data space.

The data space number in the index data space list specifies which data space under this index is to be associated with the join operation as the current data space.

The data space associated with the slot in the DS list (for the cursor) which corresponds with this join definition slot, is considered the current data space. The current data space address must be identical with the address of the data space referenced by the data space number in the index data space list.

The data space number of the previous data space indicates which data space under the cursor is to be associated with this join definition as the previous data space. This value must identify either the primary data space or a previously defined secondary data space.

The number of field identities contain the number of elements in the array of field identities.

The field identities template specifies the field identities in the output mapping intermediate buffer and the relationship to apply to these fields.

The field number indicates a field location in the intermediate output mapping buffer as defined by the intermediate mapping definition in the cursor for the previous data space. This location contains the join value(s) to be used. The definition of this field(s) in the intermediate buffer must be identical to the definition of the key field(s) comprising the composite key for this DS in the index for this secondary data space.

Packed and zoned numeric key fields used in the join must have been specified with numeric ordering options other than internal form. Internal ordering allows key fields in the index to contain non-preferred S/38 signs. Floating point key fields require that the cursor's floating point computational attributes are identical to the index's floating point computational attributes. A template value invalid exception will be signaled on the join field identities template field corresponding to the key field in error for these conditions.

The composite key is constructed using the join field(s), inserting fork characters and ordered as defined by the key field attributes for the data space referenced by the data space number in the index data space list. The number of field identities (join fields) may be greater than the number of key fields comprising the composite key, however, the excess join fields are ignored when constructing the key, likewise, fewer join fields may be specified.

The join relationship indicates the relationship to be applied when comparing the join value in the current data space to the join value from the field in the intermediate buffer associated with the previous data space.

The format of the group-by processing template follows:

- Group-by processing definition template

– Number of function specifiers	Bin(2)
– Function specifier definition (repeated for each definition)	Char(10)
Field number	Bin(2)
Result field description	
Field type	Bin(2)
Field length	Bin(2)
Rounding mode	Char(1)
Hex 02 = Round to zero (truncate)	
Hex 04 = Decimal round	
Unassigned values reserved	
Hex 05 = Not applicable	
Operation	Char(1)
Hex 01 = Count	
Hex 02 = Materialize key field	
Hex 03 = Sum	
Hex 04 = Minimum	
Hex 05 = Maximum	
Remaining values reserved	
Reserved (binary 0)	Char(2)

The group-by processing definition template describes the operation to be performed on specified field(s).

The number of function specifiers field specifies the number of elements in the function specifier definition table.

The field number specifies the relative position of the field within a source buffer as defined by the mapping in the cursor. A template value invalid exception is signaled if the field number specified has no corresponding field defined in the mapping template in the cursor. The source for the field will be either the data-space entry if output intermediate mapping is not specified or the output intermediate buffer.

The allowable field types and field lengths for the group-by result field description are the same as those allowable for input and output mapping described earlier

The field number is ignored when the count and materialize key field operation is specified. The count is incremented by 1 for each entry participating in the group. The field type for sum operation disallows a field type of character. In addition, the field type for the count must be a binary(4) or float(8) type.

The materialize key field group-by operation indicates a key field of the composite group-by key is to be materialized in the result field. The order of the function specifiers with materialize key field operations defines the key field to be materialized. The first function specifier with materialize key field specified will materialize the second key field, and so on. Key fields defined to be materialized but not participating in the group designated by the key field count or key byte count in Retrieve Sequential Data Space Entries, will contain default value of hex 40 for character and zeros (in the appropriate representation) for numerics). The field number and result field description are ignored for materialized key fields. The result field type and length will be identical with the key field description of the key field as materialized by Set Cursor, Retrieve Sequential, and Materialize Cursor Attributes instructions. Specifying the materialize key field is not allowed if the cursor is not over an index.

The group-by result fields define the first fields of the group-by intermediate buffer. The remaining fields are defined by the optional group-by intermediate mapping template and are assigned the field numbers immediately following the number assigned to the group-by result fields.

Minimum and maximum are restricted to length of 256 bytes when result field type is character. Result field type and length for minimum and maximum must be identical with the type and length of the field referenced by the field number.

The format of the selection template for output and group-by selection follows:

- Length of selection template Bin(4)
- Number of selection descriptors Bin(2)
- Reserved (binary 0) Char(8)
- Number of data pointers Bin(2)
- Data pointer (repeated Data pointer
- Selection descriptor (repeated) Char(16)
  - Descriptor type Char(1)

    Hex 00 = Operand is a field.
    Hex 01 = Operand is literal.
    Hex 03 = Operand is a pattern.
    Hex 04 = Operator.
    Unassigned values are reserved.
  - Operation/operand location Char(7)

    Operation descriptor
    Reserved (binary 0) Char(5)
    Operation Char(2)

    or

    Operand is field descriptor
    Operation location Char(1)
    Field source
    Hex 00 = Data space
    Hex 01 = Cursor intermediate buffer
    Hex 02 = Group-by intermediate buffer
    Hex 03 = Reserved
    Hex 04 = Key field
    Unassigned values reserved
    Field number Bin(2)
    Starting character Bin(2)
    Ending character Bin(2)

    or

    Operand is literal descriptor
    Operand location Char(1)
    Field source
    Hex 00 = Data space
    Hex 01 = Cursor intermediate buffer
    Hex 02 = Group-by intermediate buffer
    Hex 03 = Reserved
    Hex 04 = Key field
    Unassigned values reserved
    Field number Bin(2)
    Starting character Bin(2)
    Ending character Bin(2)

    or

    Operand is literal descriptor
    Reserved (binary 0) Char(1)
    (bit 0,1 used internally)
    Reserved (binary 0) Char(4)
    Array position of data pointer Bin(2)

    or

    Operand is a pattern
    Reserved (binary 0) (Char 1)
    (bit 0 used internally)
    Offset from start of this template to pattern descriptor array Bin(4)
    Number of pattern descriptors in array Bin(2)

    or

    All other descriptor types
    Reserved (binary 0) Char(7)
  - Reserved (binary 0) Char(8)
- Pattern descriptor arrays Char(*)

The selection template must be aligned on a 16-byte boundary.

The allowable operation codes and definitions follow:

Operation Codes

Hex 0001 = Equal
Hex 0002 = Not equal
Hex 0003 = Greater than or equal
Hex 0004 = Less than or equal
Hex 0005 = Greater than
Hex 0006 = Less than
Hex 0007 = Range (low bound <= field; >= high bound)
Hex 0008 = Range (low bound <+ field; > high bound)
Hex 0009 = Range (low bound < field; <= high bound)
Hex 000A = Range (low bound < field; < high bound)
Hex 000B = OR
Hex 000C = XOR
Hex 000D = AND
Hex 000E = NOT
Hex 000F = Select and exit if true
Hex 0010 = Omit and exit if false
Unassigned values reserved

The selection template(s) will be used to perform selection against the specified data in the location given.

The template will be used to produce a run-time Boolean result stack.

The Boolean result stack houses the intermediate results. All values stored on this stack are Boolean in nature.

The resulting selection or omission is designated by the value residing on the top of this run-time stack. The select and exit if true operation causes the value on the top of this stack to remain unchanged. (If it is already true, the corresponding DSE is selected.) Similarly the omit and exit if false operation leaves the Boolean result residing on the top of this run-time stack unaltered.

The final result residing on the top of this Boolean stack thereby reveals whether the DSE has been selected (true resides on the top of the stack) or has been omitted (false resides on the top of the stack).

The operand is field descriptor specifies this descriptor designates on operand with the operand location specified by field source and field number. The field number describes the relative location of the associated field in the field source. The field source may be either a data space entry field, intermediate buffer field, or a key field. The field type of the field referenced by field number may not be type of dummy.

The key field refers to the field number in the composite key of the data space index over the data space as referenced by the cursor. The key field number does not include fork character fields. The selection is performed on the key field contents prior to appending of fork characters or manipulating the fields into internal physical key representation as specified by the key field specification.

The following are the allowable operand locations when specifying different selection templates:

Selection	**Allowable Operand Location**
Output	Data space entry Cursor intermediate output buffer Key field
Group-by	Group-by intermediate buffer

The starting character and ending character value of binary zero indicates the entire field should be used. A non-zero value is valid only for fields with type of character.

The operand is literal descriptor specifies this descriptor designates an operand with the operand identified as a literal. The literal type and location is given by a data pointer in the array of data pointers. The specific data pointer location in the array is given by the array position of data pointer field for this descriptor.

The operand is a pattern specifies this descriptor designates an operand with the operand identified as an array of pattern descriptors. The pattern scan is described by the array of pattern descriptors. The array of pattern descriptors is located by an offset from the start of the selection template. The specific offset is given by the offset from start of this template to pattern descriptor array field for this descriptor.

The operator specifies this descriptor designates an operator, with the operation to perform given by the operation field for this descriptor. The allowable operations are listed under operation codes.

The low bound and high bound of the range operators must be literals.

The select and exit if true and omit and exit if false descriptor types indicate specific action to take when these results occur. The absence of these descriptor types will leave the result of the current operation at the top of the result stack; the last operation would control the selection outcome for the entrie selection process.

The format of a pattern descriptor follows:

• Pattern descriptor	Char(16)
– Descriptor type	Char(1)
Hex 00 = Field	
Hex 01 = Literal	
Hex 02 = Span	
Unassigned values reserved	
– Descriptor	Char(7)
Wild card span	
Span type	Char(1)
Hex 01 = Fixed	
Hex 02 = Float	
Unassigned values reserved	
Span width (binary 0 for float span)	Bin(2)
Reserved (binary 0)	Char(4)
or	
Field descriptor	
Location	Char(1)
Field source	
Hex 01 = Data space	
Hex 01 = Cursor intermediate buffer	
Hex 02 = Group-by intermediate buffer	
Hex 03 = Reserved	
Hex 04 = Key field	
Unassigned values reserved	
Field number	Bin(2)
Starting offset	Bin(2)
Ending offset	Bin(2)
or	
Literal Descriptor	
Reserved (binary 0) (bit 0,1 used internally)	Char(1)
Reserved (binary 0)	Char(4)
Array position of data pointer	Bin(2)
– Reserved (binary 0)	Char(8)

The field descriptor and literal descriptor are identical with that described in the selection descriptor.

The field or literal of the pattern must be character and likewise the field of the target must be character. The wild card span pattern descriptor type specifies the width of bytes to ignore before the following pattern scan begins. A fixed span specifies the number of bytes designated by span width is to be skipped after the current position, before the following field or literal scan begins. A float span specifies that the following pattern scan may begin in any byte following the current position inclusive of the current byte. The current position is initially established by the associated selection description for the target of the scan. As field or literal pattern matches are satisfied, this position is then changed to the byte in the target immediately following the byte(s) matching the criteria.

The pattern descriptor array may begin with any of the descriptor types. Any combination of descriptor types may be specified.

The pattern scan is performed from left to right of the specified location.

The order of operands, operator, and exit criterion descriptors to specify for each selection criterion follows:

For pattern match operations:

1. Operand field descriptor or literal

2. Operand pattern match

For dual bounded range operations:

1. Operand: Low bound

2. Operand: Field description or literal

3. Operand: High bound

4. Operation

For all other operations:

1. Operand field description or literal

2. Operand field description or literal

3. Operation

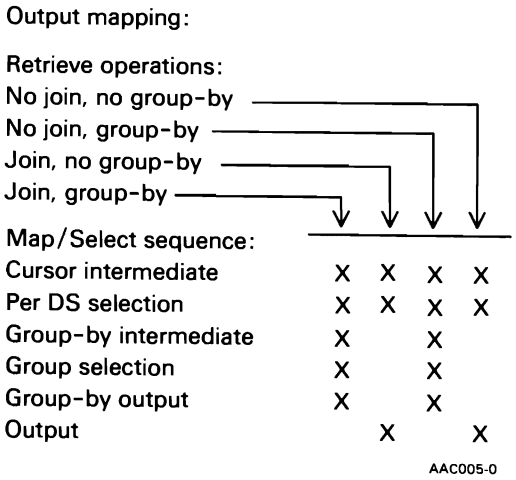
The OR, XOR, and AND operations may follow any two consecutive sets of the preceding operation. The operation is applied to the last two results obtained. The NOT operation may follow any operation set and is applied to the result of that operation.

Output selection may be specified for a subset of the data spaces in the DS list.

When output selection is specified on a join cursor, the selection criteria associated with a specific data space may specify any field number in the intermediate mapping table associated with that same data space or any data space preceding that data space in the data space list.

Fields and literals of an operation must be of identical type and length.

The mapping sequences with selection follow:



Output mapping:

Retrieve operations:

Map/Select sequence:	Join, group-by	Join, no group-by	No join, group-by	No join, no group-by
Cursor intermediate	X	X	X	X
Per DS selection	X	X	X	X
Group-by intermediate	X		X	
Group selection	X		X	
Group-by output	X		X	
Output		X		X

The number of selection descriptors must not be greater than 24,563.

The number of pattern descriptors must no exceed 10,917.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
- Insert
  - User profile of creating process
  - Context identified in operand 2
- Object management
  - Data spaces identified in operand 2
  - Data space index identified in operand 2

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Access group identified in operand 2
  - Context identified in operand 2
  - User profile of creating process

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	Operands 2	Other
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
12 Data Base Management			
08 Incomplete key description		X	
13 Invalid mapping template		X	
14 Invalid selection template		X	
15 Data space not addressed by index		X	
1B Logical data space entry size limit exceeded		X	
1D Logical key size limit exceeded		X	
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
38 Template Specification			
01 Template value invalid		X	

CREATE DATA SPACE (CRTDS)

Op Code (Hex)	Operand 1	Operand 2
045A	Data space	Data space template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction creates a data space according to the data space template specified by operand 2. The template describes the type of data space to be created, the characteristics of that data space, and the attributes of the fields that make up the individual entries within the data space. Addressability to the newly created data space is returned in the system pointer stored in the addressing object specified by operand 1.

The format of the data space template is as follows:

• Template size specification	Char(8)
– Number of bytes provided by the user	Bin(4)*
– Number of bytes that can be materialized	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes	Bit 0
0 = Not permanent	
1 = Permanent	
– Space attribute	Bit 1
0 = Fixed-length	
1 = Variable-length	
– Initial context	Bit 2
0 = Addressability is not inserted in context.	
1 = Addressability is inserted in context.	
– Reserved (binary 0)	Bits 3-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)

- Initial value of space — Char(1)

- Performance class — Char(4)
  - Space alignment — Bit 0
    - 0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    - 1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.
  - Reserved (binary 0) — Bits 1-4
  - Main storage pool selection — Bit 5
    - 0 = Process default main storage pool is used for object.
    - 1 = Machine default main storage pool is used for object.
  - Reserved (binary 0) — Bit 6
  - Block transfer on implicit access state modification — Bit 7
    - 0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
    - 1 = Transfer the machine default storage transfer size. This value is 8 storage units.
  - Reserved (binary 0) — Bits 8-31

- Reserved (binary 0) — Char(7)

- Context — System pointer

- Reserved (binary 0) — Char(16)

- Data space attributes — Char(2)
  - Relative entry — Bit 0
  - Reserved (binary 0) — Bits 1-2
  - Initial allocation — Bit 3
    - 0 = Use default allocation
    - 1 = Allocate for maximum number of entries
  - Contiguous return — Bit 4*
    - 0 = Contiguous storage not allocated
    - 1 = Contiguous storage allocated
  - Unit return — Bit 5*
    - 0 = Not allocated on requested unit
    - 1 = Allocated on requested unit
  - Conversion error checking — Bit 6
    - 0 = Conversion error checking not enabled
    - 1 = Conversion error checking enabled
  - Contiguous allocation — Bit 7
  - Floating-point attributes specified — Bit 8
    - 0 = Use default floating-point attributes
    - 1 = Use specified floating-point attributes
  - Reserved (binary 0) — Bits 9-15

- Maximum number of entries — Bin(4)

- Entry number increment — Bin(2)

- Unit identification — Char(1)

- Compression threshold — Char(1)

- Length of the entry definition table — Bin(2)

- Offset to the entry definition table — Bin(4)

- Length of the default values entry — Bin(2)

- Offset to the default values entry — Bin(4)

• Floating-point attributes	Char(1)
– Floating-point overflow mask 0 = Masked 1 = Unmasked	Bit 0
– Floating-point underflow mask 0 = Masked 1 = Unmasked	Bit 1
– Floating-point zero divide mask 0 = Masked 1 = Unmasked	Bit 2
– Floating-point inexact result mask 0 = Masked 1 = Unmasked	Bit 3
– Floating-point invalid operand mask 0 = Masked 1 = Unmasked	Bit 4
– Reserved (binary 0)	Bit 5
– Floating-point rounding mode 00 = Round to positive infinity 01 = Round to negative infinity 10 = Round to zero (truncate) 11 = Round to nearest or even	Bits 6-7
• Reserved (binary 0)	Char(5)

Note: The value of an entry shown here with an asterisk (*) is ignored by this instruction.

The data space template must be aligned on a multiple of 16 bytes.

The object identification specifies the symbolic name that identifies the data space within the machine. A type code of hex 0B is implicitly supplied by the machine. The object identification is used to identify the data space on materialize instructions as well as to locate the object in a context that addresses the object.

Data spaces are created as permanent objects and exist in the machine until explicitly destroyed by the user. A space may be associated with the created data space. The space may be fixed or variable in size. The initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified; the actual size allocated depends on an algorithm defined by a specific implementation. A fixed size space of zero length causes no space to be allocated. If no space is allocated, this value is ignored.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended, this byte value is also used to initialize the new allocation. If no space is allocated, this value is ignored.

The user profile governing process execution is assigned ownership of the object, and the storage occupied by the data space is charged to this user profile.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created data space is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

The performance class parameter provides information that allows the machine to manage the data space with consideration for the overall performance objectives of operations involving the data space.

The data space attributes entry specifies the type of data space being created and its allocation requirements. The type of the data space can be specified to be relative.

If the initial allocation attribute is specified (binary 1), sufficient storage is allocated to contain the number of data space entries specified by the maximum number of entries field. Data spaces are implicitly extended until logically the number of entries exceeds the maximum number of entries specified by the entry number increment parameter.

If initial allocation is not specified (binary 0), a default initial allocation and extension allocation are used. If initial allocation is specified (binary one) and the maximum number of entries field contains a value of zero, a template value invalid exception is signaled.

The values of the contiguous return bit and unit return bit are set by this instruction. The contiguous return bit (binary 1) indicates the data portion of the data space is contiguously allocated on auxiliary storage. The contiguous return bit (binary 0) indicates either that the data portion of the data space is not contiguously allocated on auxiliary storage or that contiguous storage was not requested. No exception is signaled as the result of failing to obtain a contiguous allocation when requested.

A value of binary one in the floating-point attributes specified field indicates that the floating-point attributes field contains the floating-point attributes for this data space. If this field contains a binary zero, the default floating-point attributes are used. The default floating-point attributes are as follows:

- The floating-point overflow mask field contains a binary one indicating that this exception is unmasked.
- The floating-point underflow mask field contains a binary one indicating that this exception is unmasked.
- The floating-point zero divide mask field contains a binary one indicating that this exception is unmasked.
- The floating-point inexact result mask field contains a binary zero indicating that this exception is masked.
- The floating-point invalid operand mask field contains a binary one indicating that this exception is unmasked.
- The floating-point rounding mode field contains a binary 11 indicating that round to nearest or even is enabled.

The conversion mapping error exception will not be signaled if the enable conversion error checking field has a value of binary 0, and if a data conversion or truncation error is encountered on a numeric field while mapping to or from the interface buffer on RETDSEN, RETSDSE, UPDSEN, INSDSEN, or INSSDSE instructions. The erroneous data will be used in generating the interface buffer or the data space entry. If the enable conversion error checking field has a value of binary 1, the conversion mapping error exception will be signaled for each entry that produces a conversion or truncation error. The indicated instructions will not detect conversion or truncation errors if the fields in the data space entry are not converted or truncated, as in direct mapping. The key conversion mapping error is always signaled when encountered, regardless of the value of the enable conversion error checking field.

If the contiguous allocation bit is binary 0, the system attempts to allocate the data space contiguously on auxiliary storage. If this bit is binary 1, the data space may not be contiguously allocated on auxiliary storage. If the initial allocation field is binary 0, the contiguous allocation bit is ignored.

The maximum number of entries field specifies the number of entries that can reside in a data space before the data space entry limit exceeded exception is signaled. If this field is 0, an implementation-defined maximum is assumed. The entry number increment field specifies an increment that can be applied to the maximum number of entries field through the use of the Data Base Maintenance instruction to derive a new upper limit. If the maximum number of entries field specifies a negative number or a number greater than the number of entries allowed in a data space, a template value invalid exception is signaled.

The unit identification entry (which is interpreted as a 1-byte unsigned binary number) indicates the auxiliary storage unit on which the data space should reside. Unit values are installation dependent. If it is not possible to allocate all of the data space on the requested unit, the instruction sets the unit return bit to binary 0; otherwise, the unit return bit is set to binary 1 indicating the data portion of the data space resides on the requested auxiliary storage unit. If no specific unit is selected (binary 0), the machine selects the unit for data space storage and returns a value of binary 0 in the unit return value. If the unit identification is nonzero, it must be valid for the machine. The Materialize Resource Management Data instruction or Destroy Cursor instruction provides the allowable valid unit numbers. If the intended unit has insufficient space to accommodate the data space, an alternative unit is selected.

The compression threshold entry is interpreted as a 1-byte unsigned binary number that specifies the percentage of deleted entries that can remain in the data space before the data space compression threshold exceeded event is signaled. The event is signaled on any De-Activate Cursor instruction where the compression threshold of a data space referenced by that cursor has been exceeded. The compression threshold represents a percentage expressed as a number between 0 and 100 (inclusive). If the percentage equals 0, the event is not signaled.

The entry definition table defines the format of the data space entries for this data space. The offset to the entry definition table defines the offset from the start of the data space template to the first byte of the entry definition table. The length of the entry definition table identifies the number of bytes in the table.

The default values entry is a character string equal in length to the computed length of the data space entry. This string defines the default values for insert operations to use for any field that is not present in the input mapping template of a Create Cursor instruction. This string also defines the default values for an Update Data Space Entry instruction to use when deleted entries are updated as well as the values to be inserted by the insert default entries option of the Data Base Maintenance instruction. The offset to the default values entry defines the offset from the start of the data space template to the first byte of the default values entry. The length of the default values entry identifies the number of bytes in the default values entry.

No data validity checking is done on the contents of the default values entry field. If the offset to the default values entry is 0, no default entry is provided, and the length field is ignored. If default values are not provided, the default values supplied by the machine are blanks (hex 40) for character fields and 0's (in the appropriate representation) for numeric fields.

The floating-point attributes are the floating-point computational attributes, that are placed in effect whenever the to data space entry or the to key mapping is executed. The floating-point attributes are as follows:

- A value of binary one for the floating-point overflow mask field indicates that the floating-point overflow exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point underflow mask field indicates that the floating-point underflow exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point zero divide mask field indicates that the floating-point zero divide exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point inexact result mask field indicates that the floating-point inexact result exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point invalid operand mask field indicates that the floating-point invalid operand exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- The floating-point rounding mode controls how floating-point values are rounded. The Store and Set Computational Attributes instruction provides a definition of the floating-point rounding modes.

The entry definition table defines the field attributes, one for each field in the data space entry. The number of fields in the data space entry (number of entries in the table) is the value of the length of the entry definition table divided by 4 bytes per field attribute.

Each field attributes entry designates the attributes that field processes in the data space entry.

The format of the field attributes is as follows:

• Field attributes	Char(4)
– Field type	Bin(2)
– Field length	Bin(2)

The following field types and specification codes are allowed:

Field Type	Specification Code (Hex)
Binary	0000
Floating-point	0001
Zoned decimal	0002
Packed decimal	0003
Character	0004

The permissible values for each of the field lengths are as follows:

Field Type	Allowed Field Length Values	
Binary	Bytes 1-2	– Length in bytes = Binary 2 or 4
Floating-point	Bytes 1-2	– Length in bytes = Binary 4 or 8
Zoned decimal	Byte 1	– Fractional digits[1] = Binary 0 to total number of digits
	Byte 2	– Total number of digits = Binary 1 to 31
Packed decimal	Byte 1	– Fractional digits = Binary 0 to total number of digits
	Byte 2	– Total number of digits = Binary 1 to 31
Character	Bytes 1-2	– Length in bytes = Binary 1 to 32 766

[1] The number of fractional digits to the right of the decimal point.

Authorization Required

- Insert
  - Context identified in operand 2
  - User profile of creating process

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Context identified in operand 2
  - User profile of creating process

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded
  0501 Machine address threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context Operation			
01 Duplicate object identification	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
12 Data Base Management			
1A Data entry size exceeded		X	
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
38 Template Specification			
01 Template value invalid		X	

CREATE DATA SPACE INDEX (CRTDSINX)

Op Code (Hex)	Operand 1	Operand 2
046A	Data space index	Data space index template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction creates a data space index that defines an alternate ordering over the entries in one or more data spaces. The data space index orders keys derived from the data space entries according to a standard collating sequence or a user-provided alternate collating sequence and can include all, or a subset, of the entries in the associated data space(s). Addressability to the newly created data space index is returned in the system pointer specified by operand 1.

The format of the data space index template is as follows:

• Template size	Char(8)
– Number of bytes provided by the user	Bin(4)*
– Number of bytes that can be materialized	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temporary 1 = Permanent (required)	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context 0 = Addressability is not inserted in context. 1 = Addressability is inserted in context.	Bit 2
– Reserved (binary 0)	Bits 3-31
• Recovery options	Char(4)
• Size of space	Bin(4)

• Initial value of space	Char(1)
• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Access group	System pointer
• Data space list pointer	Space pointer
• Alternate collating template pointer	Space pointer

• Selection template pointer	Space pointer
• Length of selection template	Bin(2)
• Length of data space list	Bin(2)
• Index attributes	Char(2)
– Floating-point attributes specified	Bit 0
0 = Use default floating-point attributes	
1 = Use specified floating-point attributes	
– Extended creation template	Bit 1
– Reserved (binary 0)	Bits 2-7
– Alternate collating template	Bit 8
0 = Template not provided	
1 = Template provided	
– Validate index option	Bit 9
0 = Create valid	
1 = Created invalidated index	
– Unit return bit	Bit 10*
0 = Not allocated on requested unit	
1 = Allocated on requested unit	
– Delayed maintenance	Bit 11
– Optimized processing mode	Bit 12
0 = Random	
1 = Sequential	
– Data space index force option	Bit 13
0 = Normal data space index force	
1 = Force all data space index changes	
– Duplicate key rules	Bits 14-15
00 = Unique keys required	
01 = LIFO duplicates permitted	
10 = FIFO duplicates permitted	
11 = Reserved	

• Unit identification	Char(1)
• Floating-point attributes	Char(1)
– Floating-point overflow mask	Bit 0
0 = Masked	
1 = Unmasked	
– Floating-point underflow mask	Bit 1
0 = Masked	
1 = Unmasked	
– Floating-point zero divide mask	Bit 2
0 = Masked	
1 = Unmasked	
– Floating-point inexact result mask	Bit 3
0 = Masked	
1 = Unmasked	
– Floating-point invalid operand mask	Bit 4
0 = Masked	
1 = Unmasked	
– Reserved (binary 0)	Bit 5
– Floating-point rounding mode	Bits 6-7
00 = Round to positive infinity	
01 = Round to negative infinity	
10 = Round to zero (truncate)	
11 = Round to nearest or even	
• Length of the data space key specifications	Bin(4)

The data space key specification is repeated for each data space.

• Key field count	Bin(2)
• Key field specification (repeated for each field in the data space key)	Char(4)
– Key field number	Bin(2)
– Key field attributes	Char(2)
Reserved (binary 0)	Bits 0-7
Ordering option	Bit 8
0 = Ascending sequence	
1 = Descending sequence	
Numeric ordering	Bits 9-10
00 = Internal form	
01 = Absolute value	
10 = Algebraic	
11 = Reserved	
Fork character	Bit 11
0 = No fork character specified	
1 = Fork character specified	
Alternate collating	Bit 12
0 = Machine default collating sequence	
1 = Alternate collating sequence	
Zone/digit force	Bits 13-14
00 = No zone/digit force	
01 = Digit force	
10 = Zone force	
11 = Reserved	
Reserved (binary 0)	Bit 15
• Extended area of creation template	
– Extended functions	Char(2)
Create index from index	Bit(0)
Non-user exit selection	Bit(1)
Reserved (binary 0)	Bit(2-15)
– Reserved (binary 0)	Char(32)
– Offset to intermediate key mapping table pointer list	Bin(2)
– Offset to source data space index list pointer	Bin(2)
– Offset to non-user exit selection template pointer list	Bin(2)
– Number of translate tables	Bin(2)
– Offset to translate table list	Bin(2)
– Index logical page size	Bin(2)
– Reserved (binary 0)	Char(6)

Notes:

1. The data space index template, data space list, and selection template must each be 16-byte aligned.
2. The values of the entries shown here with an asterisk (*) are ignored by this instruction.

The data space index is owned by the user profile that governs process execution. The owning user profile is implicitly assigned all authority states for the data space index. The storage occupied by the data space index is charged to this owning user profile.

The object identification specifies the symbolic name that identifies the data space index within the machine. A type code of hex 0C is implicitly supplied by the machine. The object identification is used to identify the data space index on materialize instructions as well as to locate the data space index in a context that addresses the data space index.

The existence attribute specifies whether the data space index is to be created as temporary or permanent. A temporary data space index, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated. A permanent data space index exists in the machine until explicitly destroyed by the user.

A space may be associated with the data space index. The space may be fixed or variable in size. The initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified; the actual size allocated depends on an algorithm defined by a specific implementation. A fixed size space of zero length causes no space to be allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended, this byte value is also used to initialize the new allocation. If no space is allocated, this value is ignored.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created data space index is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

The performance class parameter provides information that allows the machine to manage the data space index with consideration for the overall performance objectives of operations involving the data space index.

When the index is created from data space(s), the data space list pointer identifies a list of system pointers. Each system pointer addresses a data space. The length of data space list entry (which must be a multiple of 16 bytes) indicates the number of bytes in the list. Only these data spaces are addressable through the data space index.

When the create index from index field contains a binary one, the data space list pointer addresses a table of 2-byte elements, each element assumed to be a Bin(2) field. Each element's value specifies a slot in the source index's data space list. This value must specify an existing slot. The data space address given by the source index in the associated data space list slot will be utilized as a data space the new index will cover. The order of the data spaces in the newly created index is determined by the order of the elements in the table.

The ordering of the data space pointers in the data space list is used, in exception data, to identify the data spaces with a 2-byte number known as the data space number. The first data space in the list is assigned the number one, and the nth data space in the list is assigned the number n.

The ordering of the data spaces is significant in data space indexes where duplicate keys are allowed because duplicate keys from different data spaces appear in the index in the same order as the data spaces appear in the list.

Index keys are normally ordered by the machine's standard collating sequence. The alternate collating template pointer (if provided) points to a fixed-length, 256-byte alternate collating template. If the pointer is not provided and any data space key specification specifies alternate collating, an exception is signaled.

Data space entry selection allows the data space index to address a selected subset of data space entries covered by the data space index, rather than address all data space entries. The number of bytes in the selection template is indicated by the length of selection template. A binary 0 in the length of selection template field indicates that selection is not used for this data space index and the selection template pointer is ignored.

The index attribute entry specifies general data space index attributes. An alternate collating template attribute value of binary 1 indicates that the alternate collating template pointer addresses a 256-byte alternate collating template. A binary 0 indicates that the pointer is to be ignored.

The create invalidated index attribute indicates that a data space index addressing no entries and marked invalid should be created. This attribute has the same effect as if the index had been operated on by the invalidate data space index option of the Data Base Maintenance instruction.

A value of 0 causes a valid, up-to-date index to be created.

A value of binary one in the floating-point attributes specified field indicates that the floating-point attributes field contains the floating-point attributes for this index. If this field contains a binary zero, the default floating-point attributes are used. The default floating-point attributes are as follows:

- The floating-point overflow mask field contains a binary one indicating that this exception is unmasked.
- The floating-point underflow mask field contains a binary one indicating that this exception is unmasked.
- The floating-point zero divide mask field contains a binary one indicating that this exception is unmasked.
- The floating-point inexact result mask field contains a binary zero indicating that this exception is masked.
- The floating-point invalid operand mask field contains a binary one indicating that this exception is unmasked.
- The floating-point rounding mode field contains a binary 11 indicating that round to nearest or even is enabled.

The unit return bit is set by this instruction. A value of binary 1 indicates that the index is on the requested auxiliary storage unit. A value of binary 0 indicates that some of the index is not on the requested unit. If no unit identification is specified (binary 0), the unit return bit is 0.

The delayed maintenance option, equal to binary 1, delays changes to the data space until a cursor that references the data space is activated. This delay is used for performance reasons. Changes to the data space index occur when an Activate Cursor instruction is issued to a cursor that references the data space index or when a Data Base Maintenance instruction is used to explicitly rebuild the index after an up-to-date index was created. A value of 0 indicates that immediate index maintenance is to be used. If duplicate key rules are equal to the unique keys, the delayed maintenance value must be 0.

If the optimized processing mode field is binary 1, then the data space index will be built and maintained in a way that attempts to optimize performance for sequential operations on the data space index. Otherwise, the optimization will be done for random access operations.

If the data space index force option attribute field contains a value of binary 1, the system will ensure that any change to data space entries that result in changes to this data space index will result in the changed portion of the data space index being written to nonvolatile storage along with the changed data space entries before the instruction is completed. A value of binary zero in this field indicates that this data space index is to be forced to nonvolatile storage when the operation being performed against it or the underlying data spaces or data space entries indicates that this force is to be performed. This option may not be specified when creating a temporary index.

The duplicate key rules have the following meaning:

- If unique keys are specified, then duplicate keys are not allowed in the index. During an index creation or rebuild, the operation is terminated if duplicate keys are detected. During insertion or modification of a data space entry, detection of a duplicate key will inhibit alteration of the data space. If the index has been implicitly invalidated by the machine, changes to the data space entries that could result in duplicate keys are not allowed. In either case, an exception is signaled.

- If duplicate keys are permitted, then the LIFO (last in, first out), or the FIFO (first in, first out) rule determines how duplicate keys are to be ordered within the data space index.

The LIFO or FIFO rules only apply to the ordering of duplicate keys acquired from entries that reside in the same data space. If LIFO is specified, then the entry with the largest ordinal number is ordered first. If FIFO is specified, then the entry with the smallest ordinal number is ordered first. When duplicate keys are acquired from entries that reside in different data spaces, the ordering is determined by the order of the data spaces as they are specified in the data space pointer list.

The unit identification entry is interpreted as a 1-byte unsigned binary number indicating a valid auxiliary storage unit on which the data space index should reside. If no unit identification is specified (binary 0), the machine selects an auxiliary storage unit for the data space index. The value of the unit identification is installation dependent.

Valid unit numbers can be obtained by using the Materialize Resource Management Data instruction. If the intended unit has insufficient space to accommodate the data space index, an alternative unit is selected.

The floating-point attributes are the floating-point computational attributes, that are placed in effect for the execution of the select/omit mapping and the select/omit program. The floating-point attributes are as follows:

- A value of binary one for the floating-point overflow mask field indicates that the floating-point overflow exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point underflow mask field indicates that the floating-point underflow exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point zero divide mask field indicates that the floating-point zero divide exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point inexact result mask field indicates that the floating-point inexact result exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- A value of binary one for the floating-point invalid operand mask field indicates that the floating-point invalid operand exception is unmasked and can be signaled. A value of binary zero indicates that the exception is masked and cannot be signaled.

- The floating-point rounding mode controls how floating-point values are rounded. The Store and Set Computational Attributes instruction provides a definition of the floating-point rounding modes.

Each data space key specification entry defines a key for a data space. A data space key specification must be defined for each data space referenced by the data space list, and its order must correspond to the order of the data spaces in the list. If more than one key specification is defined for a data space, then the data space must appear in the data space list more than once, and each entry in the data space provides more than one key to the index.

When intermediate key mapping is specified, the key field(s) defined in the key field specifications refer to the field(s) defined in the intermediate key mapping buffer, not those defined in the data space entry.

The key field count entry specifies the number of key field specification entries for a particular data space. A key field specification entry appears for each field extracted from the data space entry as well as each fork character to be used in creating the key for a particular data space. The key field location entry identifies the relative position of the field in the data space entry. The first field in the entry is relative position 1.

The key field number entry identifies the relative position of the field in the data space entry or intermediate key buffer. The first field in the source location is relative position 1.

The key field attributes entry specifies the attributes of the corresponding key field.

The ordering option attribute specifies whether the key field is collated in ascending or descending sequence. Descending sequence is valid with any field attribute except fork character.

The numeric ordering attribute specifies whether numeric fields are to be ordered based on their internal representation value, algebraic value, or absolute numeric value. The numeric ordering attributes of algebraic or absolute value causes the specified numeric ordering to be enforced independent of a field's numeric type or internal physical representation. If the field is a floating-point field and algebraic collating is specified, then the collating sequence is:

> negative infinity → real numbers → positive infinity.

If the field is a floating-point field and absolute collating is specified then the collating sequence is:

> real numbers → infinities.

If internal form numeric ordering is specified, ordering is performed according to the physical storage representation of the key field. For example, a packed decimal number has its sign on the right. This causes the ordering to alternate between positive and negative numbers. For zoned decimal, the sign is in the left half of the rightmost byte, which causes the ordering to be 10 positive numbers followed by 10 negative numbers. For floating-point, the sign is the leftmost bit, the exponent is next, and the significant is last; so the collating sequence is:

> positive real numbers → positive infinity → negative real numbers → negative infinity →.

Numeric ordering can be used with any data type except character. Numeric ordering is valid with the ascending and descending field attributes only. Any other attribute specified with numeric ordering results in an exception.

The fork character attribute indicates that a data space entry field or intermediate key buffer field is not being specified and that the key field location entry contains a fork character (rather than the identity of a field within the data space entry field) to be inserted into the composite key at this position. Byte 1 of the key field location is ignored, and byte 2 must contain the fork character to be inserted into the composite key. It is important to note that the data space index functions append information to the rightmost portion of each key, and, therefore, it may be necessary to place a fork character at the end of each short key to ensure that the appended information does not affect the ordering of this key with respect to longer keys. If the fork character option is specified, all other key field attributes must be binary 0 or an exception is signaled.

The alternate collating attribute indicates that the value acquired from the data space entry is to be modified in accordance with the alternate collating template before being placed into the key. This modification is performed after the zone or digit force changes have been applied but before the descending sequence changes, if either is specified. This attribute is valid for character and zoned decimal fields only; it is also valid with the descending sequence and either zone/digit force key field attributes. Any other data type or key field attributes result in an exception.

The zone/digit force attribute specifies a modification to 4 bits of every byte in the specified key field. Zone force (10) causes the leftmost 4 bits (the zone portion) of every byte in the field to be set to zeros. Digit force (01) causes the rightmost 4 bits (the digit portion) of every byte in the field to be set to 0's. These attributes are valid for the character and zoned data fields only; they are also valid with the descending sequence and the alternate collating key field attributes. Any other data type or key field attributes result in an exception.

The order in which the key field specifications appear in the template determines the order of the fields in the resulting key. The data space key field count must include both the data key fields extracted from the data space entry as well as the fork characters that comprise the resulting key.

The alternate collating template, if one exists, is used as a translation table needed for a specific alternate collating sequence. This translation table must consist of a 256-byte table of replacement values. The replacement value for a specific byte is located in the table at an offset equal to the byte's binary value. For example, if hex C1 is to be replaced with hex F2, the byte residing at offset hex C1 in the table must contain the replacement value hex F2. When alternate collating sequence is specified for a field, the field is translated before being placed into the key. For the example above, this means that when the keys are automatically ordered, hex C1 = A is logically placed in the index between hex F1 = 1 and hex F3 = 3. Thus, an alternate collating sequence of 1A3 is achieved.

A value of binary one in the create index from index field specifies this index is to be created via the contents of an existing data space index. The newly created index will be constructed using the data space entries addressed by the source data space index. The existing data space index is addressed by the source data space index system pointer. The offset to source data space index list pointer defines the offset from the beginning of the data space index creation template to the first byte of a list of one system pointer to the source data space index. If the source data space index is invalid, the new index will be built by referencing the data space entries directly.

If the source data space index becomes invalid after the build of the new index begins, an exception data space index invalid is signaled and the creation fails.

Any existing selection template(s) from the source data space index will be adopted by the newly created data space index.

The offset to intermediate key mapping templates pointer list defines the offset from the beginning of the data space index creation template to the first byte of a list of space pointers to intermediate key mapping templates.

When the offset to intermediate key mapping templates pointer list contains a non-zero value, for each data space in the data space list there must be a corresponding key mapping template space pointer in the same position in the key mapping templates pointer list. Null pointers in the list are permissible and indicate intermediate key mapping for the corresponding data space is not to be performed.

Each key mapping template space pointer locates a mapping template which defines the operations to be performed on specified fields. The resulting fields will be used to construct the composite key in the index. The resulting fields to be used will be determined by the data space key specifications.

The offset to translate table list addresses an array of translate tables starting at that offset past the start of the template. Each table is 256 bytes long. These tables are available for use by intermediate key mapping when performing the translate operation. The number of tables in the array is designated by the number of translate tables field. A value of binary zero in this field designates no translate tables are provided and the offset to translate table list field is ignored.

For an explanation of a translate table, refer to the example under the alternate collating template definition.

The index logical page size field specifies the number of bytes to use in a logical index page. A value of hex 0000 specifies a system default is to be used. The other allowable values are hex 0200, hex 0400, hex 0800, hex 1000, hex 2000, hex 4000, or hex 8000. All other values are reserved.

The example below shows how a translation table could be organized to cause the numbers 0-9 (hex F0 through hex F9) to appear before the characters A-Z (hex C1 through hex E9) in a collating sequence. To accomplish this ordering, the numbers 0 through 9 (hex F0 through hex F9) must take on the values hex C1 through hex CA and the values hex C1 through hex EF must take on the values hex CB through hex F9. The following translation table causes this to happen.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9
EA	EB	EC	ED	EE	EF	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	FA	FB	FC	FD	FE	FF

Note: C0 is translated to C0
C1 is translated to CB
EF is translated to F9
F0 is translated to C1
F9 is translated to CA
FA is translated to FA

The format of the key mapping template is as follows:

• Number of bytes in key mapping template	Bin(4)
• Mapping type	Char(2)
– Input mapping	Char(1)
Hex 03=Intermediate mapping table provided	
All other values reserved	
– Output mapping (must = hex 04; not applicable	Char(1)
• Reserved (ignored-only for alignment)	Char(10)
• Intermediate Mapping	
– Number of key mapping specifications	Bin(2)
– Create index only options	Char(1)
Use derived results for key	Bit 0
Reserved (binary 0)	Bit 1-7
– Reserved (binary 0)	Char(11)
– Number of data pointers	Bin(2)
– Data pointers (repeated)	Data pointer
—Intermediate key mapping specification (repeated)	Char(32)
—Field location of operand 1	Char(8)
Operand 1 location/type	Bin(2)
Hex 0000 = Data space entry	
Hex 0003 = Intermediate key buffer	
Hex 0005 = Null (type)	
Hex 0006 = Literal (type)	
Unassigned values reserved	
Field number	Bin(2)
Start character	Bin(2)
End character	Bin(2)
or	
Array position of data pointer	Bin(2)
Reserved (binary 0)	Char(4)
or	
Reserved (binary 0)	Char(6)
Result field attributes	Char(5)
Field type	Bin(2)
Field length	Bin(2)
Rounding mode	Char(1)
Hex 40 = Decimal round	
—Operation	Char(1)
—Field location of operand 2	Char(18)
Operand location/type	Bin(2)
Hex 0000 = Data space entry location	
Hex 0003 = Intermediate key buffer (location)	
Hex 0005 = Null (type)	
Hex 0006 = Literal (type)	
Hex 0007 = Translate tablet (type)	
Unassigned values reserved	
Operand description	Char(16)
Field number	Bin(2)
Start character	Bin(2)
End character	Bin(2)
Reserved (binary 0)	Char(10)
or	
Array position of data pointer to literal	Bin(2)
Reserved (binary 0)	Char(14)
or	
Array position of translate table	Bin(2)
Reserved (binary 0)Char(14)	

The key mapping template must be aligned on a 16-byte boundary.

The number of key mapping specifications specifies the number of key mapping specification entries in the key mapping table.

The order of the key mapping specification entries in the intermediate mapping table implicitly specifies the order of the result fields in the key mapping buffer.

The field number entry is the relative location of the associated field in the data space or intermediate key mapping buffer as established by Create Data Space and Create Data Space Index instructions. A value of 1 identifies the first field and so on. When intermediate key mapping is specified, the intermediate key mapping buffer is the source for the fields defined by the data space key specifications. The field number must reference a previously defined field.

The start character and end character fields specify the byte positions in the designated operand 1 or operand 2 fields. A value of binary 0 in these fields designates the entire field will be used. The end character field must contain a value of binary zero when the start character field contains a binary zero. When the start character field contains a non-zero positive value, the end character field must contain a value at least as great. Negative values in these fields are not permissible. A value of 1 specifies the first position and so on.

When the operand type is a literal, the operand description references a data pointer to the associated literal.

The result field attributes for intermediate key buffer mapping describe the result field in the intermediate key mapping buffer. For arithmetic operations, this requires both operands to be defined identically.

The permissible field types are the same as allowed by create cursor intermediate mapping with the exclusion of relative record number field type. The restrictions and operations unique to individual field types are also documented in the create cursor architecture.

Floating point computations performed on behalf of intermediate key mapping will use the floating point attributes defined for ths index.

The permissible field lengths are the same as those allowed for the field types defined for create cursor mapping.

The permissible operations are the same as those allowed by create cursor intermediate mapping.

The format of the user-exit selection template is as follows:

• Selection routine pointer	System pointer
• Selection routine program template pointer	Space pointer
• Data space selection specification (repeated for each data space in the data space list)	Char(*)

Note: The value of the entry shown here with an asterisk (*) is ignored by this instruction.

When the user selection template is specified and when a key is to be put into the data space index, the user exit selection routine is passed a space pointer which addresses an interface buffer.

The selection routine program template pointer contains addressability to the program template used for the creation of the selection routine (see Chapter 8. *Program Management Instructions*). It is ignored by the Create Data Space Index instruction and is materialized by the Materialize Data Space Index Attributes instruction.

A value of binary one in the non-user exit selection field specifies non-user exit selection processing should occur.

The offset to non-user exit selection template pointer list defines the offset from the beginning of the data space index creation template to the first byte of the list of space pointers to non-user exit selection template. A pointer must be supplied for each data space in the data space list. A null pointer value is permissible and non-user exit selection will not be performed for that data space.

The selection templates are addressed by space pointers in the non-user exit selection template list. This is a list of space pointers aligned on a 16-byte boundary. The selection template referenced by a particular slot in the non-user exit selection template list is associated with the data space in the corresponding slot in the data space list. A null pointer in the non-user exit selection template list indicates selection is not specified for that data space.

The format of the non-user exit selection template is as follows:

• Length of selection template	Bin(4)
• Number of selection descriptors	Bin(2)
• Reserved (binary 0)	Char(8)
• Number of data pointers	Bin(2)
• Data pointer (repeated)	Data pointer
• Selection descriptor (repeated)	Char(16)
– Descriptor type	Char(1)
Hex 00 = Operand is a field.	
Hex 01 = Operand is a literal.	
Hex 03 = Operand is a pattern.	
Hex 04 = Operator.	
Unassigned values are reserved.	
– Operation/operand location	Char(7)
Operation descriptor	
Operation	Char(2)
Reserved (binary 0)	Char(5)
or	
Operand location	Char(1)
Field source	
Hex 00 = Data space	
Hex 01 = Reserved	
Hex 02 = Reserved	
Hex 03 = Intermediate key buffer	
Hex 04 = Key field	
Field number	Bin(2)
Starting offset	Bin(2)
Ending offset	Bin(2)
or	
Operand is literal descriptor	
Reserved (binary 0)	Char(1)
(bits 0,1 used internally)	
Reserved (binary 0)	Char(4)
Array position of data pointer	Bin(2)
or	
Operand is a pattern	
Reserved (binary 0)	Char(1)
(bit 0 used internally)	
Offset from start of this template to pattern descriptor array	Bin(4)
Number of pattern descriptors in array	Bin(2)
or	
All other descriptor types	
Reserved (binary 0)	Char(7)
– Reserved (binary 0)	Char(8)
• Pattern descriptor array	Char(*)

The selection template must be aligned on a 16-byte boundary.

The format of a pattern descriptor follows:

• Pattern descriptor	Char(16)
– Descriptor type	Char(1)
Hex 00 = Field	
Hex 01 = Literal	
Hex 02 = Span	
Unassigned values reserved	
– Descriptor	Char(7)
Wild card span type	Char(1)
Hex 01 = Fixed	
Hex 02 = Float	
Unassigned values reserved	
Span width	Bin(2)
(binary 0 for float span)	
Reserved (binary 0)	Char(4)
or	
Field descriptor	
Location	Char(1)
Field source	
Hex 00 = Data space	
Hex 01 = Reserved	
Hex 02 = Reserved	
Hex 03 = Intermediate key buffer	
Unassigned values reserved	
Field number	Bin(2)
Starting offset	Bin(2)
Ending offset	Bin(2)
or	
Literal descriptor	
Reserved (binary 0)	Char(1)
(B it 0,1 used internally)	
Reserved (binary 0)	Char(4)
Array position of of data pointer	Bin(2)
– Reserved (binary 0)	Char(8)

The selection template will be used to perform selection against the specified data in the location given.

The field source of key (binary 010) refers to the field number in the composite key associated with the source data space index when creating the index from another index. This field source may not be specified if not creating an index from another index.

The description of the selection template may be found in the Create Cursor architecture.

It is not allowable to specify both length of selection template as n on-zero (user exit selection specified), and non-user exit selection as binary one (non-user exit selection specified). It is not permissible to specify user exit selection when creating the index from an existing index if the existing source index contains user exit selection.

When the user exit selection template is specified and when a key is to be put into the data space index, the user exit selection routine is passed a space pointer which addresses an interface buffer. The storage for the interface buffer is allocated from the process automatic storage area. The first 2 bytes of the buffer are a return value and must be set by the selection routine to indicate whether addressability to the entry just passed is to be placed in the index. Binary 0 indicates that addressability to the entry is to be included in the index, and any other value indicates that addressability is not to be included in the data space index.

The second 2 bytes of the buffer contain the data space number that indicates the data space from which the fields have been extracted. This number corresponds to the order of the data spaces as specified in the data space list associated with the data space index template.

The data space number is followed by the fields mapped from the data space entry that is being passed to the selection routine. The fields are presented in the buffer as a continuous string.

If an error occurs in the selection routine, a data space index routine failure exception is signaled, and the data space entry is neither inserted or updated.

The data space selection specification entry contains locations and attributes of the fields that are to be passed to the selection routine. The selection routine determines whether or not addressability to the entry is to be placed in the index. The fields are presented to the selection routine in the order of specification and with the attributes described in the template. This implies that values residing in the data space entry may need to be transformed (mapped) into equivalent values while being assembled in the selection buffer. This transformation process may involve conversions and truncations that are data sensitive. If any such conversion or truncation errors are encountered during this transformation, the conversion error checking attribute associated with the data space will govern whether these errors are suppressed or reported as events. The data space selection specification entry has the following format:

- Data space selection specification (repeated for each data space)

– Number of selection fields	Bin(2)
– Field specification (repeated for each field)	Char(6)
Field location	Bin(2)
Field attributes	Char(4)

A data space specification entry must be present for each data space defined for the data space index, and it must be specified in the same order as the data spaces are defined in the data space list. The argument and number of selection fields designate the number of fields (from the data space) that are to be passed to the selection routine.

If the number of selection fields is 0, the selection routine is not invoked, and every entry's key is inserted into the data space index for that particular data space. A field specification entry determines the fields that are passed to the selection routine. The order in which the fields are specified establishes the order in which the corresponding mapped field values appear in the selection buffer for the selection routine. The number of field specification entries must equal the number of selection fields value for that data space. The field location entry specifies the relative field position in the data space entry of the field that is to be passed to the selection routine. The first field in the data space entry is identified by relative position 1. The field attributes entry specifies the attributes that the field is to have when it is passed to the selection routine. The definition and meaning of the field attributes is the same as the field attributes in the Create Cursor instruction mapping templates. The dummy field attribute may be used to align data in the selection buffer, but the contents of the dummy field are binary 0. If a conversion or truncation error occurs in mapping data to the selection buffer and enable conversion error checking was specified for that data space, a data space index selection routine failure exception is signaled and the data space entry is neither inserted nor updated in the data space. If checking was not specified, the selection routine is presented the invalid fields.

Inserting, updating, and deleting data space entries can occur concurrently with creating or rebuilding a data space index over the data space.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
- Insert
  - User profile of creating process
  - Context identified by operand 2
- Object management
  - Data spaces identified by operand 2
- Operational
  - Selection routine identified by operand 2

Lock Enforcement

- Materialize
  - Selection routine identified by operand 2
  - Contexts referenced for address resolution
- Modify
  - User profile of creating process
  - Context identified by operand 2
- Implicit locks
  - The data space index being created is implicitly locked LENR for the duration of this instruction.
  - Source data space index (when creating an index from an index), is locked LSRD during this instruction.
  - The data spaces addressed by operand 2 are implicitly locked LSRD during this instruction.

Events

0002 Authorization
- 0101 Object authorization violation

0008 Data space index
- 0401 Data space entry not addressed by data space index

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded
- 0501 Machine address threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage		X	X
12 Data Base Management			
0B Duplicate key value detected while building a unique data space index		X	
13 Invalid mapping template		X	
14 Invalid selection template		X	
1C Key size limit exceeded		X	
1E Selection routine buffer size limit exceeded		X	
1F User exit routine criteria not satisfied		X	
22 Data space index with selection routine build determination		X	
26 Data space index with invalid floating-point field build termination		X	
33 Data space index with non-user exit selection		X	
39 Derived field operation error during build index		X	

Exceptions

Exception	Operands		Other
	1	2	
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
06 Machine lock limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
05 Object not available to process		X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
06 Instruction cancellation			X
07 Instruction termination			X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
38 Template Specification			
01 Template value invalid		X	

DATA BASE MAINTENANCE (DBMAINT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0413	Data space or data space index	Maintenance option	Number of entries

Operand 1: System pointer.

Operand 2: Character(1) scalar (fixed-length).

Operand 3: Binary scalar or null.

Description: This instruction performs the function identified by the option field in operand 2 on the data space or data space index identified by operand 1. Operand 3 is required for options hex 06, 07, and 08 and is ignored if present for options hex 01-05.

Maintenance Option Value (Hex)	Function to Be Performed	Operand 1
01	Rebuild index	Data space index
02	Invalidate index	Data space index
03	Reset data space	Data space
04	Reserved	
05	Increment maximum number of entries	Data space
06	Insert deleted entries	Data space
07	Insert default entries	Data space
08	Reset data space allocation specified	Data space

Rebuild Index–The invalid data space index identified by operand 1 is rebuilt according to the definition supplied when the data space index was created. If a truncation or conversion error occurs when filling the selection buffer and enable conversion error checking is specified for the data space or if an error occurs within the selection routine, a data space index with selection routine build determination exception is signaled, and the rebuild is terminated. The data space index is not available for the duration of the operation.

Invalidate Index–The data space index is invalidated and no further maintenance is performed on it. The data space index must be rebuilt before it is used again. Storage held by the data space index keys is released. The original definition of the index remains intact. The data space index must not be currently in use by an activated cursor.

Reset Data Space–The data space is reset to an empty status (all data space entries are removed) and all valid data space indexes over the data space are updated to reflect the empty status. A new allocation is obtained based on the creation attributes of the data space. The data space must not be currently in use.

Increment Maximum Number of Entries–The current maximum number of entries limit for the data space specified is incremented by the entry number increment that was specified when the data space was created. This option is used to respond to the data space entry limit exceeded exception that is signaled by the Insert Data Space Entry instruction, the Insert Sequential Data Space Entries instruction, the Update Data Space Entry instruction, the Copy Data Space Entries instruction, or the initialize default entries option of the Data Base Maintenance instruction.

Insert Deleted Entries–The number of entries specified by operand 3 is inserted into the data space specified by operand 1. Since the entries are deleted entries, this operation will not cause the number of entries in the data space to exceed the designated limit, but the compression threshold may be exceeded. If the compression threshold is exceeded, no event will be signaled; however, a subsequent De-Activate Cursor instruction will recognize this condition and signal an event. The number of entries value in operand 3 must be greater than 0.

Insert Default Entries–The number of entries specified by operand 3 is inserted into the data space specified by operand 1. The field values for the inserted entries come from the default values entry in the specified data space. If inserting the entries causes the number of entries (undeleted) in the data space to exceed the designated limit, the corresponding exception is signaled and no entries are inserted. Inserting default entries cannot result in the compression threshold event being signaled. The number of entries value in operand 3 must be greater than 0. An object not eligible for operation exception is signaled if the data space has a data space index defined over it prohibiting duplicate keys.

Reset Data Space with Allocation Specified–The data space is reset to an empty status (all data space entries are removed) and all valid data space indexes over the data space are updated to reflect the empty status. A new allocation is obtained based on the operand 3 value (number of entries). An operand 3 value of zero or greater than the limit for number of entries in a data space will cause a scalar value invalid exception. If the data space has the initial allocation attribute, a new allocation is obtained based on the maximum of the initial allocation value and the operand 3 number of entries value.

The journal entry for reset data space with allocation specified is identical to the option hex 03 reset data space. Performing an Apply Journaled Changes instruction with a reset data space journal entry causes an allocation to be obtained based on the creation attributes for initial allocation. An allocation that is obtained but not currently in use may be returned to the system because of certain conditions or exceptions.

Inserting, updating, and deleting data space entries can occur concurrently with rebuilding a data space index over the data space.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Object management
  - Data space (reset options or insert entries option)
  - Data space index (invalidate option)
- Retrieve
  - Contexts referenced for address resolution
- Delete
  - Data space (reset options)
  - Data space (insert deleted entries)
- Insert
  - Data space (insert default entries)
  - Data space (insert deleted entries)
- Operational
  - Data space index (rebuild option)
  - Data space (increment maximum number of entries option)

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Data space if increment maximum number of entries or insert entries options are specified
- Object control
  - Data space index if invalidate option
- Implicit locks
  - Rebuild option

    Data spaces locked implicitly LSRD
    for the duration of the instruction
    Data space index locked implicitly LEAR
    for the duration of the instruction
  - Reset option

    Data space locked implicitly LENR
    for the duration of the instruction

Events

0002 Authorization
  0101 Object authorization violation

0008 Data space index
  0301 Data space index invalidated

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

001A Journal port
  0301 Entry not journaled

001CJournal space
  0301 Threshold reached
  0401 Journal space attached to a
  journal port became unusable

Exceptions

Exception	Operands			
	1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X		X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation	X			
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X			X
12 Data Base Management				
04 Data space entry limit exceeded	X			
0B Duplicate key value detected while building a unique data space index	X			
22 Data space index with selection routine build determination	X			
23 Data space index selection routine failure				X
26 Data space index with invalid floating-point field build termination	X			
27 Data space index key with invalid floating-point field				X
33 Data space index with non-user exit selection routine build termination				X
34 Non-user exit selection routine failure				X
38 Derived field operation error				X
39 Derived field operation error during build index				X
1A Lock State				
01 Invalid lock state	X			
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded	X			
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
04 Object not eligible for operation	X			
05 Object not available to process	X			
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2C Program Execution				
06 Instruction cancellation				X
07 Instruction termination				X
2E Resource Control Limit				
01 User profile storage limit exceeded	X			
07 Instruction termination				X
30 Journal Management				
02 Entry not journaled				X
32 Scalar Specification				
01 Scalar type invalid			X	
03 Scalar value invalid		X	X	

DE-ACTIVATE CURSOR (DEACTCR)

Op Code (Hex)	Operand 1
0401	Cursor

Operand 1: System pointer.

Description: If the cursor is activated to this process, the cursor is de-activated. All entries locked to this cursor are unlocked. Each data space in use by this cursor is taken out of use. All changed data spaces charged by this cursor are forced to nonvolatile storage. The data space index, if present, is taken out of use. A data space index is forced to nonvolatile storage when the forcing of a changed data space referencing the data space index causes the data space index to no longer reference any unforced data spaces.

The cursor is then disconnected from the process and is available to any process for activation. An event is signaled for each data space in use by the cursor that currently exceeds its compression threshold. If the cursor is not active to this process, an exception is signaled. If the cursor is under commitment control, an operation not valid under commitment control exception is signaled. The cursor must be removed from commitment control before it is deactivated.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Implicit locks
  - Implicit LEAR lock removed from the cursor
  - Implicit LSRD lock removed from the data space
  - Implicit LSRD lock removed from the data space index(es)
  - Implicit LSUP lock removed from data space, if obtained for locked data space entries

Events

0002 Authorization
0101 Object authorization violation

0007 Data space
0301 Data space compression threshold exceeded

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

001A Journal port
0301 Entry not journaled
0401 Journal space attached to a journal port became unusable

001C Journal space
0301 Threshold reached

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	X
04 Object not eligible for operation	X	
05 Object not available to process	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
3E Commitment Control		
10 Operation not valid under commitment control	X	

DELETE DATA SPACE ENTRY (DELDSEN)

Op Code (Hex)	Operand 1
0481	Cursor

Operand 1: System pointer.

Description: The first entry referenced by the cursor's locked entry queue is deleted from the data space in which it resides. The cursor must be activated to this process and must have previously been set (with the lock entry option) to the entry to be deleted. If no entry is locked, an exception is signaled. The deletion of a data space entry from the data space in which it resides does not affect the ordinal numbers assigned to other entries in the same data space. The keys associated with the data space entry that is deleted are removed from all data space indexes over the data space. An implicit LSUP (lock shared update) lock is applied against a data space only when the number of currently locked entries to this cursor from this data space goes from 0 to 1. This LSUP lock is removed only when the number of entries currently locked to this cursor from this data space goes from 1 to 0. If this instruction encounters an abnormal condition, the entry is not deleted or unlocked.

If the data space entry is being deleted under commitment control, the lock that was identified by the locked entry queue for the cursor is transferred to the controlling commit block. Once transferred to the commit block, the data space entry remains locked to the commit block until all uncommitted changes controlled through the commit block are committed or decommitted. If there are any valid unique keyed data space indexes over this data space entry, the key of this entry in that data space index is reserved (an entry may not be added or changed in any data space that conflicts with the deleted key) until this change is committed or decommitted. All views of the data space entry indicate that the entry has been deleted. Because the entry logically remains locked under commitment control, no other user of the data space can change the deleted data space entry. The issuing process can update this entry through this same cursor or another cursor that is under the control of the same commit block.

If this change is later decommitted, the original data space entry is placed back in the data space at its original ordinal entry position.

If a delete operation intent was not indicated during an Activate Cursor instruction, an invalid data base operation exception is signaled.

Authorization Required

- Delete
  - Data space affected
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Implicit locks
  - Implicit LSUP lock removed from the affected data space if the cursor is not under commitment control or the cursor activation options indicate the implicit lock is to be removed.

Events

0002 Authorization
    0101 Object authorization violation

0008 Data space index
    0301 Data space index invalidated

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port became unusable

001CJournal space
    0301 Threshold reached

Exceptions

Exception	**Operand 1**	**Other**
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	X
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
12 Data Base Management		
0D No entries locked	X	
25 Invalid data base operation	X	
37 Operation not valid with join cursor	X	
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
05 Object not available to process	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
30 Journal Management		
02 Entry not journaled		X

DESTROY CURSOR (DESCR)

Op Code (Hex)	Operand 1
0429	Cursor

Operand 1: System pointer.

Description: A previously created cursor is destroyed, and addressability to the cursor is deleted from the context (if any) that addresses the cursor. The system pointer identified by operand 1 is not modified by the instruction and a subsequent reference to the cursor through the pointer causes an object destroyed exception to be signaled. If the cursor is currently activated to this process, the cursor is de-activated before being destroyed. See De-activate Cursor instruction for a description of the de-activate function's authorities, locks, and exceptions. If the cursor is active but not to this process, an exception is signaled. If the cursor is damaged and its state cannot be determined, it is destroyed. If the cursor is under commitment control, an operation not valid under commitment control exception is signaled. The cursor must be removed from commitment control before it is destroyed.

Authorization Required

- Object control
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Object control
  - Operand 1
- Materialize
  - Contexts referenced for address resolution
- Modify
  - Access group which contains operand 1
  - Context which addresses operand 1
  - User profile owning operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port became unusable

001C Journal space
    0301 Threshold reached

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
04 Object not eligible for operation	X	
05 Object not available to process	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
30 Journal Management		
02 Entry not journaled		X
3E Commitment Control		
10 Operation not valid under commitment control	X	

DESTROY DATA SPACE (DESDS)

Op Code (Hex)	Operand 1
0421	Data space

Operand 1: System pointer.

Description: The data space referenced by operand 1 is removed from the system, and addressability to the data space is deleted from the context (in any) that addresses that data space.

The system pointer identified by operand 1 is not modified by the instruction, and a subsequent reference to the data space causes the object destroyed exception to be signaled.

If the data space is currently in-use by this or other processes in the system, or any uncommitted changes are outstanding for the data space an exception is signaled and the data space is not destroyed. In-use means that a cursor is active over the data space, or that the Create Data Space Index or Data Base Maintenance instructions are currently using the data space.

If a data space index refers to this data space, an exception is signaled, and the object is not destroyed.

If the data space is damaged so that its state or the existence of data space indexes referencing it cannot be determined, the data space is destroyed.

Authorization Required

- Object control
  - Operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Object control
  - Operand 1

- Modify
  - Context which addresses operand 1
  - User profile owning operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port became unusable

001C Journal space
    0301 Threshold reached

Exceptions

Exception	**Operand 1**	**Other**
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

DESTROY DATA SPACE INDEX (DESDSINX)

Op Code (Hex)	Operand 1
0425	Data space index

Operand 1: System pointer.

Description: The data space index referenced by operand 1 is removed from the machine, and addressability to the data space index is deleted from the context (if any) that addresses the data space index. The system pointer identified by operand 1 is not modified by the instruction, and a subsequent reference to the data space index causes the object destroyed exception to be signaled.

If the data space index is currently in-use by this or other processes in the system, an exception is signaled. In-use means that a cursor is active over the data space index or that some data base maintenance operation is in progress against this object. If the Data Base Maintenance instruction is currently using the data space index, an exception is signaled and the data space index is not destroyed.

If the data space index is damaged and its state cannot be determined, it is destroyed.

Authorization Required

- Object control
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Object control
  - Operand 1
- Modify
  - Context which addresses operand 1
  - User profile owning operand 1

Events

0002 Authorization
 0101 Object authorization violation

000C Machine resource
 0201 Machine auxiliary storage threshold exceeded

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation		X
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X

ENSURE DATA SPACE ENTRIES (ENSDSEN)

Op Code (Hex)	**Operand 1**
0499	Cursor

Operand 1: System pointer.

Description: The instruction ensures that all changes to data space entries that have resulted from operations involving the identified cursor since it was activated to this process are forced to nonvolatile storage. The referenced cursor must have been activated to this process. At the completion of the instruction, all data base changes (entries that were inserted, updated, or deleted) made through this cursor are recorded on nonvolatile storage. The instruction does not directly ensure the data space indexes that reference the data space. Therefore, on a system failure, the indexes may have to be rebuilt even though the Ensure Data Space Entries instruction was issued. If, however, the ensuring of a data space results in no uninsured data spaces being referenced by a data space index, then the data space index is also ensured to reduce the chance of it being invalidated.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

001A Journal port
  0301 Entry not journaled
  0401 Journal space attached to a journal port became unusable

001C Journal space
  0301Threshold reached

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	X
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	X
03 Object suspended	X	
05 Object not available for process	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
30 Journal Management		
02 Entry not journaled		X

ESTIMATE SIZE OF DATA SPACE INDEX KEY RANGE (ESTDISKR)

Op Code (Hex)	Operand 1	Operand 2
0432	Options template	Cursor

Operand 1: Space pointer.

Operand 2: System pointer.

Description: This instruction estimates the number of records in a given range in a data space index to facilitate performance decisions involving the data space index under the operand 2 cursor and the underlying data space specified in the operand 1 option template.

Notes:

1. The operand 2 cursor must be active and must be over a data space index or a 2204 exception (object not eligible for operation) will be signaled.
2. If the operand 2 cursor has an associated selection template, this will not be taken into account in the estimate returned.

The format of the options template is as follows:

• Input options	Char(2)
– Depth cutoff choice	Bit 0
– Early exit specified	Bit 1
– Field/byte count choice	Bit 2
– Key count specified for 1st key	Bit 3
– Key count specified for 2nd key	Bit 4
– Reserved (binary 0)	Bits 5-15
• Key 1 count	Bin(2)
• Key 2 count	Bit(2)
• Granularity of accuracy	Bin(2)
• Data space number	Bin(2)
• Early exit criterion	Bin(4)
• Reserved (binary 0)	Char(2)
• First search key	Space pointer

• Last key search	Space pointer
• Return code	Char(2)
• Logical page size (return value)	Char(2)
• Total number of keys in tree (return value)	Bin(4)
• Total number of pages in tree (return value)	Bin(4)
• Page density (return value)	Bin(4)
• Estimated number of logical pages in specified key range (return value)	Bin(4)
• Key estimate (return value)	Bin(4)
• Reserved (binary 0)	Char(32)

The options template must be aligned on a 16-byte boundary.

If the depth cutoff choice indicator in the input options has a value of binary one, then gradularity of accuracy will be used to restrict the depth of the search thru. the index. If the depth cutoff/choice indicator in the input options has a value of binary zero, then the index will be searched up to level n-1, where n is the deepest level of the index corresponding to the specified key range. The granularity of accuracy field must be zero if the depth cutoff choice has a value of binary zero or a template value invalid exception will be signaled.

Granularity of accuracy is interpreted as a percentage giving the effective depth of tree search to be performed. The larger the value, the deeper the search of the index tree, and thus the more accurate the estimate. This value must be greater than or equal to 100 otherwise, a template value invalid exception will be generated. The deepest level to search is calculated by multiplying the percentage times the maximum level of the first and last keys and truncating to the nearest integer.

Note: A percentage of 100% will still return an estimate.

The data space number is with respect to the operand 2 cursor and is used to designate the variety of key mapping and key building to perform. It must have a value greater than zero and less than or equal to the number of data spaces under the cursor. If the data space is not in the active subset of data spaces under this cursor, a template value invalid exception will be generated.

The first search key specifies the low end of the search range. This is the key of the least magnitude of the two input keys after the user key has been mapped and the appropriate key massaging (for alternate collating sequence, descending collating sequence, and/or numeric ordering) has been applied.

The last search key specifies the high end of the search range. This is the key of the greatest magnitude of the two input keys after the user key has been mapped and the appropriate key massaging (for alternate collating sequence, descending collating sequence, and/or numeric ordering) has been applied.

Enough space to contain the maximum logical key length must be allowed for both the low and the high keys because the entire logical key will be copied from the user's space.

If the field/byte count choice has a value of binary one, then key count specified for first key, key count specified for second key, key1 count, and key2 count are used as *byte* counts. A byte count indicates that the indicated number of bytes beginning with the leftmost byte of the mapped key will be used as the key with which to scan the index.

Otherwise, if the count choice has a value of binary zero, then the above key count options are used as field counts. A field count indicates that the indicated number of fields beginning with the leftmost field of the mapped key will be used as the key with which to scan the index.

The key processing for byte and field counts is identical to that performed in the Set Cursor instruction. Refer to the set cursor architecture for more information.

If the key count specified for first key indicator in the input options has a value of binary one, then key 1 count must have a positive integer value which specifies the number of fields or bytes assumed to be in the key value in the first search key to be used. If the key count specified for first key indicator in the input options has a value of binary one and the key 1 count is less than zero, a template value invalid exception will be signaled. If the key 1 count is zero and there are no leading fork characters in the key, then a template value invalid exception will be signaled.

The key count specified for second key and key 2 count are defined analogously.

Trailing fork characters will be appended to the respective last field.

If the key count specified for first key indicator in the input options has a value of binary 0, then the search will start at the beginning of the index. Similarly, if the key count specified for second key indicator in the input options has a value of binary 0, then the search will end at the last entry of the index.

If the early exit specified indicator in the input options has a value of binary one, the early exit criterion must have a value which is used as an upper bound on the estimate. The search is terminated if and when the estimated number of entries in the range of interest ever exceeds this value.

If the early exit specified indicator in the input options has a value of binary zero, the early exit criterion must be zero or a template value invalid exception will occur.

The key estimate is the number of keys in the designated key range. It is the product of the estimated number of logical pages in specified key range and page density. The estimated number of logical pages in specified key range is the number of logical pages found up to the depth percentage specified by the granularity of accuracy. The page density is the approximate number of keys per logical page.

Total number of keys in the tree is the number of keys in the index under the operand 1 cursor. This value includes keys from all data spaces under the index.

Total number of pages in the trees is the number of logical pages currently in use.

Logical page size is the number of bytes in a logical page of the index. (Logical page size is always an even multiple of physical page size.)

The return codes are assigned as follows:

0 = Successful
1 = 1 key in range
2 = Search path did not diverge (no range at specified depth)
3 = Early exit
4 = Zero keys in range
5 = Logically damaged index
6 = Unsupported index structure
7 = Keys not in ascending order after mapping

The following table indicates which of the output fields are set for every possible return code:

Output Fields	Return Code			
	0000	0001 or 0002	0003 or 0004 or 0005	0006 or 0007
Logical page size	Set	Set	Set	0
Total number of keys in tree	Set	Set	Set	0
Total number of pages in tree	Set	Set	Set	0
Page density	Set	Set	Set	0
Estimated number of logical pages in specified key range	Set	Set	0	0
Key estimate	Set	0	0	0

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
  - All data spaces under the operand 2 cursor

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002Authorization
  0101 Authorization violation

0008Data space index
  0301 Data space index invalidated

000CMachine resources
  0201 Machine auxiliary storage exceeded

000DMachine status
  0101 Machine check

0010Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016Machine observation
  0101 Instruction reference

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	X
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage		X	X
12 Data Base Management			
02 Mapping error			X
07 Data space index invalid			X
08 Incomplete key description			X
27 Data space index with invalid floating point field			X
1A Lock State			
01 Invalid lock state		X	
1C Machine Dependent Exception			
03 Machine storage limit exceeded			X
06 Machine lock limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object destroyed	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
04 Object not eligible for operation		X	
24 Pointer Specification			
01 Pointer does not exist		X	X
02 Pointer type invalid		X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand		X	
07 Invalid operand attribute		X	
08 Invalid operand value range		X	
0C Invalid operand ODT reference		X	
0D Reserved bits are not zero	X	X	
2E Resource Control Limit			
02 Process storage limit exceeded			X
38 Template Specification			
01 Template value invalid	X		

INSERT DATA SPACE ENTRY (INSDSEN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0483	Cursor	Option list	Interface buffer

Operand 1: System pointer.

Operand 2: Character(7) variable scalar (fixed-length).

Operand 3: Space pointer.

Description: Data values in the interface buffer addressed by the operand 3 space pointer and control values designated the operand 2-option list are used to create and insert a new data space entry into the data space identified by the operand 1 cursor (which must be activated to this process). The order of the data fields in the interface buffer is assumed to be the same order as defined in the Create Cursor instruction input mapping template for that particular data space.

The ordinal entry number assigned to the new data space entry is returned in the option list upon completion of this instruction. All valid data space indexes addressing the data space are updated. A check for duplicate keys is made on data space indexes that have the unique attribute. If no duplicate keys are found, all the indexes are updated. If a duplicate key is found, no indexes are updated, and the entry is not inserted into the data space. For any field not specified in the cursor's input mapping template, the corresponding value from the data space's default values entry is used.

Any data sensitive mapping error encountered during the presenting of the new entry to the user exit routine, associated with a select/omit data space index that references the data space, causes the data space index to be invalidated and an event is signaled.

The option list has the following format:

• Data space requested	Bin(2)
• Ordinal entry number assigned	Bin(4*)
• Control attributes	Char(1)
– Forced write option	Bit 0
– Reserved (binary 0)	Bits 1-7

Note: The value of the entry shown here with an asterisk (*) is returned by this instruction.

The data space requested field must always be supplied and indicates the data space into which the entry is to be inserted. The value is the data space number which corresponds to the data space in the data space list identified by Create Cursor or Activate Cursor instructions.

A value of 1 in the forced write option bit causes the entry to be written immediately to nonvolatile storage.

If an attempt is made to insert an entry that would cause the maximum number of entries limit to be exceeded, the data space entry limit exceeded exception is signaled, and the entry is not inserted.

The current addressing of an entry by the cursor for retrieving, updating, or deleting is unaffected by the intervening execution of the insert instruction.

If the data space entry is being inserted under commitment control, a lock is placed on the new entry and the lock is held by the controlling commit block. The new entry remains locked to the commit block until all uncommitted changes controlled through the commit block are committed or decommitted. All views of the data space entry indicate that the entry has been inserted into the indicated data space. Because the new entry remains locked, no other user of the data space can change the inserted data space entry. The issuing process can update this entry through this same cursor or through another cursor that is under the control of the same commit block. If this entry is later decommitted, the ordinal entry of the previously inserted data space entry contains a deleted data space entry.

If an insert operation intent was not indicated when the Activate Cursor instruction was performed, an invalid data base operation exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Insert
  - Data space affected
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Data space affected
- Implicit locks
  - If the data space is not currently implicitly locked LSUP or higher and the cursor is under commitment control, an LSUP lock is implicitly placed on the data space until either a commit or a decommit is performed.

Events

0002 Authorization
  0101 Object authorization violation

0008 Data space index
  0301 Data space index invalidated

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

001A Journal port
  0301 Entry not journaled
  0401 Journal space attached to a journal port became unusable

001C Journal space
  0301 Threshold reached

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation	X			X
10 Damage Encountered				
04 System object damage state	X		X	X
44 Partial system object damage	X		X	X
12 Data Base Management				
01 Conversion mapping error				X
04 Data space entry limit exceeded				X
09 Duplicate key value in existing data space entry				X
0F Duplicate key value in uncommitted data space entry				X
21 Unable to maintain unique key DSI				X
23 Data space index select routine failure				X
25 Invalid data base operation				X
27 Data space index key with invalid floating-point field				X
34 Non-user exit selection routine failure				X
36 No mapping code specified	X			
37 Operation not valid with join cursor	X			
38 Derived field operation error				X

Exceptions

Exception	Operands 1	2	3	Other
1A Lock State				
01 Invalid lock state	X			X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	X
03 Object suspended	X	X	X	
05 Object not available to process	X			
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length		X		
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
2E Resource Control Limit				
01 User profile storage limit exceeded				X
02 Process storage limit exceeded				X
30 Journal Management				
02 Entry not journaled				X
32 Scalar Specification				
01 Scalar type invalid				X
02 Scalar attributes invalid		X		
03 Scalar value invalid		X		
3E Commitment Control				
06 Commitment control resource limit exceeded	X			

INSERT SEQUENTIAL DATA SPACE ENTRIES (INSSDSE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0487	Cursor	Option template	Interface buffer

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: Space pointer.

Description: Information contained in the interface buffer addressed by the operand 3 space pointer is used to create and insert new data space entries into the data space identified by the operand 1 cursor (which must be activated to this process) and the operand 2 option template. The order of the fields in each entry in the interface buffer is assumed to be the same order as defined in the Create Cursor instruction input mapping template for that particular data space.

Each entry (the total number to be inserted is identified in the option template) is assumed to begin in the first position of the next interface buffer entry (the length of each entry in the buffer is defined in the option template).

All data space indexes addressing the data space are updated accordingly.

The option template has the following format:

• Data space requested	Bin(2)
• Control attributes	Char(2)
– Forced write option	Bit 0
– Reserved	Bits 1-15
• Buffer entry length	Bin(2)
• Number of entries	Bin(2)
• Ordinal entry number	Bin(4)*
• Interface buffer position	Bin(2)*

Note: The value associated with each entry shown here with an asterisk (*) is modified by this instruction.

The data space requested field must always be supplied and indicates the data space into which the entries are to be inserted. The data space number must correspond to the position this data space occupied in the data space list identified by the Create Cursor or the Activate Cursor instructions.

A forced write option value of 1 causes the new entries to be immediately written to nonvolatile storage.

The buffer entry length field defines the starting position of each entry relative to the beginning of the previous entry in the interface buffer. The first entry always begins in position 0 of the interface buffer. If the buffer entry length was 200, for example, the second buffer entry would begin in position 200, the third in position 400, and so on. The data space entry is created by performing the operations/conversions defined in the input mapping template for the designated data space in the Create Cursor instruction. Mapping begins with the first position of the buffer entry and may continue into other buffer entries. The buffer entry length must be greater than or equal to 0.

The number of entries field indicates the total number of entries to be mapped from the interface buffer to the data space. This field must have a value greater than 0.

The ordinal entry number assigned to the last entry inserted into the data space is returned in the option list upon successful completion of this instruction.

The interface buffer position (identifying the entry in the interface buffer that caused certain exception conditions) are returned when those exceptions are signaled. A value of 1 indicates the first entry, a value of 2 indicates the second, and so on. A value of 0 indicates that there were no exceptions.

A check for duplicate keys is made on data space indexes (over the data space) that have the unique attribute for each entry in the interface buffer. If a duplicate is found (duplicates may occur among entries within the interface buffer), the insert fails and none of the entries are inserted. The interface buffer position is updated to indicate which entry in the interface buffer was a duplicate key. No attempts are made to find subsequent errors. If no duplicate keys are found, all of the indexes are updated for each entry. Conversion mapping errors result in similar instruction completion.

Any data sensitive mapping error that is encountered during the presenting of one of the new entries to the user exit routine (associated with a select/omit data space index referencing the data space) causes the data space index to be invalidated and an event to be signaled.

If the insertion of one of the entries attempts to cause the maximum number of entries to be exceeded, the data space entry limit exceeded exception is signaled and none of the entries are inserted.

For any field not specified in the cursor's input mapping template, the corresponding value from the data space's default values entry is used.

The current addressing of any entry by the cursor for retrieval, updating, or deletion is unaffected by the intervening execution of this instruction.

If the indicated cursor is under commitment control, an operation not valid under commitment control exception is signaled and the operation is terminated.

If an insert operation intent was not indicated when the Activate Cursor instruction was performed, an invalid data base operation exception is signaled.

Authorization Required

- Insert
  - The data space affected

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Data space affected

Events

0002 Authorization
- 0101 Authorization violation

0008 Data space index
- 0301 Data space index invalidated

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

000D Machine status
- 0101 Machine check

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

001A Journal port
- 0301 Entry not journaled
- 0401 Journal space attached to a journal port became unusable

001C Journal space
- 0301 Threshold reached

Exceptions

Exception	**Operands** 1	2	3	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation				X
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
12 Data Base Management				
01 Conversion mapping error				X
04 Data space entry limit exceeded				X
09 Duplicate key value in existing data space entry				X
0F Duplicate key value in uncommitted data space entry				X
25 Invalid data base operation				X
27 Data space index key with invalid floating-point field				X
34 Non-user exit selection routine failure				X
36 No mapping code specified	X			
37 Operation not valid with join cursor	X			
38 Derived field operation error				X
1A Lock State				
01 Invalid lock state	X			X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	X
03 Object suspended	X	X	X	
05 Object not available to process	X			
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			

Exception		Operands 1	2	3	Other
2A	Program Creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length		X		
	0C Invalid operand ODT reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program Execution				
	06 Instruction cancellation				X
	07 Instruction termination				X
2E	Resource Control Limit				
	01 User profile storage limit exceeded				X
30	Journal Management				
	02 Entry not journaled				X
38	Template Specification				
	01 Template value invalid		X		
3E	Commitment Control				
	10 Operation not valid under commitment control				X

MATERIALIZE CURSOR ATTRIBUTES (MATCRAT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
043B	Receiver	Cursor	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar (fixed-length).

Description: The operational statistics or the creation template associated with the cursor identified by the operand 2 system pointer is materialized into the space identified by operand 1. The materialization options specified by operand 3 determine the information to be materialized: Hex 00 signifies the creation template, and hex 01 signifies the statistics.

If statistics are requested and the cursor is not activated to the current process, an exception is signaled. If statistics are requested, the cursor is activated to the current process, and the cursor is not set, then only the materialization length and cursor attributes portion of the statistics are materialized.

If the cursor type is join, only the statistics for the first data space in the data space list will be materialized.

If the creation template is specified, a similar template is materialized. (See the Create Cursor instruction, earlier in this chapter, for a definition of the template.)

If the cursor was created with the option to make the cursor creation template not materializable, then an object not eligible for operation exception will be signaled when materialize creation template option is specified.

The values in the new template are as specified at the creation of the cursor except in the following cases:

- Current values are provided for the object identification, initial context, context, and size of associated space.
- The pointers specifying the various templates may differ because they are built contiguously in the receiver operand 1.

The format of the materialization output for statistics is as follows:

• Materialization length	Char(8)
– Number of bytes provided by the user	Bin(4)
– Number of bytes that can be materialized	Bin(4)
• Cursor attributes	Char(10)
– Cursor status	Char(2)
Reserved (binary 0)	Bits 0-3
Cursor under commitment control 0 = No commitment control 1 = Commitment control	Bit 4
Commitment control lock indicator 0 = No LSUP lock on a data space under the cursor 1 = LSUP lock on a data space under the cursor	Bit 5
Reserved (binary 0)	Bits 6-14
Cursor addressability set 0 = Cursor not set 1 = Cursor set	Bit 15
– Number of locked entries referenced by locked entry queue	Bin(2)
– Data space number of the first entry referenced by the locked entry queue	Bin(2)
– Ordinal entry number of the first entry referenced by the locked entry queue	Bin(4)
• Option list	Char(*)
– Length of option list	Bin(4)
– Rule option	Char(1)
– Search attributes	Char(1)
– Control attributes	Char(1)
– Key field count	Char(1)
– Relative/ordinal number	Bin(4)
– Data space key format	Bin(2)
– Data space number	Bin(2)
– Ordinal entry number	Bin(4)
– Number of data spaces in the following restricted search list	Bin(2)
– Data space included in the restricted search list (1 to 32); repeated for each data space	Bin(2)
• Data space entry key	Char(*)

The cursor under commitment control value of binary 1 indicates that the cursor is under commitment control. A value of binary 0 in this field indicates that the cursor is not under commitment control.

The commitment control lock indicator value of binary 1 indicates that commitment control holds a LSUP lock on at least one data space under the cursor. This bit is valid only if the cursor under commitment control value is binary 1.

The cursor set attribute indicates that the cursor currently addresses an entry for retrieval. The values in the option list are those used in the last successful Set Cursor instruction operation except key field count, which is the number of fields in the materialized key. A key count of 0 indicates a key is not materialized. The restricted search list materialized does not contain duplicate occurrences of the same data space; the entries are in ascending order.

A Set Cursor instruction that used a search attribute of binary 1 in the position indicator field is not materialized because the cursor positioning is not recorded in the cursor. Only a data space entry lock acquired during the Set Cursor instruction can be materialized as part of the cursor attributes field.

The data space entry key is the key associated with the entry addressed for retrieval by the cursor. This key is for the entry indicated by the data space number and ordinal entry number materialized in the option list. A key is materialized only if the cursor is over a data space index, every key field was specified in the cursor output mapping template for that data space, retrieve authority for that data space is satisfied, and the entry is not deleted from the data space or omitted from the data space index. The fields within the key are ordered as specified in the data space key specification for the data space in the Create Data Space Index instruction; the fields have the same attributes as specified in the output mapping template in the Create Cursor instruction. Fork characters are not in the materialized key.

The first 8 bytes of the materialization output in both forms of the materialization identify the total number of bytes provided and the number of bytes that can be materialized.

If fewer than 8 bytes are available in the space identified by the receiver operand, a materialization length exception is signaled. The instruction materializes as many bytes as can be contained in the receiver's space. If the space of the receiver is greater than that required to contain the materialization, the excess bytes are unchanged. When a key is materialized, additional bytes are set to binary 0 if the key is shorter than the longest key defined by the data space index and the cursor output mapping template.

No exceptions (other than the materialization length exception) are signaled when the receiver contains insufficient space for the materialization. If the cursor creation template is specified, the receiver must be aligned on a multiple of 16 bytes.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Data space referenced, if a key is materialized
  - Contexts referenced for address resolution
- Operational
  - Operand 2

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
  - Operand 2

Events

0002 Authorization
    0101 Object authorization violation

0008 Data space index
    0301 Data space index invalidated

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	 2	 3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X		
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
12 Data Base Management				
02 Key mapping error				X
38 Derived field operation error				X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	X
03 Object suspended	X	X	X	
04 Object not eligible for operation		X		
05 Object not available to process		X		
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MATERIALIZE DATA SPACE ATTRIBUTES (MATDSAT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0437	Receiver	Data space	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar (fixed-length).

Description: The operational statistics or the creation template associated with the data space identified by the operand 2 system pointer is materialized into the byte area identified by operand 1. The materialization option parameter specified by operand 3 determines the information to be materialized. The format of the option follows:

• Materialization options	Char(1)
– Reserved (binary 0)	Bits 0-3
– Suppress index related statistics	Bit 4
– Suppress references to default entry	Bit 5
– Suppress references to the field table	Bit 6
– Materialize option 0 = Creation template 1 = Operational statistics	Bit 7

A value of binary 1 in the materialize option field indicates that the operational statistics for the data space should be materialized. A value of binary 0 in the materialize option field indicates the creation template should be materialized.

The index attributes materialized will be materialized from contents in the data space header. The invalid statistic is not guaranteed to be the most recent. It may indicate the index is in a valid state when the index is in an invalid state. When it indicates the index is in an invalid state, it is guaranteed to be in that state at that time.

If the creation template is requested, the instruction materializes a copy of the template as defined in the Create Data Space instruction. Values in the creation template are as specified at the creation of the data space, with the following exceptions. The object identification, initial context, context, size of the associated space, contiguous return bit, unit return bit, initial allocation, entry number increment, compression threshold, and the maximum number of entries contain the current values. The entry definition table and default values entry are contiguous in the space provided. If no default values entry was provided in the creation template, the machine defaults are materialized.

A value of binary 1 in the suppress references to the default entry field indicates the default entry is not materialized as part of the creation template, and the offset to the default entry field in the creation template materialized is 0. However, the length of the default entry is materialized as described in the Create Data Space instruction template. A value of binary 1 in this field without a value of binary 0 in the materialize options field (materialize creation template) will result in a scalar value invalid exception.

A value of binary 1 in the suppress references to the field table field indicates the field definition table is not materialized as part of the creation template, the offset to the entry definition table field in the Create Data Space instruction template materialized is 0, and the length of the entry definition table field in the Create Data Space instruction template materialized is 0. A value of binary 1 in this field without a value of binary 0 in the materialize option field (materialize creation template) will result in a scalar value invalid exception.

If statistics are requested, the materialization has the following format:

• Materialization length	Char(8)
– Number of bytes provided by the user	Bin(4)
– Number of bytes that can be materialized	Bin(4)
• Number of entries	Bin(4)
• Number of deleted entries	Bin(4)
• Size of the data space	Bin(4)
• Number of distinct data space indexes over the data space	Bin(2)
• Reserved (binary 0)	Char(10)
• Data space index pointer (repeated for each distinct data space index)	System pointer
• Data space index attributes (repeated for each data space index pointer materialized)	Char(2)
– Type = select omit	Bit(0)
– Type = unique	Bit(1)
– Invalidated by user	Bit(2)
– Invalidated by machine	Bit(3)
– Index in logging mode	Bit(4)
– Reserved	Bit(5-15)

The first 8 bytes of the materialization output in both materialization options identify the total number of bytes provided by the user for materialization and the total number of bytes available to be materialized. If fewer than 8 bytes are available in the space identified by the receiver operand, a materialization length exception is signaled. The instruction materializes as many bytes as can be contained in the receiver's space. If the space of the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled in the event that the receiver contains insufficient space for the materialization. The receiver must be aligned on a multiple of 16 bytes.

The number of entries is the number of retrievable entries in the data space. This number is the number of entries that have been inserted minus the number of entries that are deleted. This number includes any entries that have been inserted under commitment control but have not been committed or decommitted.

Deleted entries occupy space in a data space, and the number of deleted entries provides an indication of how much space they occupy. This number includes any entries that have been deleted under commitment control but have not been committed or decommitted.

The number of entries and the number of deleted entries returned by this instruction may not be accurate if system failures occur during the data space update functions (Delete Data Space Entry, or Update Data Space Entry instructions). These values are used when the data space entry limit exceeded exception or the data space compression threshold exceeded event is signaled.

The size of the data space indicates the total space taken up on auxiliary storage by the data space.

A value of binary 1 in the suppress index related statistics field indicates the statistics related to data space indexes (number of distinct data space indexes over the data space and data space index pointers fields) should be suppressed (binary 0). A value of binary 1 in the suppress index related statistics field without a value of binary 1 in the materialize option field results in a scalar value invalid exception.

A system pointer is provided for each distinct data space index addressing the specified data space if the statistics related to data space indexes are not suppressed.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Operational
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X		
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MATERIALIZE DATA SPACE INDEX ATTRIBUTES (MATDSIAT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0433	Receiver	Data space index	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar (fixed-length).

Description: The operational statistics or the creation template associated with the data space index identified by the operand 2 system pointer is materialized into the space identified by operand 1. The materialization options specified by operand 3 determine the information to be materialized: Hex 00 signifies the creation template; hex 01 signifies the operational statistics without the resetting of the timestamp and counts; and hex 02 signifies the operational statistics with the resetting of the timestamp and counts.

If the creation template is requested, the instruction materializes a copy of the creation templates as defined in the Create Data Space Index instruction for the data space index. Values in the template are as specified at the creation of the data space index, with the following exceptions. The object identification, initial context, context, size of the associated space, and the unit return bit contain current values. Any attributes that can be modified by the Modify Data Space Index Attributes instruction also contain the current values. In case this index was created from an existing index, the materialized templates may show the values as adopted/merged from the current index. The pointers that specify the various templates may be different because they are built contiguously in the space provided. The pointer to the selection routine is set to 16 bytes of binary 0, and a space pointer to the selection routine program template is materialized. The program template that is materialized has the following special values set:

- Number of bytes available for materialization is 0
- Initial context is binary 0
- Size of space is set to 0
- Context pointer is null

If data space index operational statistics are requested, the materialization has the following format:

• Materialization length	Char(8)
– Number of bytes provided by the user	Bin(4)
– Number of bytes that can be materialized	Bin(4)
• Size of the data space index	Bin(4)
• Timestamp of this materialization	Char(8)
• Timestamp acquired from the data space index	Char(8)
• Data space index status	Char(2)
– Reserved (binary 0)	Bits 0-13
– Index in logging mode	Bit 14
– Data space index invalid	Bit 15
• Data space status (repeated for each data space addressed by the index)	Char(12)
– Number of entries addressed by the index	Bin(4)
– Number of non-entries in this data space	Bin(4)
– Number of accesses to the data space using this index	Bin(4)
• Number of entries in logging sid	Bin(4)
• Reserved	Char(12)

The size of the data space index indicates the total space occupied on auxiliary storage by the data space index. The timestamp of this materialization is the current machine timestamp. The current timestamp is also stored in the data space index if materialization option hex 02 is specified. The timestamp from the data space index is the timestamp stored in the object at creation or at the last materialization with option hex 02 on this data space index. Timestamps are 64-bit unsigned binary values. Bit 41 equals 1024 microseconds.

The data space index invalid status indicates that the data space index needs to be rebuilt before it can be used. The data space index invalid indication is also returned if the data space index is in the process of being rebuilt by another process or a delayed maintenance index is in overflow condition.

The data space status indicates, for each data space addressed by the data space index, the number of entries in the data space index due to that data space, the number of non-deleted entries in that data space, and the number of accesses to the data space index due to that data space. If the data space index is invalid or being rebuilt, or is in overflow condition, the data space status fields (except number of non-deleted entries) field will represent the old values and may not be correct.

The number of entries addressed by the index is the number of retrievable entries in the data space except when the data space index selection routine has omitted some entries. If any data space entries have been inserted, deleted, or updated under commitment control but have not been committed or decommitted, they appear in their present state. If this data space index has been created with the delayed maintenance option, then these numbers reflect the statistics at the time of the most recent cursor activation over the data space index or at the time of the most recent rebuild of the data space index. If the data space index is currently being referenced through an active cursor, the statistics are current.

The number of accesses to the data space is the number of cursor positioning operations completed that address this data space. A materialization option of hex 02 resets this number to 0.

The order of the data space status entries in the materialization output is the same as the order in which the data spaces were defined when the index was created.

The number of entries in logging sid is the number of entries that will be processed during catch-up time. If the index is not in logging mode or the logging sid overflow bit is on, this field will contain binary zeros.

The first 8 bytes of the materialization output in the materialization options identify the total number of bytes provided by the user for materialization and the total number of bytes available to be materialized. If fewer than 8 bytes are available in the space identified by the receiver operand, a materialization length exception is signaled. The instruction materializes as many bytes as can be contained in the receiver's space. If the space of the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions (other than the materialized length exception) are signaled in the event that the receiver contains insufficient space for the materialization. If the creation template is specified, the receiver must be 16-byte aligned.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Operational
  - Operand 2

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	 2	 3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X		
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MODIFY DATA SPACE ATTRIBUTES (MODDSAT)

Op Code (Hex)	Operand 1	Operand 2
043A	Data space	Data space modification template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction modifies the data space specified by operand 1 to the data space attributes specified by operand 2.

Additional information on data space attributes is available under the description of the Create Data Space instruction.

The format of the operand 2 data space modification template is as follows:

• Modification indicators (must be hex 14005001)	Char(4)
• Indicator attributes	Char(2)
– Reserved (binary 0)	Bits 0-2
– Allocation	Bit 3
0 = Use default	
1 = Use template value	
– Reserved (binary 0)	Bits 4-15
• Reset maximum number of entries	Bin(4)
• Entry number increment	Bin(2)
• Unit identification	Char(1)
• Compression threshold	Char(1)
• Reserved (binary 0)	Char(22)
• Current maximum number of entries	Bin(4)

The data space modification template must be aligned on a multiple of 16 bytes.

The modification indicators field must specify a value of hex 14005001. Any other value causes a template value invalid exception to be signaled.

The allocation indicator attribute specifies whether a machine default value or a template field value is to be used for certain functions of the Data Base Maintenance instruction. The template fields related to this indicator attribute are the reset maximum number of entries field and the entry number increment field.

The reset maximum number of entries field specifies the value to be used to reset the current maximum number of entries attribute for the data space when the data base maintenance reset data space function is performed on it. This value can optionally be specified for use as the allocation size the data space is to be reset to. When the allocation indicator attribute specifies use template value, this field also specifies the number of entries for which the data space is to be reallocated on the reset and it must contain a value greater than zero. When the allocation indicator attribute specifies use default, an internal machine default is used for the number of entries for which the data space is to be reallocated on the reset, and this field must contain a value greater than or equal to zero. In either case, the field value is used to reset the current maximum number of entries attribute as described above and must not be greater than the maximum number of entries allowed in a data space. If, in the latter case, this field contains a value of zero, it specifies that no limit is to be made on the number of entries the data space can contain. If any of the restrictions on the value of this field are violated, the template value invalid exception is signaled.

The entry number increment field optionally specifies the value to be used to increment the current maximum number of entries data space attribute when the data base maintenance increment maximum number of entries function is performed on the data space. When the allocation indicator attribute specifies use template value, this field specifies the number of entries value for the increment function. In this case, it must contain a value greater than or equal to zero or the template value invalid exception is signaled. A value of zero specifies that the increment function is not to be allowed for the data space. When the allocation indicator attribute specifies use default, an internal machine default is used for the number of entries value for the increment function and the value of this field is ignored.

The unit identification field specifies the auxiliary storage unit on which the data space should be reallocated when the data base maintenance reset data space function is performed on it. A value of zero specifies that system default allocation should be performed. If the value specifies an invalid unit ID, the template value invalid exception is signaled.

The compression threshold field specifies the percentage of deleted entries that can remain in the data space before the data space compression threshold exceeded event is signaled. The value can range from 0 to 100. A value of zero specifies that the event should never be signaled. If the value is outside the valid range, the template value invalid exception is signaled.

The current maximum number of entries field specifies the number of entries that can reside in the data space before the data space entry limit exceeded exception is signaled. The value must be greater than or equal to zero. A value of zero specifies that no limit is to be made on the number of entries the data space can contain. A positive value must be greater than or equal to the actual number of entries currently in the data space, must be greater than or equal to the value specified for the reset maximum number of entries field, and must be less than the maximum number of entries allowed in a data space. If any of the restrictions on the value of this field are violated, the template value invalid exception is signaled. Note that this data space attribute will also be modified when either of the data base maintenance increment maximum number of entries or reset data space functions is performed on the data space.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Object management
  - Data space identified by operand 1

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modification
  - Data space identified by operand 1

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X		
02 Boundary alignment	X	X		
03 Range	X	X		
06 Optimized addressability invalid	X	X		
08 Argument/Parameter				
01 Parameter reference violation	X	X		
0A Authorization				
01 Unauthorized for operation	X			
10 Damage Encountered				
04 System object damage	X	X		
44 Partial system object damage		X		
1A Lock State				
01 Invalid lock state	X			
1C Machine Dependent Exception				
03 Machine storage limit exceeded				X
06 Machine lock limit exceeded				
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X		
02 Object destroyed	X	X		
03 Object suspended	X	X		
24 Pointer Specification				
01 Pointer does not exist	X	X		
02 Pointer type invalid	X	X		
03 Pointer addressing invalid object	X	X		
2A Program Creation				
06 Invalid operand type	X	X		
07 Invalid operand attribute	X	X		
08 Invalid operand value range	X	X		
0C Invalid operand ODT reference	X	X		
0D Reserved bits are not zero	X	X		
38 Template Specification				
01 Template value invalid		X		

MODIFY DATA SPACE INDEX ATTRIBUTES (MODDSIA)

Op Code (Hex)	Operand 1	Operand 2
047A	Data space index	Data space index modification template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction modifies the attributes of a data space index that already exists. The data space index modification template identified by operand 2 provides the information needed to modify the data space index.

The format of the data space index modification template is described as follows:

• Modification indicators	Char(4)
– Reserved (binary 0)	Bits 0-1
– Delayed maintenance	Bit 2
0 = No delayed maintenance change	
1 = Delayed maintenance attribute change	
– Reserved (binary 0)	Bit 3
– Data space index force option	Bit 4
0 = No data space index force option change	
1 = Data space index force option change	
– Duplicate key rules	Bit 5
0 = No change in duplicate key rule	
1 = Change in duplicate key rule	
– Unit ID	Bit 6
0 = No change in unit ID	
1 = Change unit ID	
– Reserved (binary 0)	Bits 7-31
• Index attributes	Char(2)
– Reserved (binary 0)	Bits 0-10
– Delayed maintenance	Bit 11
0 = Change to immediate maintenance mode	
1 = Change to delayed maintenance mode	
– Reserved (binary 0)	Bit 12
– Data space index force option	Bit 13
0 = Normal data space index force	
1 = Force all data space index changes	
– Duplicate key rules	Bits 14-15
00 = Unique keys	
10 = FIFO keys	
01 = Reserved	
11 = Reserved	
• Unit ID option value	Char(1)
• Reserved (binary 0)	Char(9)

The data space index modification template must be aligned on a multiple of 16 bytes.

The modification indicators indicate which of the attributes/parameters of the data space index are to be modified by the instruction. A value of binary one indicates the listed attribute is to be changed. If the indicator contains a binary zero, the attribute is not to be changed and the corresponding field or fields in the modification template are ignored.

Only one attribute can be changed at a time by this instruction. If the modification indicators indicate change in more than one index attribute, the template value invalid data space exception will be signaled.

If the delayed maintenance indicator has a value of binary 1, the data space index attributes indicated by operand 1 will be modified as indicated in the delayed maintenance attribute field. An object not eligible for operation exception will be signaled if the data space index is a unique keyed data space index and the delayed maintenance attribute change field contains a binary one. If index key type indicator has a value of binary one, the duplicate key rules field in the index will be modified as indicated by duplicate key rules field. An object not eligible for operation exception will be signaled if the index is valid and change in key rule attributes other than from unique to FIFO specified. In case data space index is invalid, the change in either direction is permissible except that an object not eligible for operation exception will be signaled. If index is delayed maintenance and duplicate key rule attributes contain binary zeros. No change from FIFO to LIFO or LIFO to LIFO is permissible.

If the data space index force option indicator has a value of binary one, the attributes of the data space index indicated by operand 1 will be modified as indicated in the data space index force option attribute field. An object not eligible for operation will be signaled if this is a temporary index.

If the unit ID indicator has a value of binary one, the unit ID of the data space index is changed to the value indicated by unit ID option value field. The unit ID value must be valid for the system.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Object management
  - Data space index

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Implicit Locks
  - Data space index being modified is implicitly locked LEAR for the duration of this instruction

Events

0002 Authorization
 0101 Authorization violation

000C Machine resource
 0201 Machine auxiliary storage threshold exceeded

000D Machine status
 0101 Machine check

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage Encountered			
04 System object damage	X	X	
44 Partial system object damage	X	X	
1A Lock State			
01 Invalid lock state	X	X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded	X		
06 Machine lock limit exceeded	X		
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
04 Object not eligible for operation	X	X	

Exceptions

Exception	Operands		Other
	1	2	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	
2C Program Execution			
06 Instruction cancellation			X
07 Instruction termination			X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
02 Process storage limit exceeded			X
38 Template Specification			
01 Template value invalid		X	

RELEASE DATA SPACE ENTRIES (RLSDSEN)

Op Code (Hex)	Operand 1	Operand 2
048E	Cursor	Release options

Operand 1: System pointer.

Operand 2: Character(1) scalar (fixed-length).

Description: The instruction releases either the first data space entry, the last data space entry, or all data space entries currently locked to the process through the cursor. Data space entries are locked to a process, one at a time, through applications of the Set Cursor instruction specifying a lock entry option. They are unlocked during the updating or the deleting of the entries through the Update Data Space Entry or Delete Data Space Entry instructions.

If they are to be unlocked without any change having been made, the Release Data Space Entries instruction is used. This instruction specifies the cursor (which must be activated to this process) through which they were locked.

If the release option field has a value of hex 00, all data space entries currently identified by the locked entry queue for this cursor are removed from the queue and added to the locked entry list for the controlling commit block. If the cursor is not under commitment control, the data space entries identified by the locked entry queue for this cursor are removed from the queue, unlocked, and the respective LSUP implicit lock is removed from the data space (if the activation options indicate to do so).

If the option field has a value of hex 01, only the first entry identified by the locked entry queue for this cursor (the entry that has been locked the longest) is unlocked, and added to the locked entry list for the controlling commit block. If the cursor is not under commitment control, the first data space entry identified by the locked entry queue for this cursor is removed from the queue, unlocked, and the implicit LSUP lock is removed from the data space (if the activation options indicate to do so).

If the option field has a value of hex 02, only the last entry identified by the locked entry queue for this cursor (the entry that was last locked) is unlocked and added to the locked entry list for the controlling commit block. If the cursor is not under commitment control, the last data space identified by the locked entry queue for this cursor is removed from the queue, unlocked, and the implicit LSUP lock is removed from the data space (if the activation options indicate to do so).

If the release option field has a value of hex 10, this instruction causes all data space entries currently identified by the locked entry queue for this cursor to be removed from the queue, unlocked (if not previously locked to the commit block), and the respective LSUP implicit lock to be removed from the data space (if the cursor activation options indicate to do so). If the entry was previously locked through the controlling commit block, the entry lock is again placed in the locked entry list of the commit block.

If the option field has a value of hex 11, this instruction causes only the first entry identified by the locked entry queue for this cursor (the entry that has been locked the longest) to be removed from the queue, unlocked (if not previously locked to the commit block), and the respective LSUP implicit lock to be removed from the data space (if the cursor activation options indicate to do so). If the entry was previously locked through the controlling commit block, the entry lock is again placed in the locked entry list of the commit block.

If the option field has a value of hex 12, this instruction causes only the last entry identified by the locked entry queue for this cursor (the entry that was locked last) to be removed from the queue, unlocked (if not previously locked to the commit block), and the respective LSUP implicit lock to be removed from the data space (if the cursor activation options indicate to do so). If the entry was previously locked through the controlling commit block, the entry lock is again placed in the locked entry list of the commit block.

Options indicated by a value of hex 10, hex 11, and hex 12 perform the indicated functions regardless of whether or not the cursor is under commitment control. No exception is signaled if there are no entries currently in the cursor's locked entry queue.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Implicit locks
  - Implicit LSUP locks are removed from the affected data spaces

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X		
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
05 Object not available to process	X		
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
03 Scalar value invalid		X	

RETRIEVE DATA SPACE ENTRY (RETDSEN)

Op Code (Hex)	Operand 1	Operand 2
048A	Interface buffer	Cursor

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The data space entry addressed by the most recent cursor positioning operation is retrieved. Addressability to the entry is provided by the operand 2 cursor, which must be activated to the current process. The fields are presented in the interface buffer, identified by the operand 1 space pointer, in the format and sequence established by the output mapping template specifications provided in the Create Cursor instruction. The entry retrieved is the entry addressed by the most recent successful cursor positioning operation using the operand 2 cursor and not necessarily the entry at the head of the locked entry queue associated with this cursor.

If a key was used directly or indirectly by the cursor (that is, the cursor is over a data space index) and that key has changed since the cursor was positioned, an exception is signaled, and the entry is not retrieved. If the Set Cursor instruction locked the entry and no intervening release, update, or delete has been performed against this cursor, no such key changes are possible.

If the retrieve operation intent was not indicated when the Activate Cursor instruction was issued, an invalid data base operation exception is signaled.

If the cursor position is the result of performing group-by processing, an object not eligible for operation exception is signaled.

Authorization Required

- Retrieve
  - Data space affected
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Object authorization violation

0008 Data space index
  0301 Data space index invalidated

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation			X
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
12 Data Base Management			
01 Conversion mapping error			X
03 Cursor not set		X	
06 Data space entry not found			X
07 Data space index invalid			X
17 Key changed since set cursor			X
25 Invalid data base operation	X		
30 Specified data space entry rejected			X
32 Join value changed			X
34 Non-user exit selection routine failure			X
36 No mapping code specified	X		
38 Derived field operation error			X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	X
03 Object suspended	X	X	
05 Object not available to process		X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

RETRIEVE SEQUENTIAL DATA SPACE ENTRIES (RETSDSE)

Op Code (Hex)	**Operand 1**	**Operand 2**	**Operand 3**
048B	Interface buffer	Cursor	Option template

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Space pointer.

Description: This instruction retrieves multiple sequential data space entries or group-by results based on the current position of the cursor identified in operand 2 and places them, in sequence, in the space identified in operand 1 according to the field mapping specifications defined at the creation of the cursor. The cursor is repositioned during the operation. The operand 2 cursor is modified to address the next sequential entry referenced through the underlying data space(s) or data space index. The data space entry addressed by the cursor or the results of processing a group-by function are then placed in the interface buffer (operand 1) in the manner described by the output mapping template specifications or group-by template defined during the Create Cursor instruction. These operations are repeated until the number of entries requested in the operand 3 option template have been placed in the interface buffer or the number of entries designated by the candidate entries to process in the extended template have been processed. The entries are in the interface buffer in the exact order that they were retrieved. Each entry in the interface buffer has up to four separate pieces of data that consist of:

- The data space number followed by the ordinal entry number of the data space entry
- The key of the data space entry (optional)
- The default values feedback (optional)
- The mapped data space entry

The pieces for retrieving group-by results are:

- The group-by results.
- The count of candidate data space entries processed while deriving the group.

- The default values feedback (optional).

- The data space number (optional).

At the completion of the instruction, the cursor addresses the last data space entry retrieved by the operation (except in key mapping exception conditions).

The format of the option template referenced by operand 3 is as follows:

• Control options	Char(2)
– Reserved (binary 0)	Bit 0
– Materialize data space and	Bit 1
– Materialize key	Bit 2
– Materialize default values feedback	Bit 3
– Extended template	Bit 4
– Process group-by	Bit 5
– Key field count specified	Bit 6
– Key byte count specified	Bit 7
– Retrieval direction ordinal number	Bit 8
– Reserved (binary 0's)	Bits 9-15
• Buffer entry length	Bin(2)
• Data space and ordinal entry position	Bin(2)
• Key position	Bin(2)
• Data space entry position	Bin(2)
• Number of entries requested	Bin(2)
• Operation status	Char(2)
– Key not returned indicator	Bit 0*
– Exception encountered indicator	Bit 1*
– Cursor not positioned indicator	Bit 2
– Candidate entries to process exceeded	Bit 3
– Reserved (binary 0)	Bits 4-15
• Interface buffer position	Bin(2)
• Reserved (binary 0)	Char(2)
• Extended template area	Char(66)
– Data space key format	Bin(2)
– Key field count	Bin(2)
– Key byte count	Bin(2)
– Candidate entries to process	Bin(4)
– Candidate entries processed	Bin(4)
– Candidate entries on primary Data space processed	Bin(4)
– Widened buffer entry length	Bin(4)
– Widened data space and entry offset	Bin(4)
– Widened key offset	Bin(4)
– Widended data space entry offset	Bin(4)
– Entries processed offset	Bin(4)
– Default values feedback offset	Bin(4)
– Reserved (binary 0)	Char(24)

A value of binary 1 in the materialize data space and ordinal number field results in the return of the data space number and ordinal entry number in the interface buffer for each entry. The position of these fields in the buffer entry is defined by the data space and ordinal entry position field. When the extended template is specified, the position of the fields in the buffer entry is defined by the widened data space and ordinal entry offset field. The data space and ordinal entry offset must contain binary zeros. A value of binary 0 in the materialize data space and ordinal number fields results in the data space and ordinal entry number not being placed in the interface buffer entry and the widened data space and ordinal entry offset field is ignored.

A materialize key value of binary 1 indicates that the key of each entry retrieved should be returned in the interface buffer. The position of the key in the interface buffer is defined by the key offset field. When the extended template is specified, the position of the key in the interface buffer entry is defined by the widened key offset field, and the key offset field must contain binary zeros. A value of binary 0 in the materialize key field will result in the key not being mapped into the interface buffer and the widened key offset field being ignored.

A materialize default values feedback value of binary 1 indicates that information regarding supplying default values when joining is performed should be returned in the interface buffer. The position in the interface is defined by the default values feedback offset field. A value of binary 0 in the materialize default values feedback field will result in the information not being returned. This field is ignored when the cursor is non-join.

The extended template control option indicates the use of the extended area of the input template.

The process group-by control attribute indicates to perform group-by functions through this cursor versus the retrieval of the data space entries viewed through the cursor.

Group-by processing will process the next/previous groups from the position the cursor currently contains.

When the cursor is directly over a data space, group-by processing cannot be specified if the cursor is set. The fields key field count specified, key byte count specified, key field count, key byte count, and DS key format fields are ignored when the cursor is directly over a data space.

When the cursor is over a primary data space index, the group is defined by the key byte count or key field count. If the cursor is not positioned, the group begins with the first or last key in the index (depending on retrieval direction). If the cursor is positioned, it must have been positioned to an entry with a corresponding key in the index. The group begins with the adjacent unique key (as defined by the key byte count or key field count). Therefore, if positioned in the midst of a group, the remaining entries in that group are bypassed and group-by processing begins with the first entry in the adjacent group.

The data space key format field is required when specifying group-by processing and key field count or key byte count is specified. This field indicates which set of key field attributes are to be used in determining the internal key length from the provided key field count or key byte count. The data space key format field must contain a number that corresponds to the position of the data space in the data space pointer list for the operand 2 cursor which was provided when the cursor was activated. This value must equal 1 when using a join cursor.

The key field count control option indicates the field key field count is used in generic key processing. The extended template must be specified when this option is specified.

The key byte count specified control option indicates the field key byte count is used in generic key processing. It is invalid to specify both key byte count and key field count. The extended template must be specified when key byte count is specified.

The retrieval direction field specifies the direction from the current position of the cursor retrieval should take. A value of binary zero indicates a forward (next) direction, a value of binary one indicates a reverse (previous) direction.

The buffer entry length field defines the length each entry occupies in the interface buffer. The format of each entry in the buffer is defined by the option template. The start of the first buffer entry is at offset zero in the interface buffer. Each successive entry in the buffer begins on the byte defined by the buffer template length. For example, if the buffer entry length is 200, the second entry starts in position 201 of the interface buffer, the third entry in position 401, and so on. The buffer entry length field must be greater than 0.

The widened buffer entry length field is identical to the buffer entry length field except that the widened buffer entry length is a Bin(4), buffer entry length is a Bin(2). When the extended template is specified, the widened buffer entry length field is used, the buffer entry length field must be binary zero. For the remainder of this architecture, buffer entry length refers to either field, whichever is specified.

Each entry is created as follows:

- If the materialize data space and ordinal number field is binary 1, the 2-byte data space number is placed into the buffer entry beginning in the position designated by the widened data space and ordinal entry offset field. This field, if specified, must contain a value greater than or equal to 0 and less than the buffer entry length. The 4-byte ordinal entry number of the data space entry is returned immediately following the data space number in the interface buffer.

- If the materialize key field has a value of binary 1, the composite key for the data space entry is returned in the buffer entry beginning in the position designated by the widened key offset field. This field, if specified, must contain a value greater than or equal to 0 and less than the buffer entry length.

- If the materialize default values feedback field has a value of binary one, the default values feedback field is returned in the buffer entry beginning in the position designated by the default values feedback offset field. This field, if specified, must contain a non-negative value and be less than the buffer length. The default value feedback field is a Char(4) field.

- The data space entry is then presented, as defined by the output field mapping specifications defined at the creation of the cursor, beginning in the first position of the buffer entry defined by the data space entry offset field or widened data space entry offset field. This field must be provided and must have a non-negative value and be less than the buffer entry length. When the extended template is specified, the widened data space entry offset field is used and the data space entry offset field must contain binary zeros.

Each entry is created in the following manner when processing group-by:

- The group-by results is presented, as defined by the group-by output mapping template at cursor creation, beginning in the position designated by the widened data space entry offset field.

- The count of candidate data space entries processed while deriving the group results, regardless of whether they belonged to the group, is returned in the buffer entry beginning in the position designated by entries processed offset. This field must contain a non-negative value and be less than the buffer entry length. This count is a Bin(4) field. This count remains at 2,147,483,647 once this tally is reached for a group.

- Default values feedback field, if applicable.

Note: If the remaining area in the buffer entry is not large enough for the entry or either of the return fields to be placed in the interface buffer, the entry is mapped into the position immediately following the buffer entry. The data space entry may be placed over the area in which the return fields were specified.

The number of entries requested field contains the number of entries that are to be presented in the interface buffer. This field must contain a positive value that is greater than 0.

A key not returned value of binary 1 indicates that even though a return of the key was requested, the system was unable to provide the key in every returned entry. See the Set Cursor instruction for the conditions which can cause this field to be set to binary 1.

If the cursor positioning status bit is returned with a value of binary 1, Retrieve Sequential was in the process of performing the positioning of the cursor. If the cursor positioning status bit is returned with a value of binary 0, Retrieve Sequential was in the process of performing the retrieval of data associated with the position established, but that entry has not been returned in the interface buffer and the interface buffer position field does not reflect the data space entry to which the cursor is not set. At the completion of a successful Retrieve Sequential, this bit no longer may be used to determine the positioning status (it is left in a retrieval status).

As the secondary positions of a join cursor are processed, there are similar positioning and retrieving phases being performed for each secondary position. The cursor positioning status bit is modified to reflect these phases in the same manner as positioning the primary position of a join cursor.

When performing group-by operation, the cursor not positioned indicator is set to binary 1 while positioning to the first entry to be processed for the group. For a join cursor, this bit field contains a binary zero while establishing the join secondary positions. After successfully positioning, the cursor not positioned indicator is set to binary 0 and remains that way until the group is successfully processed (that is, group result is rejected or returned in user interface).

When the cursor indicates default values are to be assigned, each data space in the cursor data space list which supplied a default value will be repeated in the default values feedback field. This field will contain a 32-bit array with each bit corresponding to a position of a data space in the cursor's data space list. The default values assigned will be taken from that data space's default entry. When a default value feedback field is set to binary 1. The position in the interface buffer to return the default values feedbakc is given by the default values feedback offset field.

An exception encountered return value of binary 1 indicates an exception was encountered while implicitly setting the cursor or retrieving the next sequential data space entry. The following exceptions, listed with the resultant cursor positioning, results in the indicator being set to binary 1:

- 1201–Conversion mapping error

  The data space and ordinal entry number (of the primary data space if a join cursor) if requested and the cursor not positioned indicator contains a binary 0, contain values which identify the data space entry the cursor is positioned to, but the interface does not reflect this entry nor the group-by results.

- 1202–Key mapping error

  The cursor is positioned to the last retrieved data space entry, or joined entry. The data space and ordinal entry number (of the primary data space if a join cursor), if requested, will have been created in the buffer entry, but the entry and the default values feedback will not be placed in the buffer entry. Group-by fields and results will not reflect the attempted positioning.

- 120A–End of path

  The number of entries retrieved field designates the number of valid entries retrieved.

- 1234–Non-user exit selection routine failure

  The cursor is positioned to the entry which caused the selection routine failure. Nothing has been placed into the user buffer for this entry.

All other exceptions result in a value of binary 1 in the exception encountered field. The number of entries retrieved field may not be updated.

The number of entries retrieved field contains the ordinal entry number of the last entry mapped into the interface buffer. Certain exception conditions (previously defined) can result in slightly different settings of this field. A value of 1 would indicate the first entry and so on.

The key field count designates the number of key fields to be used as comparison in the composite key used as the group-by value. The definition of this field is identical to the definition of this field in Set Cursor.

The key byte count designates the number of bytes of the logical key structure to be used in comparison on the group-by key. The definition of this field and the restrictions on its use is identical to the definition and restrictions of this field in Set Cursor.

When the key field count specified and key byte count specified control options contain a value of binary zero, the entire index is considered the group. Note that when this option is specified, even though each key is considered part of the group, the group begins with the adjacent entry the cursor is currently positioned to/

When Retrieve Sequential requires the use of a key associated with the current cursor position, and the cursor was positioned without a key, an invalid rule option exception is signaled. This includes normal retrieval processing via the index and group-by processing when the current key value is required.

Candidate entries include entries which may not be reflected in the user buffer (may have been omitted via selection criteria). Entries which may have been omitted via key selection criteria (refer to Create Cursor) may not be included in the candidate count.

The candidate entries to process designates a limit to the number of entries to process before returning to the MI user (includes entries processed but not reflected in the user interface buffer). This field is cumulative and designates the limit on number of entries to process for all groups when in group-by processing. A value of zero in this field designates that there is no limit.

Upon exceeding the candidate entries to process value, the instruction terminates and returns normally with the candidate entries to process exceeded bit set to binary 1. The cursor position and interface buffer results will not reflect the attempt to position to the entry exceeding the limit.

The candidate entries processed return field contains the count of candidate entries processed; includes both returned entries and entries not returned.

The candidate entries processed on primary data space return field contains the count of candidate entries processed on the primary data space of a join cursor. It includes both returned entries and entries not returned. For a non-join cursor, this field contents are the same as the candidate entries processed field.

If the cursor was not set (that is, is not addressing any data space entry) prior to the execution of this instruction, it retrieves the first and last entry(s) or entries in the data space or data space(s) index indicated by the cursor.

This instruction does not lock any data space entries. However, any entries previously locked to this cursor remain locked to the cursor.

If the indicated cursor is under commitment control, an operation not valid under commitment control exception is signaled and the operation is terminated.

If the retrieve operation intent was not indicated when the Activate Cursor instruction was issued, an invalid data base operation exception is signaled.

All deleted entries encountered are skipped.

Authorization Required

- Retrieve
  - Data space affected
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

0008 Data space index
    0301 Data space index invalidated

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation				X
10 Damage Encountered				
04 System object damage state				X
44 Partial system object damage		X		X
12 Data Base Management				
01 Conversion mapping error				X
02 Key mapping error				X
07 Data space index invalid				X
08 Incomplete key description				X
0A End of path				X
19 Invalid rule option		X		
25 Invalid data base operation	X			
34 Non-user exit selection routine failure				X
36 No mapping code specfied	X			
38 Derived field operation error				X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	X
03 Object suspended	X	X	X	
04 Object not eligible for operation		X		
05 Object not available to process		X		
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X

Exceptions

Exception	Operands 1	2	3	Other
2C Program Execution				
06 Instruction cancellation				X
07 Instruction termination				X
2E Resource Control Unit				
01 User profile storage limit exceeded				X
38 Template Specification				
01 Template value invalid			X	
3E Commitment Control				
10 Operation not valid under commitment control				X

SET CURSOR (SETCR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
048C	Cursor	Option template	Returned key	Requested key

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: Character variable scalar or null.

Operand 4: Character scalar or null.

Description: This instruction is used to establish addressability through the operand 1 cursor to a particular entry within a data space. The cursor must be activated to this process. The option template identified by operand 2 governs the setting of the cursor.

This instruction causes the cursor to address or not to address an entry in a data space according to the search arguments given in operand 2 and operand 4. Addressability to the desired entry is stored in the cursor identified by the operand 1 system pointer. The key of the desired entry is optionally returned in operand 3.

The option template has the following format:

• Length of option template	Bin(4)*
• Rule option	Char(1)
• Search attributes	Char(1)
• Control attributes	Char(1)
• Key length description	Char(1)
– Key length type	Bit 0
0 = Length specified as count of key fields	
1 = Length specified as key byte count	
– Key length	Bits 1-7
• Relative/ordinal number	Bin(4)
• Data space key format	Bin(2)
• Data space number (return value)	Bin(2)
• Ordinal entry number (return value)	Bin(4)

• Number of data spaces in restricted search list (maximum of 32 entries)	Bin(2)
• Data space to be included in a restricted search list (0 to 32)	Bin(2)
• Extended option list area	
– Extended functions	Char(2)
Candidate entries to process provided	Bit 0
Candidate limit exceeded (return field)	Bit 1
Reserved (binary 0)	Bit 2-15
– Candidate entries to process	Bin(4)
– Candidate entries processed (return value)	Bin(4)
– Candidate entries processed primary data space	Bin(4)
– Default values feedback (return value)	Char(4)
– Reserved (binary 0)	Char(28)

Note: The value of the entry shown here with an asterisk (*) is ignored by this instruction.

The rule option indicates the type of search to be done. The type of search that can be done through the cursor depends on whether the cursor is addressing data spaces through a data space index or it is addressing data spaces directly. The following table indicates the allowable values of the rule option and when they can be used.

		Cursor Over:	
		Data	**Data**
	Value	**Space**	**Space(s)**
Rule Option	**(Hex)**	**Index**	**Directly**
First	01	X	X
Last	02	X	X
Next	03	X	X
Previous	04	X	X
Next unique	05	X	
Previous unique	06	X	
Relative	07	X	X
Ordinal	08	X	X
Key–before	09	X	
Key–equal or before	0A	X	
Key–equal	0B	X	
Key–equal or after	0C	X	
Key–after	0D	X	
Next equal	0E	X	
Previous equal	0F	X	
Same	21	X	X

For a cursor activated over a data space index, the meaning of each rule option is as follows:

- First–The cursor is set to address the entry associated with the first key in the data space index.

- Last–The cursor is set to address the entry associated with the last key in the data space index.

- Next–The cursor is set to address the entry associated with the next key in the data space index after the key currently referenced by the cursor.

- Previous–The cursor is set to address the entry associated with the previous key in the data space index before the key currently referenced by the cursor.

- Next unique–The cursor is set to address the entry in the data space index after the current entry that has a key value that is different (as qualified by the key length description) than the key value of the current entry.

- Previous unique–The cursor is set to address the entry in the data space index before the current entry that has a key value that is different (as qualified by the key length description) than the key value of the current entry.

- Relative–The cursor is set to address an entry in the same data space as the current entry by adding the specified relative/ordinal number to the ordinal number of the current entry. If a key for the entry exists in the data space index, the cursor is set so that a subsequent rule option of next, previous, next unique, or previous unique can be used.

- Ordinal–The cursor is set to address the entry in the data space indicated by the first field entry in the template area reserved for the restricted data space search list and having the ordinal number specified by the relative/ordinal number field. If a key for the entry exists in the data space index, the cursor is set such that a subsequent rule option of next, previous, next unique, or previous unique can be used. The number of data spaces in restricted search list field must have a value greater than zero. When the cursor is type join, the dat space referenced by the restricted search list must be the primary data space.

- Key operations–In the following key operations, the key length description indicates the length of the key value in the operand 4 argument. The key length type specifies whether key length indicates a count of fields or a count of bytes comprising the key value. The fields are expected to be ordered as specified in the key format (see the Create Data Space Index instruction, earlier in this chapter) for the data space specified in the data space key format field. The fields must also have the field attributes as specified in the output mapping template in the Create Cursor instruction. If a key field is referenced more than once in the output mapping template, the first occurrence of the field in the template determines the attributes of the field. No fork characters are in the operand 4 argument, only data fields.

  The cursor is set to address the entry associated with the key found in the index as follows:
  - *Key—before* finds the first key in the data space index before the specified key value.
  - *Key—equal or before* finds the first key in the data space index before the specified key value only when no key in the data space index matches the specified key value.
  - *Key—equal* finds the first key in the data space index that matches the specified key value.
  - *Key—equal or after* finds the first key in the data space index after the specified key value only when no key in the data space index matches the specified key value.
  - *Key—after* finds the key in the data space index after the specified key value.

- Next equal–The cursor is set to address the entry in the data space index after the current entry that has a key value that is the same (qualified by the key length description) as the key value of the current entry. If the cursor is not positioned to a key value, an invalid rule option exception is signaled. If there is no data space entry after the current entry with a key value that matches the key value of the current entry (qualified by the key length description), an entry not found exception is signaled.

- Previous equal–The cursor is set to address the entry in the data space index before the current entry that has a key value that is the same (qualified by the key length description) as the key value of the current entry. If the cursor is not positioned to a key value, an invalid rule option exception is signaled. If there is no data space entry before the current entry with a key value that matches the key value of the current entry (qualified by the key length description), an entry not found exception is signaled.

- Same–The current cursor position is not changed. If the cursor is not currently set, a cursor not set exception is signaled. If the data space entry (the address of the data space entry that resulted from a previous Set Cursor or Retrieve Sequential Data Space Entries instruction) was through a rule option that used a data space index for positioning but that data space entry is not addressed by that data space index, an entry not found exception is signaled. If the data space entry no longer has the same key value in that data space index, a key value changed exception is signaled.

  If the data space entry (the address of the cursor that resulted from a previous Set Cursor or Retrieve Sequential Data Space Entries instruction) was through a rule option that did not result in the use of a data space index for positioning and that data space entry is deleted and its significance attribute is binary 0, an entry not found exception is signaled.

  When using a rule option of same with a join cursor, if any join field used in positioning the join cursor has changed since positioning, a join value changed exception is signaled. If any of the data space entries participating in the join no longer have the exact same key value in the associated data space index, a key value changed exception is signaled. If any of the data space entries participating in the join are no longer addressed by the associated data space index, an entry not found exception is signaled.

A Set Cursor instruction with the rule option of same, does not change the status (when a data space index is used to position the cursor) of the previous successful Set Cursor or Retrieve Sequential Data Space Entries instruction.

The meaning of the valid rule options are as follows (for multiple data spaces, the search continues into the next data space, when needed, except when restrict to requested data space is specified):

- First–The cursor is set to address the first undeleted entry.
- Last–The cursor is set to address the last undeleted entry.
- Next–The cursor is set to address the first undeleted entry following the entry currently addressed by the cursor.
- Previous–The cursor is set to address the first undeleted entry preceding the entry currently addressed by the cursor.
- Relative–The cursor is set to address an entry in the same data space as the current entry by adding the specified relative/ordinal number to the ordinal number of the current entry. A relative/ordinal number causing the search to exceed the bounds of the data space will result in an end of path exception.
- Ordinal–The cursor is set to the entry in the data space indicated by the first field entry in the restricted data space search list and having the ordinal number specified by the relative/ordinal number field. When the cursor is type join, the data space referenced by the restricted search list must be the primary data space. The number of data spaces in restricted search list field must be greater than zero.
- Same–The current cursor position is not changed. If the cursor is not currently set, a cursor not set exception is signaled. If the entry the cursor is addressing as the result of the previous successful Set Cursor or Retrieve Sequential Data Space Entries instruction is deleted and the deleted entry significance attribute contains a binary zero, an entry not found exception is signaled.

When positioning a join cursor, the returned key, data space number and ordinal number return fields, entry deleted, and key return bit reflect only the position of the primary data space participating in the join.

Secondary data space positions which contributed default values will be set to a default position (that is, the values for that data space originated from the default entry).

Priming a join cursor position means to determine a position for each secondary data space defined under the cursor. Priming is performed without regard for the current position held for this data space but is based on the results of the position of the previous data space as defined by the join order.

All of the rule options except next, previous, next equal, previous equal, and same refer to positioning of the primary. The secondary positions are then primed as determined by the position of the primary. The possible positions in secondaries (first/last of duplicate join values) are determined by the implied direction of the rule option. For rule options of first, next unique, relative (zero or positive), ordinal, key equal, key equal or after key after, the first possible secondary data space positions are returned. For rule options of last, previous unique, key before, relative (negative), the last of possible secondary data space positions are returned.

For rule option of key equal or before the last or first of possible secondary data space positions are returned depending on whether the key is a before or an equal. When the key is a before or an equal. When the key is a before key, the last of possible secondary data space positions are returned. When the key is an equal key, the first of possible secondary data space positions are returned.

When performing a next equal, previous equal, next or previous operation, and each data space associated with the join cursor has a position, the last data space in the defined join order is positioned to the adjacent joined entry. If a secondary does not have a position (null position), an adjacent search begins with the preceding data space in the join order and each secondary's null position is them primed. Likewise, when performing an adjacent search and the end of possible join value is reached for that data space, the adjacent search begins with the previous data space (as defined by the join order) and priming of subsequent secondary positions then follows.

When adjacent searches have reached the end of all data space under the join cursor, an end of path exception is signaled.

The search attributes are used to modify the normal operations of the searches indicated by the rule options. The search attributes fields and their functions are as follows:

Restricted to requested data spaces	Bit 0
Trailing fork characters	Bit 1
Deleted entry significance	Bit 2
Entry deleted return bit	Bit 3
Key return bit	Bit 4
Materialize key indicator	Bit 5
Positioning indicator	Bit 6
Extended option list	Bit 7

- Restricted to requested data spaces–A value of binary 1 causes the search indicated by the rule option to continue until an entry is found in a data space indicated by the restricted data space search list. This search attribute is ignored in the following situations:
  - Cursor over data spaces directly (no index) and rule option = next, previous
  - Rule option = ordinal, relative, same

  If the restricted to requested data spaces attribute is specified and an entry is not subsequently found, an end of path exception is signaled, unless rule = ordinal or key equal is used, for which an entry not found exception is signaled. If rule = relative, the resulting ordinal number is valid, and the designated entry has been deleted, then an entry not found exception is signaled. If the ordinal number is not valid, then an end of path exception is signaled.

  The search attribute restrict to requested data spaces may not be designated if the cursor is of type join. A template value invalid exception is signaled for this request.

- Trailing fork characters–This attribute is considered when the key length description field is used. A value of binary 0 indicates that fork characters following the last complete key field indicated by key length description are not to be used during the search. A value of binary 1 indicates that the fork characters immediately following the last complete key field indicated by the key length description field, should be used during the search.

- Deleted entry significance–Deleted entries are generally skipped in all searches except when the rule option is relative, ordinal, or same. They result in entry not found exceptions for relative, ordinal, or same because specific entry is referenced. When the cursor is directly over data spaces or when the rule option is relative or ordinal and the cursor is over a data space index, the deleted entry significance value of binary 1 allows the ordinal positions formerly occupied by deleted entries to be addressed. When this attribute is binary 1 and the search criterion leads to a deleted entry, the entry is not skipped. Instead, the cursor is set to address the deleted entry and the entry deleted return bit is set in the option list. A subsequent attempt to retrieve this entry results in an entry not found exception, but an update allows a new entry to be inserted into the space occupied by the deleted entry. For such an update, values for fields not in the input mapping template are supplied by the default values entry. When deleted entry significance is specified with a join cursor and the primary position is to a deleted data space entry, the subsequent secondary positions will be set to null (that is, no position established).

- The entry deleted return value is altered only when the deleted entry significance option is specified and a deleted entry is addressed. A value of binary 1 for deleted entry significance is invalid when the cursor is over a data space index and the rule option is not relative, ordinal, or same.

- A key return value of binary 1 indicates the key of the desired entry has been returned in operand 3. A value of binary 0 indicates the key was not returned for one or more of the reasons listed in the discussion of the return key.

- A materialize key indicator value of binary 1 indicates that the key of the desired entry should be returned in operand 3 upon successful completion of the Set Cursor instruction. If operand 3 is null or cannot contain the entire key, a scalar value invalid exception is signaled.

- A positioning indicator of binary 0 causes the cursor to be positioned at the entry indicated by the options list and the current position of the cursor. A position indicator of binary 1 leaves the position of the cursor unchanged. The return values of the operand 2 options list and the operand 3 key returned will be returned as if the positioning indicator was binary 0. Data space entry locks will be queued on the cursor's locked entry queue. The set cursor operation cannot be materialized by the Materialize Cursor Attributes instruction.

A value of binary one in the extended option list field indicated the extended area of the option list has been supplied and is to be processed.

The search attributes are meaningful only for the primary data space position of a join cursor.

The control attributtes refer only to locking and updating and are not meaningful with a join cursor.

The control attributes are options that control the status of the entry upon completion of the Set Cursor instruction. The control attributes field is as follows:

– Forced write option	Bit 0
– Lock entry check	Bit 1
– Lock entry return bit	Bit 2
– Locked entry queuing selector	Bit 3
0 = Locked as the last entry on the queue	
1 = Locked as the first entry on the queue	
– Data space entry lock option	Bits 4-5
00 = Shared use	
01 = Lock entry with no wait	
10 = Reserved	
11 = Lock entry with wait	
– Access state modifications	Bits 6-7
When entering lock wait for a data space entry	Bit 6
0 = Do not modify access state when entering wait	
1 = Modify access state when entering lock wait	
When leaving lock wait	Bit 7
0 = Do not modify access state when leaving lock wait	
1 = Modify access state when leaving lock wait	

Each of the control options is described as follows:

- Forced write option–If equal to binary 1, the data space entry, when updated or deleted, is forced to nonvolatile storage before the completion of the associated instruction.

- Lock entry check–This option works in conjunction with the queue entry that is specified in the locked entry queuing selector. If this option is specified and the data space entry that is addressed by this instruction is locked to the same entry on the cursor's locked queue as the entry that is specified in the locked entry queuing selector and a lock of the data space entry is requested, then the lock on the data space is not changed and the cursor is set to address the requested data space entry. If this option is not specified and an attempt is made to lock a data space entry that is already on this cursor's locked entry queue, a data space entry locked exception is signaled. A template value invalid exception is signaled if this option is selected and the data space entry lock option contains a value of binary 00.

- Locked entry return bit–This field contains a return value of binary 1 if the lock entry check option is selected and the resulting data space entry is already locked to this cursor at the last entry on the cursor's locked entry queue. This field contains a return value of binary 0 if the lock entry check option is selected and the resulting data space entry is not already locked to any cursor. If the lock entry check option is not specified, this field is not modified.

- Locked entry queuing selector–If equal to binary 0, the data space entry lock obtained by this Set Cursor instruction is placed on the last entry of the cursor's locked entry queue. If equal to binary 1, the data space entry lock obtained by this instruction is placed on the first entry of the cursor's locked entry queue.

- Shared use–This option does not cause the entry to be locked and does not check to see whether the entry is currently locked. This option allows an entry to be retrieved but not subsequently updated or deleted. When cursor is of type join, the data space entry lock option may only designate shared use. A template value invalid exception is signaled for the disallowed lock requests.

- Lock entry with no wait–This option allows a data space entry to be addressed so that it can subsequently be updated or deleted. This request causes the entry to be locked to the cursor. If the cursor is under commitment control and the entry is locked to the controlling commit block, the cursor again holds the lock. If the entry is locked to a cursor or some other commit block, the data space entry locked exception is signaled immediately.

- Lock entry with wait–If the entry is not presently locked, this option is the same as lock entry with no wait. If the entry is already locked, the requesting process is put in a wait state. The process waits either until the entry becomes available–in which case the request is honored–or for a prespecified amount of time (specified at the time the cursor was activated) elapses–in which case a data space entry locked exception is signaled. This exception is an indication of a potential deadlock.

  When an entry is locked, an implicit LSUP lock is applied to the data space if the cursor activation options indicate to do so. The data space entry must have the first data space entry lock for this data space, the lock must be implicitly applied at activation time, and the lock must have lower priority than LSUP.

- Access state modification–The access state modification attributes control the changing of the access state of the process access group for the executing process during the entering of or leaving a wait for a locked entry. The option has no effect if the process instruction wait access state control attribute specifies that access state modification is not allowed. If the process attribute value specifies that access state modification is allowed and the option is modify access state, the process access group defined for the process has its access state modified.

A set cursor operation causes most of the results of the previous set cursor operation to be negated except when the positioning indicator is binary 1. However, it is possible to accumulate locks on data space entries for purposes of updating or deleting multiple entries. This is accomplished by issuing repetitive Set Cursor instructions with the lock option set to lock entry. Each successive Set Cursor instruction causes an additional entry to be locked and addressability to the entry to be put in a FIFO (first in, first out) locked entry queue associated with the cursor.

Each combination of a Set Cursor instruction (with the lock option specified) and a Retrieve Data Space Entry instruction causes another entry to be locked (Set Cursor instruction). The address of the entry is placed in the FIFO queue (Set Cursor instruction), and the retrieved entry is placed in the user's interface buffer (Retrieve Data Space Entry instruction). The FIFO queue identifies all entries that are locked by the cursor.

The entries can later be modified and unlocked from the FIFO queue in one of the following manners:

- An Update Data Space Entry or Delete Data Space Entry instruction is issued. Either of these instructions causes the first entry referenced by the queue to be removed from the queue (either updated or deleted) and unlocked. If the cursor is under commitment control, the lock is transferred to the controlling commit block and the data space entry remains locked through the commit block until all uncommitted changes are either committed or decommitted.

- A Release Data Space Entries instruction is issued. This instruction causes one or all of the entries in the queue to be removed from the queue and unlocked (without modification). If the option is to release only one entry, then this instruction causes the first entry referenced by the queue to be removed from the queue and unlocked. If the cursor is under commitment control, the lock may optionally be transferred to the controlling commit block and the data space entry remains locked through the commit block until all uncommitted changes are either committed or decommitted.

- The cursor is de-activated.

If no entries are addressable by the FIFO queue when an Update Data Space Entry or Delete Data Space Entry instruction is issued, an exception is signaled.

Intervening Insert Data Space Entry instructions have no effect on the FIFO queue or the positioning of the cursor.

The key length description designates the length of the key value assumed to be in the operand 4 argument to be used for searching the data space index.

The key length value only includes data fields supplied by the user for the Set Cursor instruction and does not include fork characters. The key length type specifies whether key length designates a count of data fields specified at data space index creation time or a count of bytes available to use as the search key value. If key length specifies a partial key that is less than all data fields combined, a generic search is performed. If key length specifies a larger key value than all actual combined data fields contained in the key for the designated data space, the key value length used is determined by the actual number of key fields specified at create data space index time. When key length using a byte count key length is specified, a generic search using a partial key field is valid only if direct key mapping exists between the logical and physical views of all key fields. A generic search on partial binary or floating point data fields is not valid and will result in a key mapping error exception. If key length specifies a partial key field and a key value is assumed in the operand 4 argument, the entire key data field is assumed to be addressable. A key length of zero is legal only if a fork character is defined as the first element of the key. If key length is zero, operand 4 is ignored. The key length description is required when the rule option is any of the key operations or next unique or previous unique or next equal or previous equal.

The relative/ordinal number is a positive or negative integer scalar that is used in conjunction with the relative or ordinal rule option. If the rule option is relative, the number is a positive or negative number indicating a relative positioning forward or backward in the data space from the currently addressed entry (including deleted entries). If the rule is ordinal, the number is the absolute position in the data space identified by the first entry in the restricted data space search list. Ordinal numbers are greater than or equal to 1. A negative or 0 ordinal number causes an end of path exception to be signaled. A relative number causing the search to exceed the bounds of the data space results in an end of path exception.

The data space key format field is required when the cursor is over a data space index and the rule option is any of the key operations. The data space key format indicates the variety of mapping and set of fork characters to be used in building the internal key from the provided key fields. The data space key format field must contain a number that corresponds with the position of an activated data space in the data space pointer list for the operand 1 cursor which was supplied (or defaulted to create cursor data space list) when the cursor was activated. The data space key format must designate the primary data space when positioning a join cursor.

The data space number field is a feedback area in the option template for the Set Cursor instruction. The value returned identifies the data space in which the data space entry to which the cursor has been positioned resides. This number corresponds with the position of the system pointer within the data space pointer list for the Create Cursor instruction.

The ordinal entry number field is a feedback area in the option template for the Set Cursor instruction. When a Set Cursor instruction operation is completed, the ordinal entry number of the data space entry currently being addressed is returned in the option template. The ordinal entry number of a data space entry is not affected by the Delete Data Space Entry instruction, and, therefore, addressability by ordinal number is also not affected by the Delete Data Space Entry instruction.

The number of data spaces in the restricted search list field identifies the number of data spaces in the restricted search list. Only entries from these data spaces will be used in the attempt to satisfy the search criteria. Each restricted search list field contains a number that corresponds with the position of the data space in the data space pointer list for this cursor. The ordering of entries in the restricted search list is not important, and duplicate entries will be eliminated by the Set Cursor instruction. The Materialize Cursor Attributes instruction will always return an ordered list of restricted data space entries without any duplicates.

Upon successful completion of the Set Cursor instruction with the cursor activated over an index, operand 3, if not null, will contain the composite key for the data space entry whose addressability is set in the cursor at the completion of this instruction; that is, the key for the entry indicated by the data space number and ordinal entry number returned in the feedback area of the option template. The key is returned only if (1) the cursor is activated over a data space index, (2) every key field was specified in the cursor output mapping template for that data space, (3) retrieve authority for that data space is satisfied, (4) the entry is not deleted, and (5) the entry has not been selected out of the data space index. The fields within the key are ordered as specified in the data space key specification for that data space in the Create Data Space Index instruction and have the same attributes as specified in the output mapping template in the Create Cursor instruction. Fork characters are not included in the returned key.

Candidate entries include entries which may not be reflected in the user buffer (may have been omitted by the selection criteria). Entries which may have been omitted by the key selection (refer to the Create Cursor instruction), will not be included in the candidate count.

The candidate entries to process designates a limit to the number of entries to process before returning to the MI user. This field must contain a zero or positive value when the candidate entries to process provided field contains a binary one and is otherwise ignored. A value of zero indicates there is no limit.

The candidate entries processed return field contains the count of candidate entries processed. This field is uninitialized if the candidate entries to process provided field contains a binary 0.

The candidate entries processed on primary data space return field contains the count of candidate entries processed on the primary data space of a join cursor. For a non-join cursor, this field contains the same value as candidate entries processed.

Refer to Retrieve Sequential Data Space Entries instruction for a description of the default values feedback fields.

The default values feedback is not returned when the candidate limit exceeded return field contains binary 1 value (exceeded the candidate limit).

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

The following charts summarize the possible results of a Set Cursor instruction for each of the valid requests.

Results of Set Cursor Without Data Space Index:

Set Cursor Rule Options to Address a Data Space	**Current State Explanation before Set Cursor Request**	**Results of Set Cursor**			
		Normal	**Entry Not Found (Exception Condition)**	**End of Path (Exception Condition)**	**Cursor Remains Not Set**
First or last	1. Data space not empty	X			
	2. All data spaces or data spaces requested are empty.			X	
Next or previous	1. Cursor set	X			
	2. Cursor set to last or first entry and no adjacent data space is among the active subset			X	
	3. Cursor set to last or first entry and an adjacent data space exists (see Note 1)	X			
	4. Cursor not positioned				X
Relative	1. Cursor set	X			
	2. New cursor setting would be set outside of data space bounds.			X	
	3. Cursor not positioned				X
	4. New setting designates deleted entry		X		
	5. Data space entry deleted, and deleted entry significance option specified	X			
Ordinal	1. Ordinal number within data space bounds	X			
	2. Ordinal number outside data space bounds			X	
	3. Data space entry deleted		X		
	4. Data space entry deleted, and deleted entry significance option specified	X			

Note: An adjacent data space must not be empty and must contain at least one nondeleted entry unless deleted entry significance is specified.

Results of Set Cursor With Data Space Index:

		Results of Set Cursor			
Set Cursor Rule Options to Address a Data Space Index	**Current State Explanation before Set Cursor Request**	**Normal**	**Entry Not Found (Exception Condition)**	**End of Path (Exception Condition)**	**Cursor Remains Not Set**
First or last	1. Index not empty	X			
	2. Index empty			X	
	3. Index does not contain key for any data space entry as specified by restricted search list.			X	
Next, previous, next unique, or previous unique	1. Cursor set	X			
	2. Cursor set to last (or first) key in the index			X	
	3. Cursor not positioned				X
	4. Cursor set to last (or first) key associated with restricted data space			X	
Relative, ordinal (see *Cursor Without Data Spaces Index Results* chart)					
Key equal	1. Key in index	X			
	2. Key not in index		X		
Key before, key-equal/before, key-equal/after, key after	1. Equal key in index	X			
	2. Equal key not in index. Next/previous key is in index.	X			
	3. Key not in index. Either no key < this key in index or no key > this key in index (depending on rule).			X	
Note: The cursor setting remains unchanged for all exception conditions.					

Authorization Required

- Operational
  - Data space affected

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Implicit locks
  - Implicit LSUP lock applied to the data space referenced by the cursor if lock option is lock and cursor activation options to do so

Events

0002 Authorization
    0101 Object authorization violation

0008 Data space index
    0301 Data space index invalidated

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X	X	X	
02 Boundary alignment	X	X	X	X	
03 Range	X	X	X	X	
06 Optimized addressability invalid	X	X	X	X	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0A Authorization					
01 Unauthorized for operation					X
10 Damage Encountered					
04 System object damage state	X	X	X	X	X
44 Partial system object damage	X	X	X	X	X
12 Data Base Management					
02 Key mapping error			X	X	X
03 Cursor not set	X				
05 Data space entry locked					X
06 Data space entry not found					X
07 Data space index invalid					X
08 Incomplete key description	X				
0A End of path					X
17 Key value changed since Set Cursor					X
19 Invalid rule option					X
27 Data space index key with invalid floating-point field					X
30 Specified data space entry rejected					X
32 Join value changed					X
34 Non-user exit selection routine failure					X
38 Derived field operation error					X
1A Lock State					
01 Invalid lock state	X				
1C Machine-Dependent Exception					
03 Machine storage limit exceeded					X
06 Machine lock limit exceeded	X				
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X	X	X	X	
02 Object destroyed	X	X	X	X	X
03 Object suspended	X	X	X	X	
04 Object not eligible for operation					X
05 Object not available to process	X				

Exception	Operands 1	2	3	4	Other
24 Pointer Specification					
01 Pointer does not exist	X	X	X	X	
02 Pointer type invalid	X	X	X	X	
03 Pointer addressing invalid object	X				
2A Program Creation					
06 Invalid operand type	X	X	X	X	
07 Invalid operand attribute	X	X	X	X	
08 Invalid operand value range	X	X	X	X	
0A Invalid operand length		X	X	X	
0C Invalid operand ODT reference	X	X	X	X	
0D Reserved bits are not zero	X	X	X	X	X
2C Program Execution					
06 Instruction cancellation	X	X	X	X	X
07 Instruction termination					X
30 Journal Management					
02 Entry not journaled					X
32 Scalar Specification					
01 Scalar type invalid			X	X	
02 Scalar attributes invalid			X	X	
03 Scalar value invalid			X	X	
38 Template Specification					
01 Template value invalid		X			
3E Commitment Control					
06 Commit control resource limit exceeded					X

UPDATE DATA SPACE ENTRY (UPDSEN)

Op Code (Hex)	Operand 1	Operand 2
0492	Cursor	Interface buffer

Operand 1: System pointer.

Operand 2: Space pointer.

Description: The first data space entry of the locked entry queue associated with the operand 1 cursor (which must be activated to the current process) is updated with information provided in the interface buffer addressed by operand 2. The fields to be updated must be presented in the interface buffer in the format and sequence established by the input mapping template specification provided in the Create Cursor instruction. Fields in the data space entry that are not included in the input mapping template are unchanged in the data space entry, unless the deleted entry update option was previously specified when this entry was locked. In which case, the unmapped fields receive default values. All of the data space indexes referencing the data space are updated to reflect the changes.

If the data space entry is being updated under commitment control, the lock identified by the cursor's locked entry queue is transferred from the cursor's locked entry queue to the controlling commit block. The data space entry remains locked to the commit block until all uncommitted changes controlled through the commit block are committed or decommitted. If there are any valid unique keyed data space indexes over this data space entry, the key for the entry before it was changed in that data space index is reserved until this change is committed or decommitted. A data space entry may not be added or changed, which results in a key that conflicts with the old key of this data space entry. All views of the data space entry indicate that the entry has been changed. Because it remains locked, no other user of the data space may change the updated data space entry. The issuing process may update or delete this entry through this same cursor or another cursor that is under control of the same commit block.

If this change is later decommitted, the original data space entry is placed back in the data space at its original ordinal entry position.

This instruction must have been preceded by a successful Set Cursor instruction (with a lock entry option specified) that caused the entry to be locked. At the successful completion of the instruction, the updated entry is unlocked and the lock is transferred to the controlling commit block. The implicit LSUP lock is removed from the data space, if the activate cursor options for the cursor indicate for this to be done. If no entry is locked, then an exception is signaled. Errors in this instruction cause the entry to remain locked.

If the update operation intent was not indicated when the Activate Cursor instruction was performed, an invalid data base operation exception is signaled.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Update
  - Data space affected

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Implicit locks
  - Implicit LSUP lock removed from the affected data space

Events

0002 Authorization
    0101 Object authorization violation

0008 Data space index
    0301 Data space index invalidated

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port became unusable

001C Journal space
    0301 Threshold reached

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		X
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
12 Data Base Management			
01 Conversion mapping error			X
09 Duplicate key value in existing data space entry			X
0D No entries locked	X		
0F Duplicate key value in uncommitted data space entry			X
21 Unable to maintain unique key DSI			X
23 Data space index select routine failure			X
25 Invalid data base operation	X		
27 Data space index key with invalid floating-point field			X
34 Non-user exit selection routine failure			X
36 No mapping code specified	X		
37 Operation not valid with join cursor	X		
38 Derived field operation			X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	X
03 Object suspended	X	X	
05 Object not available to process	X		
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
30 Journal Management			
02 Entry not journaled			X
32 Scalar Specification			
01 Scalar type invalid	X	X	

Chapter 17. Source/Sink Management Instructions

The following chapter describes the source/sink management instructions. The instructions are in alphabetic order. For an alphabetic summary of all the instructions, see *Appendix A. Instruction Summary*.

CREATE CONTROLLER DESCRIPTION (CRTCD)

Op Code (Hex)	Operand 1	Operand 2
0496	Controller description	Controller description template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: A CD (controller description) is created in accordance with the controller description template. A system pointer that addresses the created CD is returned in the pointer specified by operand 1. The template identified by operand 2 must be 16-byte aligned, and any pointers specified within the template must also be 16-byte aligned. The format of this template is as follows:

• Template size specification	Char(8)
– Size of template	Bin(4)
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attribute 1 = Permanent (required)	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Reserved (binary 0)	Bit 2
– Access group 0 = Not member of access group (required)	Bit 3
– Replacement option 0 = Create as new (required)	Bit 4
– Reserved (binary 0)	Bits 5-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

- Performance class — Char(4)
  - Space alignment — Bit 0
    - 0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    - 1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.
  - Reserved (binary 0) — Bits 1-4
  - Main storage pool selection — Bit 5
    - 0 = Process default main storage pool is used for object.
    - 1 = Machine default main storage pool is used for object.
  - Reserved (binary 0) — Bit 6
  - Block transfer on implicit access state modification — Bit 7
    - 0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
    - 1 = Transfer the machine default storage transfer size. This value is 8 storage units.
  - Reserved (binary 0) — Bits 8-31

- Reserved (binary 0) — Char(39)

- CD definition data — Char(16)
  - CD type — Char(2)
    - Hex 0000 = Attached to system
    - Hex 0100 = Attached to ND
  - Controller identification — Char(8)
    - Unit type — Char(4)
    - Model number — Char(4)
  - Reserved (binary 0) — Char(6)

- CD specific data — Char(*)

Note: The value associated with each entry shown here with an asterisk (*) is ignored by this instruction.

The created object is owned by the user profile governing process execution. The user profile that owns the object is implicitly granted all authority states to the object. The storage occupied by the created object is charged to this same user profile.

The template size specification entry within the CD template must indicate the number of bytes to be used in defining the CD to be created.

The object identification specifies the symbolic name that identifies the object within the machine. A type code of hex 12 is implicitly supplied by the machine. The object identification is used on materialize instructions to identify the object and also to locate the object through the machine context.

Addressability to the CD is inserted in the machine context.

A space that is fixed or variable in size can be associated with the created object. The initial allocation of storage for the space is as specified in the size of space entry. The machine allocates a space at least the size specified; the actual size allocated is machine model dependent. (The maximum amount of storage that can be specified for the associated space is approximately 16 MB minus 4 K.) Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this byte value is also used to initialize the new additional bytes in the space. An associated space is not allocated for a fixed size space of zero length. The maximum size of a CD object is approximately 4 K bytes.

The performance class parameter provides information that allows the machine to manage the object with consideration for the overall performance objectives of operations involving the context.

One format is defined to specify the creation of a CD. The CD specific data entry defines this structure, and the CD type entry defines the structure of the remaining portion of the template.

• Forward object group	Char(32)
– Forward object pointer, ND For type 00, binary 0 For type 10, forward ND (if unspecified, binary 0)	System pointer
– Switched network forward, ND connection (if unspecified, binary 0)	System pointer*
• Backward pointer list data	Char(32)
– Pointer to backward object list (if unspecified, binary 0)	Space pointer
– Number of backward object pointers	Bin(2)
– Reserved (binary 0)	Char(14)
• Physical definition data	Char(16)
– Physical address	Char(8)
For type 00,	
Reserved (binary 0)	Char(6)
CD (OU number) address	Bit 16
For type 10,	
Reserved (binary 0)	Char(4)
Station's link address for SDLC, BSC multipoint, or X.25 switched virtual circuit. Binary 0 for BSC point-to-point.	Char(2)
ND (OU#) address for leased line or X.25 permanent virtual circuit. Binary 0 for switched line or X.25 switched virtual circuit.	Char(2)
– Power control Hex 0000 = No Hex 0100 = Yes	Char(2)
– Reserved (binary 0)	Char(6)
• State change/status definition	Char(16)
– State change/status field	Char(8)*
– Reserved (binary 0)	Char(8)

• ND candidate list data	Char(32)
– Pointer to ND candidate list (binary 0 for CD types 00 and 10 that are nonswitched and do not have the switched backup mode)	Space pointer
– Number of ND candidate pointers	Bin(2)
– Reserved (binary 0)	Char(14)
• Station control information	Char(32)
– Exchange identification (SNA only)	Char(21)
Byte 0	Char(1)
XID format	Bits 0-3
Physical unit type	Bits 4-7
XID field length	Char(1)
XID	Char(4)
Block number	Bit(12)
Specific ID	Bit(20)
Reserved (binary 0)	Char(2)
Configuration flags	Char(1)
Physical unit characteristics	Char(1)
Maximum length received	Char(2)
Reserved (binary 0)	Char(4)
Frames limit	Char(1)
Reserved (binary 0)	Char(4)
– Station definition	Char(1)
Line discipline 10 = SDLC 01 = BSC 11 = Defined by line discipline extension field 00 = Other or not applicable	Bits 0-1
Switched network 0 = No (nonswitched network or not applicable) 1 = Yes (switched network)	Bit 2
Role 0 = BSC station, secondary SDLC station, APPC station, or not applicable 1 = Primary SDLC station, APPC station, or BSC multipoint tributary	Bit 3
Switched network backup feature (on nonswitched network) 0 = No 1 = Yes	Bit 4
Data rate select feature 0 = No 1 = Yes	Bit 5
Reserved (binary 0)	Bits 6-7

– Line discipline extension field	Char(1)
Hex 00=Not applicable (defined previously)	
Hex 01=X.25	
– Reserved (binary 0)	Char(1)
– Path information unit type (SNA format ID, SDLC only)	Char(2)
– Reserved (binary 0)	Char(6)
• Selected mode data	Char(16)
– Selected mode	Char(2)
Reserved (binary 0)	Bits 0-2
Switched network backup mode	Bit 3
0 = Nonswitched mode	
1 = Switched mode	
BSC protocol modes	Bit 4-6
000 = Normal mode	
001 = MTAM mode	
010 = 3270 emulation mode	
Reserved (binary 0)	Bits 7-15
– Delayed contact control	Char(2)
Hex 0000 = No	
Hex 0100 = Yes	
– BSC delay data	
Delay time (this program)	Bin(2)
Delay time (device)	Bin(2)
– Reserved (binary 0)	Char(8)
• Activate physical unit information	Char(16)
– ACTPU required	Char(1)
Hex 00 = No	
Hex 01 = Yes	
– ACTPU parameters	Char(9)
Request code	Char(1)
Activation type	Char(1)
Profile number	Char(1)
SSCP ID	Char(6)
– Reserved (binary 0)	Char(6)
• Dial digits	Char(32)
– Reserved (binary 0)	Char(6)
– Number of dial digits used	Bin(2)
– Dial digits field	Char(16)
– Reserved (binary 0)	Char(8)
• Specific characteristics	Char(*)
– Specific characteristics length (contains the length of the following specific data area)	Bin(2)
– Specific data	Char(*)
• XID information area	
– XID information length (contains the length of the following XID data)	Bin(2)
– XID information data	Char(*)
• Unit-specific contents	Char(*)
– Unit-specific length (contains the length of the following unit-specific parameters)	Bin(2)
– Unit-specific modify length (contains the length of the specific area that can be modified)	Bin(2)
– Unit-specific parameters	Char(*)
Area that can be modified	Char(*)
Area that can only be materialized	Char(*)

For BSC controllers, the following unit-specific parameters have been defined:

- Unit-specific length for switched line controllers (contains a value 112+18y, where y is the number of remote XIDs in the following list).

- Unit-specific length for nonswitched line controllers is reserved for 112 bytes of binary zeros.

- Unit-specific modify length (same as both above bullets).

• Unit-specific parameters	Char(*)
– BSC miscellaneous attributes	Char(2)
First switched operation indicator	Bit 1
0 = Device must transmit first	
1 = Device must receive first	
Reserved (binary 0)	Bit 15
– Reserved (binary 0)	Char(8)
– This system's XID	Char(18)
XID length	Bin(2)
XID	Char(15)
Reserved (binary 0)	Char(1)
– Number of remote XIDs	Bin(2)
– Reserved (binary 0)	Char(1)
– MTAM signon/logon	Char(81)
– Remote XID list for dial in/out	Char(*)
XID length	Bin(2)
XID	Char(15)
ID type	Char(1)

- Backward object list, LUD	System pointer

This entry defines the list of backward objects LUDs (logical unit descriptions) and is located by the backward object list pointer. One pointer entry (or binary 0's) is present for each attached LUD (logical unit description).

- ND candidate pointers, ND (If not specified, binary 0)	System pointer

This list, if present, defines the ND (network description) candidates and is located by the ND candidate list pointer. The number of entries in the list is determined by the number of ND candidate pointers.

Note: The value associated with each entry shown here with an asterisk (*) is ignored by this instruction.

A CD logically represents a physical device controller or a communications controller for devices in a communications network. Two versions of CDs are supported. A type 00 CD attaches directly to the system. A type 10 CD attaches to the system through an ND. The structure of the creation template (size and order of entries) for each CD type is identical; however, the values and meanings of certain entries may depend on the type and, in some cases, on other values specified in the template.

The unit type indicates the IBM product number or a representative number for other equipment manufacturers' products. The model number defines the unit model number of the controller.

The forward object group indicates any association of the CD with an ND. The forward object pointer indicates a permanent association. For type 00 CDs, this entry is not used and must be binary 0. For type 10 CDs, this entry is used to specify the line that the CD is attached to except when it is attached to a switched network. This entry is optional in a create template. If a forward object is not specified for type 10 (indicated by binary 0's), no association is made until a Create Network Description instruction is executed that specifies this CD as one of its backward objects. A type 10 CD that requires a forward object is unusable until the forward object is specified. If a forward object pointer is specified, then the object identified by the system pointer must be in a varied off state to allow the connection of this CD. If the object is not in the varied off state, a source/sink state invalid exception is signaled.

The switched network forward connection pointer is used only for switched networks and can be materialized to determine the ND currently used by the CD. This entry is ignored in the creation template.

The backward object list identifies the set of LUDs to be associated with the CD. A pointer locates the list of system pointers identifying the LUDs. This list is optional and, if not specified (indicated by binary 0's in the pointer to backward object list entry), the association of the CD to any LUDs is made when the LUDs are created. Any LUDs specified as associated with this CD cannot be associated with another CD.

The physical address defines a unique address by which the controller is known in the system. The definition depends on the CD type field as indicated in the template.

For a type 00 CD, the operation unit number is specified. For a type 10 CD, the CD station address and the operational unit number of the associated ND are specified. For switched connections, the ND operational unit number is not specified on create (binary 0). For switched connections and also for nonswitched connections that have the switched network backup feature, a unique address can be established only by an association of the exchange identification (SSCP ID for a primary station) with the physical address. Exchange identification (XID) or SSCP ID assignments must be made by the user to ensure a unique address. If a unique address is not established, the source/sink physical address exception is signaled by the Create instruction.

The power control entry (allowed only for CD type 10) specifies whether power to the control unit can be remotely turned on or off from the system. If yes (hex F1F0), the control unit power is turned off or on through the Modify Controller Description instruction.

The status change/status definition entry is used to change the state of the CD through a Modify Controller Description instruction or to determine the current state of the CD through the Materialize Controller Description instruction. No information can be specified on the Create Controller Description instruction.

The ND candidate list defines a set of network descriptions that describe line appearances suitable to the characteristics of the created CD. The list is used in switched connections to define possible switched lines with which this CD can communicate. The list is not present (indicated by binary 0's in the pointer to ND candidate list entry) for a type 00 CD or for a type 10 CD that is nonswitched and does not have switched network backup mode.

The list of pointers for the ND candidate list must be either system pointers to network descriptions or binary 0's. The desired number of ND candidates must be supplied because the number of these pointers cannot be changed once the CD is created.

Station control information: This entry is made up of subentries, which are as follows:

- XID (Exchange identification, SNA only): Station ID sequence for establishing identity of the station. The contents of this field allow establishing a unique identification of the physical unit that this CD object is to represent. The block number is assigned by the manufacturing plant at the time of manufacture or installation for every physical unit. Each unit can identify itself with this XID information.

- Station definition: The subentries for this entry are defined as follows:
  - Line discipline and line discipline extension field: These entries define the protocol to be used for link level communications. All stations communicating over the link must follow the same protocol (SDLC, BSC, or X.25) at any point in time.
  - Switched network: This entry establishes whether or not the data link is established through the public switched network. If not, a nonswitched or private facility is implied.
  - Role: For SNA, SDLC or X.25 (secondary/primary): When this entry is set, this controller represents the SNA station that assumes the role of a primary station on this line. Otherwise, either this CD represents a secondary SNA station on a network for which System/38 is the primary station or the field is not applicable.

    For BSC–When this entry is set, this controller represents the far end BSC control station for which System/38 has assumed the role of a multipoint tributary station. Otherwise, normal BSC point-to-point communication is assumed.

    For APPC (advanced program to program communications)–This entry should be set to one if this CD represents a primary station and is, or will be, attached to an ND that is a secondary station. This entry should be set to zero if this CD represents a temporary station and is, or will be, attached to an ND that is a primary station. During APPC operations, the role indicator may be changed by the machine when the network connection is made because any station (CD) can change its role from one interval connection to another interval connection.

  - Switched network backup: This entry indicates that the station has a modem with the switched network backup capability. The normal communication facility is nonswitched. To use this option, the selectable mode field bit for switched backup operation must be set.
  - Data rate select: This entry indicates that the station has a modem that is capable of operating at either full speed or half speed. To use this option, the selectable mode field bit for selected rate must be set.

- Path information unit type: This entry defines the SNA format identifier supported by this controller.

Selected mode data: This field is used by modify instructions and create instructions to initialize the operating state of the CD for whichever options have been defined for this station. The switched network backup mode bit determines if the CDS that are for switched network backup are to be operated in nonswitched or switched backup mode. The BSC protocol modes bits are used to specify the binary synchronous communications operating protocol. Normal mode is used for all protocol mode except MTAM (MULTI-LEAVING telecommunication access method). MTAM is used for BSC remote job entry applications. The delayed contact control indicator is used by nonswitched stations to periodically attempt to contact the station if the initial contact was not successful. If the indicator is set, the CD contact event will be signaled only after the station is contacted. If the indicator is not set, the CD contact event (unsuccessful subtype) will be signaled after an unsuccessful attempt to contact the station and no further attempt will be made.

Activate physical unit parameters: This entry defines the parameters used to establish the MSCP (machine services control point) to the physical unit session. The SSCP ID subentry is the identification of the SNA system services control point in the network and, for the case of primary controllers, must be established uniquely within the System/38 network.

Dial digits: This entry contains the number to be dialed to establish a connection with the station represented by this CD.

Specific characteristics: This entry defines the set of characteristics that are described uniquely for each controller at the time of object creation. See the *IBM System/38 Functional Reference Manual—Volume 2*, GA21-9800 for the details concerning local source/sink devices, communications, and locally attached work stations for this entry.

XID information area: This entry can be materialized and defines the XID (exchange ID) data area. For SDLC, the two data formats consist of a fixed-length format and a variable-length format. The fixed-length format is a subset of the variable-length format and identifies the physical unit type and specific station. The variable format provides the physical unit description, which includes configuration characteristics, information field length, maximum output count, and addressing characteristics. For BSC, the format is fixed length for XIDs as defined in the remote XID lists of the unit-specific contents area. On a create template, only the XID information length field is referenced to allocate the proper amount of space in the CD. The remaining part of the XID information is ignored.

Unit-specific contents: This entry defines the set of specific modifiable parameters and the parameters which are supplied by the machine (materializable only) for the controller unit described in this CD. The nonmodifiable part of this entry is ignored on a create or modify instruction. The modifiable part of this entry may or may not be required to contain correct data at the time of creation depending on the specific controller that is to be created.

For BSC controllers, the following fields are defined:

- BSC first switched operation indicator entry specifies to the System/38 in what direction data transfer is allowed to take place when a switched link is first activated. Subsequent sessions are not controlled by this indicator, but are controlled by the first request I/O operation, which must indicate in which direction data transfer is to take place.

- This system's XID entry specifies the identification characters to be used by the System/38 when identifying itself to the far end switched stations.

- MTAM signon/logon entry contains the sign on or log on data that is passed to the host system at connection time when operating in the MTAM protocol mode.

- Remote XID list entry specifies the array of possible XIDs that the far end station represented by this CD is allowed to identify itself to this system. On switched connections, the System/38 ensures that the incoming XID matches the requirements of one of the entries in this list before completing the connection. The received XID will then be returned in the XID information area of this CD.

Additional information about this entry is contained in the *IBM System/38 Functional Reference Manual—Volume 2,* GA21-9800 for local source/sink devices, communications, and locally attached work stations.

The values supplied within the CD template must meet the requirements to create a CD for the physical controller being described. If the values are not compatible with limitations and ranges known to the machine, a template value invalid exception is signaled, and the CD is not created.

The physical address and exchange identification supplied within the template must be unique from any existing CDs. If not, a source/sink duplicate physical address exception is signaled and the CD is not created. The physical controller and its associated machine support components must be installed on the system before the CD can be created. If the internal machine configuration records do not indicate that these physical components are installed, a source/sink resource not available exception is signaled, and the CD is not created.

Authorization Required

- Privileged instruction
- Insert
  - User profile of creating process
- Operational
  - Source/sink objects identified in operand 2

Lock Enforcement

- Modify
  - User profile that is to own this object
  - Source/sink objects specified as forward and backward objects identified in operand 2

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	 2	Other
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
02 Privileged instruction			X
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
02 Machine context damage state			X
04 System object damage state		X	
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer address invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
32 Scalar Specification			
01 Scalar type invalid	X	X	
34 Source/Sink Management			
01 Source/sink configuration invalid			X
02 Source/sink duplicate physical address			X
03 Source/sink invalid object state			X
04 Source/sink resource not available			X
38 Template Specification			
01 Template value invalid		X	
02 Template size invalid		X	

CREATE LOGICAL UNIT DESCRIPTION (CRTLUD)

Op Code (Hex)	Operand 1	Operand 2
049A	Logical unit description	Logical unit description template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: A LUD (logical unit description) is created in accordance with the logical unit description template. A system pointer that addresses the created LUD is returned in the pointer specified by operand 1. The template identified by operand 2 must be 16-byte aligned and any pointers specified within the template must also be 16-byte aligned. A LUD template is defined as follows:

• Template size specification	Char(8)
– Size of template	Bin(4)
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attribute	Bit 0
1 = Permanent (required)	
– Space attribute	Bit 1
0 = Fixed-length	
1 = Variable-length	
– Reserved (binary 0)	Bit 2
– Access group	Bit 3
0 = Not member of access group (required)	
– Replacement option	Bit 4
0 = Create as new (required)	
– Reserved (binary 0)	Bits 5-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(39)
• LUD definition data	Char(16)
– LUD type	Char(2)
00= Attached directly to system	
10= Attached to type 00 CD	
30= Attached to type 10 CD	
– LUD identification	Char(8)
Device type	Char(4)
Model number	Char(4)
– Reserved (binary 0)	Char(6)
• LUD specific data	Char(*)

The created object is owned by the user profile governing process execution. The user profile that owns the LUD is implicitly granted all authority states to the object and also charged for the storage occupied by the created object.

The template size specification entry within the CD template must indicate the number of bytes to be used in defining the CD to be created.

The object identification specifies the symbolic name that identifies the object. A type code of hex 10 is implicitly supplied by the machine. The object identification is used to identify the object on materialize instructions and also to locate the object through the machine context.

Addressability to the LUD is inserted in the machine context.

A space that is fixed or variable in size can be associated with the created object. The initial allocation of storage for the space is as specified in the size of space entry. The machine allocates a space of at least the size specified; the number of bytes allocated is machine model dependent. (The maximum amount of storage that can be specified for the associated space is approximately 16 MB minus 4 K.) Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this value is also used to initialize the additional bytes in the space. A fixed size space with a zero length causes no space to be allocated. The maximum size of a LUD object is 4 K bytes.

The performance class parameter provides information that allows the machine to more effectively manage the object.

Three types of LUDs are defined:

Type	**Attachment**
00	Attached directly to the system
10	Attached to a type 00 CD
30	Attached to a type 10 CD

This attachment mechanism defines the information content and, therefore, the structure of the LUD template.

The following structure is used to define LUD type 00 (attached to system), type 10 (attached to CD), and type 30 (attached to CD and to ND).

• Pointer group data	Char(16)
– Forward object pointer (if unspecified, binary 0)	System pointer
• Physical definition data	Char(16)
– Physical address	Char(8)
For LUD type 00,	
a. Reserved (binary 0)	Char(6)
b. LUD OU number	Bit 16
For LUD type 10,	
a. Reserved (binary 0)	Char(2)
b. LU address	Char(2)
c. Reserved (binary 0)	Char(2)
d. CD OU number	Bit 16
For LUD type 30,	
a. Reserved (binary 0)	Char(2)
b. LU address	Bit 16
c. CD station address	Bit 16
d. ND OU number (binary 0 if switched line)	Bit 16
– Power control Hex 0000 = No Hex 0100 = Yes	Char(2)
– Reserved (binary 0)	Char(6)
• State/status definition	Char(16)
– State change/status field	Char(8)*
– Reserved (binary 0)	Char(8)

• Session definition data	Char(32)
– Session information (SDLC devices)	Char(20)
Pacing (inbound)	Bin(2)
Pacing (outbound)	Bin(2)
RU size (buffer size)	Bin(2)
Reserved (binary 0)	Bin(2)
ACTLU required Hex 00 = No Hex 01 = Yes	Char(1)
ACTLU parameters	Char(3)
ACTLU response	Char(8)*
– Reserved (binary 0)	Char(4)
– Session information (BSC devices)	Char(4)
Modes	Char(1)
Contention resolution 0 = Secondary station role 1 = Primary role	Bit 0
Ignore request for test 0 = Perform request for test as requested 1 = Pass request for test as data	Bit 1
Reserved (binary 0)	Bits 2-7
– Reserved (binary 0)	Char(3)
– Reserved (binary 0)	Char(4)

• Load/dump definition data	Char(16)
– Load/dump indicator	Char(13)
Load/dump device	Char(1)
Hex 00=Not a load/dump device	
Hex 01=Load/dump device-noninterruptible and nonexchangeable	
Hex 11=Load/dump device-interruptible	
Hex 21=Load/dump device-exchangeable	
Operating mode	Char(1)
Hex 00=Data interchange mode–not load/dump	
Hex 01=Load mode	
Hex 02=Dump mode	
Load/dump pending	Char(2)*
Hex 0000 = None	
Hex 0100 = Load pending	
Hex 0200 = Dump pending	
Corresponding primary address	Char(2)*
Load/dump exchange status (exchangeable devices only)	Char(3)
For materialize LUD operation:	
Hex 000000 = This device is not current	
Hex 010000 = This device is current	
For modify LUD operation:	
Hex 000000 = No modification requested	
Hex 01YYYY = Exchange this device to become current where YYYY is the logical unit address of the LUD which is now current	
Reserved (binary 0)	Char(1)
Load/dump performance attributes	Char(3)
Load/dump buffer storage	Bin(2)
Load/dump process priority	Char(1)
– Reserved (binary 0)	Char(3)
• Specific characteristics	Char(y + 2)
– Specific characteristics length (contains the length of the following specific characteristics area)	Bin(2)
– Specific data	Char(VAR)
• Retry value sets	Char(6y + 2)
– Retry value length (contains the length of the following retry value area)	Bin(2)
– Error type	Char(2)
– Error retry value	Bin(2)
– Reserved (binary 0)	Bin(2)
• Error threshold sets	Char(8y + 2)
– Error threshold length (contains the length of the following error threshold area)	Bin(2)
– Error type	Char(2)
– Threshold value	Bin(2)
– Reserved (binary 0)	Bin(4)
• Device-specific contents	Char(y + 4)
– Device-specific length (contains the length of the following device-specific parameters)	Bin(2)
– Device-specific modify length (contains the length of the device-specific area that is modifiable)	Bin(2)
– Device-specific parameters	Char(VAR)
Modifiable area	Char(VAR)
Materializable only area	Char(VAR)*

Note: The value associated with each entry shown here with an asterisk (*) is ignored by this instruction.

An LUD logically represents a physical device. An LUD can be associated with a CD (controller description) through the forward object pointer contained in the LUD. The CD represents the physical controller, physical I/O port, or communications line to which this device is attached. The LUD can be associated directly with the system when the physical device is directly attached to the system.

The forward object pointer establishes addressability to the associated forward objects through the forward object pointers supplied in the LUD template. Addressability is also established within these associated objects back to the LUD being created. It is not mandatory that the associated object pointer be supplied in the LUD template because as long as the pointer is supplied, either in the LUD or within the creation templates of these associated objects, proper addressability is established by similar logic within the Create instruction of the other source/sink objects. When the associated object pointer is supplied, this object must exist. If a forward object pointer is specified, then the object identified by the system pointer must be in a varied off state to allow this LUD to be attached. If the object is not in a varied off state, then the source/sink object state invalid exception is signaled. When the associated object pointer is not supplied, this pointer location in the template must contain 16 bytes of binary 0's.

When a forward object is not required (LUD type 00), the forward pointer location in the template must be binary 0. If the LUD template pointer area does not meet the previously mentioned requirements, an exception is signaled and the LUD is not created.

The LUD device type entry defines the IBM product number or a representation number for an end-use mechanism.

The physical address defines the unique address by which this device is known physically in the system. The content of the physical address entry depends on the attachment mechanism (LUD type) of the device.

The power control entry specifies whether the device is capable of having its power turned on or off independent of the system. If the device has this capability, it is done through the Modify Logical Unit Description instruction.

The state/status definition entry is not used by the Create Logical description instruction. But this entry can be materialized (Materialize Logical Unit Description instruction) to show the current status of the LUD. This entry can be modified (Modify Logical Unit Description instruction) to change the status of the LUD.

The session information entry defines parameters that allow the machine to control the device while in use. The following session parameters for SDLC devices are defined:

- The pacing inbound and pacing outbound entries are each comprised of a 2-byte count entry. For more information about pacing, refer to the *IBM System/38 Functional Reference Manual—Volume 2,* GA21-9800 for local source/sink devices, communications, and locally attached work stations and to the *IBM Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic.*

- The RU (request/response unit) size entry defines the size of the buffer in the unit described by this object.

- The ACTLU (activate logical unit) required entry defines whether the ACTLU parameters and ACTLU response entries have any meaning for this device.

- The ACTLU parameters entry is used by the MSCP (machine services control point) to establish the MSCP-to-LU (logical unit) session and to provide the data that can be received as a response to an ACTLU sequence.

- The ACTLU response entry is a field in which the only characters that can be materialized are the response characters provided by the device when the MSCP-to-LU session is established.

The following session parameters for BSC devices are defined:

- The contention resolution entry indicates whether the System/38 should act in a secondary station role and lose contention conflicts or act in a primary role for contention resolution.

- The ignore request for test entry indicator being set on causes the machine to pass all requests for tests received from the station to the user by a receive request I/O operation. If this indicator is not set on, the machine performs any tests requested by the remote station, and test results are recorded in the error log.

The load/dump indicator entry defines whether an I/O device can be used for the load/dump function.

For those devices that can be used as load/dump devices, the load/dump indicator further defines whether the device is to be used in load mode, dump mode, or normal mode. Noninterruptible load/dump devices can operate in normal, load, or dump modes but cannot change modes while in active or inactive session. Interruptible load/dump devices can also change modes while in an inactive session state, according to the set of rules described in the Modify LUD instruction. For these interruptible devices only, the load/dump pending field indicates whether any pending load/dump activity exists if the LUD is in an inactive session state. An exchangeable load/dump device can exchange load or dump request I/O operations with other exchangeable load/dump devices if the devices are activated in the same modes (all in load mode or all in dump mode) and have the same corresponding primary device address. The load/dump exchange status field of the Modify LUD instruction is used to cause an exchange to occur and indicates which device is current at any time.

The load/dump performance attributes are used to establish a range of load/dump performance characteristics for the process that currently exists relative to the process that is doing the load/dump.

The load/dump process priority field is used to specify the priority of the load/dump processes within the machine. A value of 0 has the highest priority.

The load/dump buffer storage field specifies the maximum amount of space that is allowed to be extracted from the user's process storage pool for this load/dump session. This value must be specified in units equal to the machine minimum transfer size for storage pools. This value can be obtained by using the Materialize Process Attributes instruction. If 0 is specified, load/dump will use an appropriate value which includes previous levels of support that did not have this parameter. If hex FFFF is specified, load/dump chooses the largest possible value allowed for the device that is being used.

Only the load/dump device indicator, load/dump operating mode indicators, and load/dump process performance attributes are used on a create. The remaining entries can be materialized and modified only and are ignored by the Create instruction.

The specific characteristics entry defines the set of characteristics that uniquely describe each device during the time an object is created. For the size and contents of this entry for a particular device, refer to the *IBM System/38 Functional Reference Manual*—Volume 2, GA21-9800 for local source/sink devices, communications, and locally attached work stations.

The retry value sets entry contains values that specify limits for various error types beyond which a higher level error recovery is invoked.

The error threshold sets values are used by the internal error logging algorithms to determine the frequency for adding device error information records to the error log.

The device-specific contents entry defines the set of specific parameters that can be modified and also those parameters that are supplied by the machine (materializable only) for the device described in this LUD. The materializable only area of this entry is ignored by this instruction. The modifiable part of this entry may or may not be required to contain correct data at the time of creation. Further definition of the parameters for the various devices is contained in the *IBM System/38 Functional Reference Manual—Volume 2*, GA21-9800.

The values supplied within the LUD template must meet the requirements to create an LUD for the physical device being described. If the values are not compatible with limitations and ranges known to the machine, a template value invalid exception is signaled, and the LUD is not created.

The physical address that is supplied within the template must be unique from any existing LUDs. If not, a source/sink duplicate physical address exception is signaled, and the LUD is not created. The physical device and its associated machine support components must be installed on the system before the LUD can be created. When the internal machine configuration records do not indicate that these physical components are installed, a source/sink resource not available exception is signaled, and the LUD is not created.

Authorization Required

- Privileged instruction
- Operational
  - Source/sink objects identified in operand 2
- Insert
  - User profile of creating process

Lock Enforcement

- Modify
  - User profile that is to own this object
  - Source/sink object specified as the forward object identified in operand 2

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded
  0501 Machine address threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0201 Machine context damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
02 Privileged instruction			X
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
02 Machine context damage state			X
04 System object damage state		X	
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer address invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
32 Scalar Specification			
01 Scalar type invalid	X	X	
34 Source/Sink Management			
01 Source/sink configuration invalid			X
02 Source/sink duplicate physical address			X
03 Source/sink invalid object state			X
04 Source/sink resource not available			X
38 Template Specification			
01 Template value invalid		X	
02 Template size invalid		X	

CREATE NETWORK DESCRIPTION (CRTND)

Op Code (Hex)	Operand 1	Operand 2
049E	Network description	Network description template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: An ND (network description) is created in accordance with the ND template. A system pointer that addresses the created ND is returned in the pointer specified by operand 1. The template identified by operand 2 must be 16-byte aligned and any pointers specified within the template must also be 16-byte aligned. This template is defined as follows:

• Template size specification	Char(8)
– Size of template	Bin(4)
– Number of bytes available for materialization	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attribute	Bit 0
1 = Permanent (required)	
– Space attribute	Bit 1
0 = Fixed-length	
1 = Variable-length	
– Reserved (binary 0)	Bit 2
– Access group	Bit 3
0 = Not member of access group (required)	
– Replacement option	Bit 4
0 = Create as new (required)	
– Reserved (binary 0)	Bits 5-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Reserved (binary 0)	Bit 6
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Reserved (binary 0)	Bits 8-31
• Reserved (binary 0)	Char(39)
• ND definition data	Char(16)
– ND type	Char(2)
00 = CDs attached	
– Reserved (binary 0)	Char(14)
• ND specific data	Char(*)

Note: The value associated with each entry shown here with an asterisk (*) is ignored by this instruction.

The template size specification entry in the ND template must indicate the number of bytes of the ND that is to be created.

The object identification specifies the symbolic name that identifies the object. A type code of hex 11 is implicitly supplied by the machine. The object identification identifies the object on materialize instructions and also locates the object in the machine context.

A space that is fixed or variable in size can be associated with the created object. The initial allocation of storage for the space is as specified in the size of space entry. The machine allocates a space at least the size specified; the actual size allocated is machine model dependent. (The maximum amount of storage that can be specified for the associated space is approximately 16 MB minus 4 K.) Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this value is also used to initialize the additional bytes in the space. A fixed size space of zero length causes no space to be allocated. The maximum size of an ND (network description) object is approximately 4 K bytes.

Addressability to the ND is inserted into the machine context.

The created object is owned by the user profile that governs process execution. The user profile that owns the created object is implicitly granted all authority states to the object and charged for the storage occupied by the created object.

The performance class parameter provides information that allows the machine to manage the object with consideration for the overall performance objectives of operations involving the context.

One type of ND is defined. Type 00 defines the NDs that are attached to one or more CDs.

• Backward object pointer group	Char(48)
– Pointer to backward object list (if unspecified, binary 0)	Space pointer
– Switched network Backward connection CD (if unspecified, binary 0)	System pointer
– Number of backward object pointers	Bin(2)
– Reserved (binary 0)	Char(14)
• Physical definition data	Char(16)
– Physical address	Char(8)
Reserved (binary 0)	Char(6)
Operational unit number	Bit 16
– Reserved (binary 0)	Char(8)
• State/status definition	Char(16)
– State change/status field	Char(8)*
– Reserved (binary 0)	Char(8)

• Line definition data	Char(16)
– Line definition	Char(10)
Line data	Char(4)
Line discipline	Bits 0-3
1000 = SDLC	
0100 = BSC	
0010 = X.25/HDCL	
Switched network	Bit 4
0 = No (nonswitched network)	
1 = Yes (switched network)	
Switched network backup	Bit 5
0 = No	
1 = Yes	
Data rate select	Bit 6
0 = No	
1 = Yes	
Role	Bit 7
SDLC	
0 = Primary SDLC station	
1 = Secondary SDLC station	
BSC	
0 = BSC point-to-point	
1 = BSC multipoint tributary	
X.25	
0 = Normal DTE appearance	
1 = DCE appearance (S/38 is the DCE)	
Reserved (binary 0)	Bits 8-11
NRZI (non-return-to-zero [inverted])	Bit 12
0 = No	
1 = Yes	
New sync	Bit 13
0 = Not applicable	
1 = New sync interface signal	
Nonclocked modem	Bit 14
0 = No	
1 = Yes	
OEM modem	Bit 15
0 = No	
1 = Yes	
Wire	Bits 16-17
00 = Two-wire backup line (if applicable), two-wire normal line	
01 = Two-wire backup line (if applicable), four-wire normal line	
10 = Four-wire switched backup line, two-wire normal line	
11 = Four-wire switched backup line, four-wire normal line	
Multipoint network	Bit 18
0 = Point-to-point	
1 = Multipoint	
Reserved	Bit 19
BSC multipoint protocol	Bit 20
0 = Not applicable	
1 = Multipoint tributary station	
Reserved (binary 0)	Bits 21-31
Switched data	Char(1)
Autodial	Bit 0
0 = No	
1 = Yes	
Autoanswer	Bit 1
0 = No	
1 = Yes	
Autoanswer sequence	Bit 2
0 = Sequence 0	
1 = Sequence 1	
Answer tone generation	Bit 3
0 = No	
1 = Yes	
Marks/spaces for answer tone	Bit 4
0 = Transmit spaces	
1 = Transmit marks	
Special answer tone (Far-end modem that requires 2025 hertz answer tone)	Bit 5
0 = No	
1 = Yes	
DCE (data communication equipment)	Bits 6-7
00 = Not applicable (nonswitched line) or IBM integrated modem not in US or Canada	
01 = Switched line not in US or Canada and not IBM integrated modem	
10 = Switched line in US or Canada including IBM-integrated modem	
11 = Reserved	
Reserved (binary 0)	Char(1)
Line speed/100	Bin(2)
Secondary address (binary 0 for primary SDLC and BSC point-to-point)	Char(2)
SDLC secondary address	Bits 0-7
Reserved (binary 0)	Bits 8-15
BSC multipoint tributary address	Bits 0-15
X.25	
Local HDLC address	Bit 0-7
Remote HDLC address	Bit 8-15
– User assigned identification	Char(4)
Reserved (binary 0)	Bit 12
User identification	Bit 20
– Reserved (binary 0)	Char(2)

• Communications initialization data	Char(16)
– Initialization data	Char(8)*
– Reserved (binary 0)	Char(8)
• Exchange identification data	Char(16)*
– XID characteristics	Char(2)
– XID	Char(4)
SNA block number	Bit 12
Machine serial (or user assigned ID)	Bit 20
– Reserved (binary 0)	Char(10)
• Selectable mode data	Char(16)
– Selectable modes	Char(2)
Network selections	Char(1)
Reserved (binary 0)	Bits 0-1
Switched network backup mode	Bit 2
0 = Nonswitched mode	
1 = Switched mode	
Selected rate	Bit 3
0 = Full speed	
1 = Half speed	
Character encoding	Bit 4
0 = EBCDIC	
1 = ASCII	
BSC protocol modes	Bits 5-7
000 = Point-to-point	
001 = MTAM mode	
010 = 3270 emulation mode	
Switched network selections	Char(1)
Reserved (binary 0)	Bits 0-1
Switched connect method	Bits 2-3
00 = Nonswitched	
10 = Only dial in allowed	
01 = Only dial out allowed	
11 = Either allowed	
Autodial mode	Bit 4
0 = Manual dial	
1 = Autodial	
Autoanswer mode	Bit 5
0 = Manual answer	
1 = Autoanswer	
Switched secondary line inactivity disconnect (SDLC or BSC)	Bit 6
SDLC	
0 = No time-out	
1 = Time-out	
BSC	
0 = Time-out	
1 = No time-out	
Reserved (binary 0)	Bit 7
– Reserved (binary 0)	Char(14)

• Communications subsystem parameters data	Char(16)
– Communications subsystem parameters	Char(12)
Data terminal ready delay	Bin(2)
Reserved (binary 0)	Char(4)
Delayed contact timer	Bin(2)
SDLC idle state detection timer (BSC receive time-out timer)	Bin(2)
Nonproductive receive timer (reserved with BSC)	Bin(2)
– Reserved (binary 0)	Char(4)
• Eligibility object group	Char(32)
– Pointer to eligibility list (if unspecified, binary 0)	Space pointer
– Number of eligibility object pointers	Bin(2)
– Reserved (binary 0)	Char(14)
• Reserved group	Char(32)
– Pointer to reserved list (reserved, binary 0)	Space pointer
– Number of list entries (reserved, binary 0)	Bin(2)
– Reserved (binary 0)	Char(14)
• Specific characteristics	Char(y + 2)
– Specific characteristics length (contains the length of the following specific data area)	Bin(2)
– Specific data	Char(VAR)
• Retry value sets	Char(6y + 2)
– Retry value length (contains the length of the following retry value area)	Bin(2)
– Error type	Char(2)
– Error retry value	Bin(2)
– Reserved (binary 0 on creation template)	Bin(2)
• Line-specific contents	Char(y + 4)
– Line-specific contents length (contains the length of the following specific data)	Bin(2)
– Line-specific contents modify length (contains the length of the line-specific area that is modifiable)	Bin(2)
– Line-specific parameters	Char(VAR)
Area that can be modified	Char(VAR)
Area that can only be materialized	Char(VAR)*

- Backward object pointers — System pointer
  - CDs if ND type 00

  This list of pointers is located by the backward object list pointer and defines the set of objects attached to this ND. The number of entries is specified in the number of backward objects entry.

- Eligibility object pointers — System pointer
  - CDs if type 00
  - Binary 0 if unspecified

  The eligibility object pointers are located by the pointer to eligibility list entry and contain an entry for each object specified in the number of eligibility objects entry.

Note: The value associated with each entry shown here with an asterisk (*) is ignored by this instruction.

An ND logically represents a physical I/O port or a communications line adapter for a communications network. As such, an ND always has one or more CDs (type 00) that are associated with it through its list of backward objects, which represent the physical devices attached to the I/O port or line.

Addressability to the associated backward objects is established, as appropriate, through the backward object pointers supplied in the ND template. Addressability is also established within these associated objects back to the newly created ND. It is not mandatory that the associated object pointers be supplied in the ND template because as long as the pointers are supplied either in the ND or within the creation templates of the associated objects, proper addressability is established by similar logic within the Create instructions of the other source/sink objects. When the associated object pointers are supplied, the objects must exist and the controller objects or the logical unit objects cannot be associated with another ND. When associated object pointers are not supplied, these pointer locations in the template must contain 16 bytes of binary 0. If the ND template pointer area does not meet previous requirements, an appropriate pointer specification exception is signaled, and the ND is not created.

The switched network backup connection pointer is used only for switched networks and can be materialized to determine the CD or LUD currently connected to this ND. This entry is ignored in the creation template.

The number of backward object pointers entry represents the number of controllers that are attached to this ND if it is a nonswitched line. This number is not supplied at create ND time but is incremented once for each Create Controller Description instruction for controllers attached to this line. A maximum of 10 controllers are allowed on any primary line.

The physical address entry defines the unique address by which the I/O port or communication lines is known internally to the machine. The physical address being supplied within the template must be unique. If not, a source/sink duplicate physical address exception is signaled, and the ND is not created. The physical I/O port or communication lines and its associated machine support components must be installed on the system before the ND can be created. If the internal machine configuration records do not indicate that these physical components are installed, a source/sink resource not available exception is signaled, and the ND is not created.

The state/status definition entry is not used by this instruction. This entry can be materialized (Materialize Network Description instruction) to define the current status of the ND; it can also be modified (Modify Network Description instruction) to change the state of the ND. See the descriptions of those instructions for a complete definition.

The line definition entry is made up of a number of subentries. These subentries are:

- Line discipline–This entry defines the protocol that is used for link level communications. All stations that communicate over the link must follow the same protocol at all times.
  - SDLC (synchronous data link control) establishes the line discipline as synchronous data link control.
  - BSC (binary synchronous communications) establishes the line discipline as binary synchronous communications.
  - X.25 establishes the line discipline as that for packet-switched data networks as per the CCITT (1980) X.25 recommendation.

- Switched network–This entry indicates whether or not the data link is established through the public switched network (0 = no, 1 = yes). If 0 is specified, a nonswitched or private facility is implied.

- Switched network backup–This entry indicates that the modem installed on this communications line is equipped with the switched network backup capability. The normal communications facility is nonswitched. To use this capability, the selectable mode field switched network back operation must be set.

- Data rate select–This entry indicates that the modem on this line has the capability to operate at either a full- or half-speed rate. The rate is selected by setting the appropriate selectable mode. This entry must not be set for BSC lines.

- Role (primary/secondary)–When this entry is set for SDLC, System/38 assumes the role of a secondary station on this line. Otherwise, System/38 assumes the role of a primary station on this line. When this entry is set for BSC, System/38 assumes the role of a BSC multipoint tributary station. Otherwise, point-to-point BSC communication is assumed.

  When this entry is set for X.25, the System/38 appears as DCE to the network. Otherwise, the System/38 appears as DTE to the network.

- NRZI–When this entry is set, System/38 uses the non-return-to-zero (inverted) transmission coding method on this line. This coding method is necessary when interfacing to data communications equipment that does not provide received data timing (internal clock required).

- New sync–This entry is valid only for SDLC primary multipoint lines and should be specified if the data communications equipment requires the new sync interface signal for synchronization of the primary station modem receiver circuitry.

- Nonclocked modem–This entry indicates that the clocking function (receive data timing) for this line is provided by the machine. When 0 is specified, the clocking function is provided by the data communications equipment.

- OEM modem–This entry is set on if non-IBM data communications equipment is installed.

- Wire–This entry indicates the physical line configuration for the modem and the communications channel and also the backup line configuration if switched network backup exists.

- Multipoint network–This entry indicates that the machine is configured as a member of a multipoint network for this line. If not set, it indicates a point-to-point configuration.

- BSC multipoint protocol–This entry indicates that the machine is configured as a BSC multipoint tributary station. If not set, it indicates a BSC point-to-point configuration.

- Autodial–This entry indicates that this switched communication line is equipped with an autocall interface. Any communications lines so equipped require two line positions within the machine so that the next sequential operational unit number cannot be assigned as the physical address of another ND object.

- Autoanswer–This entry indicates that the switched communications line is equipped with a capability to automatically connect incoming calls.

- Autoanswer sequence–This indicator specifies which of two answer sequences is to be used in performing autoanswer functions as determined by the characteristics of the modems being used.

- Answer tone generation–This entry indicates that the machine provides the answer tone signal required by certain modems with the autoanswer capability.

- Transient marks/spaces for answer tone–This indicator specifies whether marks or spaces should be transmitted for performing the answer tone function as required by the modems with the autoanswer capability.

- Special answer tone–This entry indicates whether or not the 38LS integrated modem on this switched line should respond with a 2025 hertz answer tone to the switched line far-end modem.

- DCE (data communications equipment)–This entry indicates the types of modems that can be used on this line. This entry indicates the modem type (an IBM integrated modem or another supported modem) used on a switched line either for the US and Canada or for all other countries.

- Line speed (rate)–This entry indicates the line speed rate in units of 100 bits per second. Permissible values are 12 (1200 bps), 20 (2000 bps), 24 (2400 bps), 48 (4800 bps), 72 (7200 bps), or 96 (9600 bps). If the modem for this communications line has the data rate select capability, this entry should be the full-speed rate.

- Secondary address–This entry contains the link level address to be used by this line when acting in a secondary SDLC station role or when acting as a BSC multipoint tributary station. For X.25, this field contains two 1-byte addresses for the local and remote ends of the HDLC link level attachment to the packet switched data network.

- User assigned identification–This entry, if not 0, causes the machine generated exchange identification data to be overridden by this user supplied identification. This field is only meaningful for SDLC secondary lines, or for APPC support for primary or secondary line configurations. See exchange identification data below.

The communications initialization data entry represents the current set of operating parameters for the communications facility represented by this ND object. This entry can be materialized and is updated by the machine during each activation of the line (Modify ND–Vary On). It is a composite of the characterization of this ND as defined by the line definition, the selectable modes, and the communications subsystem parameters of this ND. It is used only by maintenance personnel for system maintenance.

Exchange identification data–This entry can be materialized and contains the exchange identification used by this System/38 when acting as a station on a network. It is uniquely defined for this system when the system is installed. Normally the machine generates a unique identification value consisting of constant values for the XID characteristics and SNA block number subfields and a binary value derived from the serial number of this machine as recorded in internal configuration records. This field is only meaningful for SDLC lines, or for APPC support for primary or secondary line configurations.

The user assigned identification subfield of the line definition entry in this template can optionally be used to supply an external value in place of the machine serial number. The Create instruction will cause the user assigned identification, if supplied, to overlay the internally generated machine serial value.

Selectable modes–This entry selects modes that can be altered from one line activation to the next.

- Switched network backup (nonswitched/switched)–When this entry is set to 1, the switched network backup capability is in use, and the communications channel exists via the switched network. When this entry is set to 0, the normal nonswitched facility is in use.

- Selected rate (full speed/half speed)–When this entry is set to 1, the transmission speed on this line is one-half that specified in the line speed field. Data rate select modem must be specified in order to run at half speed. When this entry is set to 0, the transmission speed specified in the line speed field is used.

- Character encoding–This field specifies whether the data transmitted across the communication link is to be encoded in EBCDIC or in ASCII.

- BSC protocol mode–This field specifies the BSC protocol options to be used.

- Switched network selections–This field defines the types of switched connection methods that are allowed when the ND is varied on and enabled to the switched enabled state (but does not actually establish the connection). The following types of switched connections can be defined in this field:
  - Allow only incoming calls
  - Allow only outgoing calls
  - Allow incoming and outgoing calls

- Autodial mode–When this entry is set to 1, the switched line connection can be established through the autocall unit. The autodial modem facility must exist for this mode to be valid. When this entry is set to 0, the switched connection is established using manual dial methods.

- Autoanswer mode–When this entry is set to 1, the switched line connection can be established through the autoanswer facilities for incoming calls. The autoanswer modem facility must exist for this mode to be valid.

- Switched secondary line inactivity disconnect–When this communications facility is configured as a secondary station on a switched SDLC network, this indicator causes the switched connection to be disconnected if the communications line is inactive for a period longer than the time indicated by the nonproductive timer. This time is approximately 30 seconds for BSC switched lines. When this entry is set to 0, no disconnect occurs.

The communications subsystem parameters entry describes the communications subsystem parameters.

- Data terminal ready delay–This entry defines the units of time that the machine waits before ending a command that resets the communications line. Each unit of time is 200 milliseconds.

- Delayed contact timer–This timer is used on SDLC primary leased configurations to determine the interval when nonresponding controllers will be recontacted if they are operating with delayed contact control. If multiple controllers are not responding, each will, in turn, be recontacted after the specified interval. Each increment corresponds to 1 second, and 0 to 600 seconds may be specified. If 0 is specified, a default time of 60 seconds will be used by the machine.

- SDLC idle state detection timer–For secondary stations, this entry is ignored. For primary stations, this entry specifies the number of 53.3-millisecond periods that are necessary to satisfy the idle state time considerations for SNA data link control. This time should be greater than the sum of the following conditions:
  - Transmission time to the secondary station
  - Processing time of the control unit's response at the secondary station (not including customer program processing time or operator response time)
  - Clear-to-send time at the secondary station modem
  - Transmission time from the secondary station

  The maximum value allowed is 255, which allows a 13.6-second delay. If a value of 0 is specified, a default value of 500 milliseconds is used. For more information about idle state time considerations, refer to *IBM Synchronous Data Link Control General Information Manual.*

- Nonproductive receive timer–For switched secondary stations, this parameter specifies the number of 500-millisecond periods that are allowed for the line to be inactive. If valid frames of information are not received within this time-out period, the line is disconnected. Normally 30 seconds is adequate, so a value of 60 should be used. The maximum time that can be specified is 127.5 seconds. If 0 is specified, a default time of 128 seconds is used. For primary stations, this entry specifies the number of 500-millisecond periods that are necessary to satisfy the nonproductive receive time considerations. The nonproductive receive timer is dependent upon the data rate (line speed field) specified by the selected rate field. Use the following table to determine, for a given line speed, the recommended value that should be specified for the nonproductive receive timer. The times given in the last column are the resulting maximum times in which to receive intelligible data. They provide enough time for 5250 devices, which can have a maximum number of 266 bytes transmitted per frame.

Line Speed (SDLC Primary Lines Only)	Recommended Parameter Value	Nonproductive Receive Timer Setting (266 Bytes per Frame)
600	11	5.5 seconds
1200	6	3.0 seconds
2400	4	2.0 seconds
4800	2	1.0 seconds
9600	2	1.0 seconds

For more information about the nonproductive receive time considerations, refer to *IBM Synchronous Data Link Control General Information Manual*, GA27-3093.

The pointer to eligibility list defines the CDs that are eligible to be attached to the ND if the ND is used for switched networks. The list contains a set of system pointers that identify the appropriate CDs (type 00 ND). The list is modifiable, but when the ND is created, the list must define the maximum number of entries allowed in the list. Undefined entries are specified by binary 0's. If the switched network protocol does not apply, the pointer to the eligibility list entry contains binary 0's.

The specific characteristics entry defines the set of characteristics that uniquely describe the network. The size and contents of this field are dependent on the specific communications facility being defined.

The retry value sets entry contains values specifying limits for various error types beyond which a higher-level error recovery is invoked.

The line-specific contents entry defines the characteristics that are uniquely described for a specific communication facility. These characteristics can be modified according to the specific communication requirements. The part of this entry that cannot be modified is ignored by a create or modify instruction. The modifiable part of this entry may or may not be required to contain correct data at the time of creation. Additional information about this entry is contained in the *IBM System/38 Functional Reference Manual–Volume 2*, GA21-9800.

Authorization Required

- Privileged instruction

- Insert
  - User profile of creating process

- Operational
  - Source/sink objects identified in operand 2

Lock Enforcement

- Modify
  - User profile that is to own this object
  - Source/sink objects specified as the backward objects identified in operand 2

Events

0002 Authorization
  0101 Object authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded
  0501 Machine address threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0201 Machine context damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
02 Privileged instruction			X
0E Context Operation			
01 Duplicate object identification		X	
10 Damage Encountered			
02 Machine context damage state			X
04 System object damage state		X	X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer address invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
32 Scalar Specification			
01 Scalar type invalid	X	X	
34 Source/Sink Management			
01 Source/sink configuration invalid			X
02 Source/sink duplicate physical address			X
04 Source/sink resource not available			X
38 Template Specification			
01 Template value invalid		X	
02 Template size invalid		X	

DESTROY CONTROLLER DESCRIPTION (DESCD)

Op Code (Hex)	Operand 1
04A1	Controller description

Operand 1: System pointer.

Description: The CD (controller description) specified by operand 1 is destroyed, and addressability to the CD is deleted from the machine context.

Addressability to this CD is also removed from the associated ND (network description) and LUDs (logical unit descriptions). The associated LUDs are rendered unusable because they cannot be varied on or otherwise used for I/O operations until another CD is created to replace this one. The associated LUDs themselves can subsequently be destroyed. The CD destroyed event data contains an indication of whether or not any associated LUDs were encountered during the destroying of this CD.

When the Destroy Controller Description instruction is executed and the CD is not in the varied off state, an exception is signaled, and the CD is not destroyed. If the CD is the only CD attached to an ND, then that ND must also be in the varied off state, or an exception is signaled and the CD is not destroyed.

If the CD is determined to be damaged during destroy processing, then the addressability contained in ND and LUDs to the CD might not be removed. If the state of the CD cannot be determined, the destroy function is completed anyway.

Authorization Required

- Object control
  - Operand 1

Lock Enforcement

- Modify
  - User profile with CD object ownership
  - Network description that is a forward description object for this CD, if any
  - Logical unit descriptions that are backward objects from this CD, if any

- Object control
  - Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
02 Machine context damage state		X
10 Damage Encountered		
04 System object damage state	X	
44 Partial system object damage		X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer address invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	
34 Source/Sink Management		
03 Source/sink object state invalid	X	

DESTROY LOGICAL UNIT DESCRIPTION (DESLUD)

Op Code (Hex)	Operand 1
04A9	Logical unit description

Operand 1: System pointer.

Description: The LUD (logical unit description) specified by operand 1 is destroyed, and addressability to the LUD is deleted from the machine context.

Addressability to this LUD is removed from any associated CD (controller description).

When this instruction is executed and the LUD is not in the varied off state (or powered off state for those devices that can have their power turned off separately), an exception is signaled and the LUD is not destroyed.

If the LUD is determined to be damaged, then addressability to the LUD might not be removed from the associated CD. If the state of the LUD cannot be determined, the destroy function is completed anyway.

Authorization Required

- Object control
  - Operand 1

Lock Enforcement

- Modify
  - User profile with LUD object ownership
  - Controller description which is a forward object for this LUD, if any
- Object control
  - Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
02 Machine context damage state		X
04 System object damage state	X	
44 Partial system object damage		X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
04 Object storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer address invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	
34 Source/Sink Management		
03 Source/sink object state invalid	X	

DESTROY NETWORK DESCRIPTION (DESND)

Op Code (Hex)	Operand 1
04AD	Network description

Operand 1: System pointer.

Description: The ND (network description) specified by operand 1 is destroyed, and addressability to the ND is deleted from the machine context.

Addressability to this ND is also removed from all associated CDs (controller descriptions). These associated CDs cannot be used (they cannot be varied on or otherwise used for I/O operations) until a new ND is created to replace this one. When the ND is not replaced, the CDs and LUDs themselves should be destroyed. The ND destroyed event data contains an indication of whether or not any associated CDs or LUDs were encountered during the destroying of this ND.

When this instruction is executed and the ND is not in the varied off state, an exception is signaled, and the ND is not destroyed.

If the ND is determined to be damaged, then addressability to the associated CD might not be removed. If the state of the ND cannot be determined, the destroy function is completed anyway.

Authorization Required

- Object control
  – Operand 1

Lock Enforcement

- Modify
  – User profile with ND object ownership
  – LUDs that are backward objects for this ND, if any
- Object control
  – Operand 1

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
02 Machine context damage state		X
04 System object damage state	X	
44 Partial system object damage		X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
04 Object storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer address invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	
34 Source/Sink Management		
03 Source/sink object state invalid	X	

MATERIALIZE CONTROLLER DESCRIPTION (MATCD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04B3	Receiver template	Controller description	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(2) scalar (fixed-length).

Description: Based on the materialization options specified by operand 3, elements of the CD (controller description) object specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes of the template size specification entry contain a value that specifies the number of bytes that can be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the template size specification entry contain a value that specifies the number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exception (other than the materialization length exception) is signaled in the event that the receiver contains insufficient area for the materialization.

The template identified by operand 1 must be 16-byte aligned.

Authorization is not set in materialized system pointers.

The scalar specified in operand 3 cannot be defined by a data pointer.

The following charts show the elements within the CD materialization templates and the corresponding materialization option values that are used to select these elements. The materialization option value specified for operand 3 must contain a value as follows:

Hex 8000–Causes a materialization of the entire contents of the CD as shown within the following chart.

Hex znnn–Causes one of the following for z = 1 or z = 4:

Hex 1nnn–Causes a materialization of only the individual element within the CD that has the corresponding value of nnn.

Hex 4nnn–Causes a materialization of any members of the set of elements within the CD that are modifiable elements. The nnn value in operand 3 is formed by a logical OR of the individual nnn option values for the desired elements as shown in the following charts.

Elements Contained in the CD Template (CD Types 00 and 10) for Materialize CD			**Element Length**	**Sub-Element Length**	**Materialize Option Values**
Template size specification			Char(8)		
Reserved (for all materialize templates except ones including object header data)			Char(8)		
Object header data (includes template size)			Char(96)		1003
CD definition data			Char(16)		1007
Forward object group			Char(32)		1005
Backward pointer list data			Char(32)		1006
Physical definition data			Char(16)		1009
State/status definition			Char(16)		z001
• State change/status				Char(6)	
Byte(s)	*Bit(s)*	*Meaning*			
0-1		**Status CD Session Count (number of LUDs in session)**		Bin(2)	
2-3		**Status CD Active Count (number of LUDs varied on)**		Bin(2)	
4		**Status**			
	0	CD active, LUD(s) in session		Bit 1	
	1	CD active, LUD(s) varied on		Bit 1	
	2	Varied on state		Bit 1	
	3	Dialing out state		Bit 1	
	4	Vary on pending and LUD(s) in vary on pending state		Bit 1	
	5	Vary on pending state		Bit 1	
	6	Reserved		Bit 1	
	7	Power on/vary off state		Bit 1	
5		**Status**			
	0	Power off state		Bit 1	
	1-3	Reserved		Bits 3	
	4	Diagnostic mode		Bit 1	
	5	Diagnostic active indicator		Bit 1	
	6-7	Reserved		Bits 2	
• Recovery/resource activation				Char(2)	
Byte(s)	*Bit(s)*	*Meaning*			
0		**Status**			
	0	Inoperative pending		Bit 1	
	1	Normal pending		Bit 1	
	2	Normal cancel		Bit 1	
	3	Normal continue		Bit 1	
	4	Normal activation pending		Bit 1	
	5-6	Reserved		Bits 2	
	7	Normal active		Bit 1	
1		Reserved		Char(1)	

Elements Contained in the CD Template (CD Types 00 and 10) for Materialize CD	Element Length	Sub-Element Length	Materialize Option Values
• Reserved		Char(8)	
ND candidate list data	Char(32)		z002
Station control information	Char(32)		100D
Selected mode data	Char(16)		z004
Activate physical unit information	Char(16)		100E
Dial digits	Char(32)		z008
Specific characteristics	Char (y + 2)		100F
XID information area	Char (y + 2)		1011
Unit-specific contents	Char (y + 4)		z010
Backward object list	Variable number of system pointers	System pointers to LUDs	1006
ND candidate list	Variable number of system pointers	System pointers to NDs or null	z002

y = Variable length of an element
z = Option value control digit. Valid values are:
z = 1 Materialize this individual element.
z = 4 Materialize this element along with any other elements of the modifiable set by ORing together their option values.

The CD session count field indicates the number of LUDs that are attached to this CD because of Modify LUD (activate session) instructions and for which Modify LUD (de-activate session) instructions have not been issued. This is a count of the number of LUDs that are currently in any one of the LUD session states.

The CD active count field indicates the number of LUDs that are attached to this CD because of Modify LUD (vary on) instructions and for which Modify LUD (vary off) instructions have not been issued. This is a count of the number of LUDs that are currently in vary on pending, varied on/no session, or any session state.

The recovery/resource activation field can be materialized and is set by the machine when conditions that require error recovery action are encountered by this object or any forward object that this object is attached to.

Refer to the Modify Controller Description instruction for details of the states in the CD object that can be materialized and for the corresponding modify operations to these states.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Operational
  - Operand 2

Lock Enforcement

- Materialize
  - Operand 2

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X		
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state		X		
44 Partial system object damage		X		X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer address invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	
34 Source/Sink Management				
01 Source/sink configuration invalid				X
38 Template Specification				
03 Materialization length exception	X			

MATERIALIZE LOGICAL UNIT DESCRIPTION (MATLUD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04BB	Receiver template	Logical unit description	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(2) scalar (fixed-length).

Description: Based on the materialization options specified by operand 3, elements of the LUD (logical unit description) object specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes of the template size specification entry contain a value that specifies the number of bytes that can be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the template size specification entry contain a value that specifies the number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exception (other than the materialization length exception) is signaled in the event that the receiver contains insufficient area for the materialization.

The template identified by operand 1 must be 16-byte aligned.

Authorization is not set in materialized system pointers.

The scalar specified in operand 3 cannot be defined by a data pointer.

The following charts show the elements within the LUD materialization templates and the corresponding materialization option values that are used to select these elements. The materialization option value specified for operand 3 must contain a value as follows:

Hex 8000–Causes a materialization of the entire contents of the LUD as shown in the following chart.

Hex znnn–Causes one of the following for z = 1 or z = 4:

Hex 1nnn–Causes a materialization of only the individual element within the LUD that has the corresponding value for nnn.

Hex 4nnn–Causes a materialization of any members of the set of elements within the LUD that are modifiable elements. The nnn value in operand 3 is formed by a logical OR of the individual nnn option values for the desired individual elements as shown in the following charts.

Elements Contained in the Template (LUD Types 00, 10, 30) for Materialize LUD	Element Length	Sub-Element Length	Materialize Option Values
Template size specification	Char(8)		
Reserved (for all materialize templates except ones including object header data)	Char(8)		
Object header data (includes template size)	Char(96)		1003
LUD definition data	Char(16)		1007
Pointer group data	Char(16)		1005
Physical definition data	Char(16)		1009
State/status definition	Char(16)		z001
• State change/status		Char(6)	
Byte(s) *Bit(s)* *Meaning*			
0 **Status**			
0-6 Reserved		Bits 7	
7 Active session state		Bit 1	
1 **Status**			
0 Suspended session state		Bit 1	
1 Quiesced session state		Bit 1	
2 Reset session state		Bit 1	
3 Varied on/no session state		Bit 1	
4 Vary on pending state		Bit 1	
5 Reserved		Bit 1	
6 Power on/vary off state		Bit 1	
7 Power off state		Bit 1	
2 **Status**			
0 Diagnostic mode		Bit 1	
1 Diagnostic active indicator		Bit 1	
2-7 Reserved		Bits 6	
3-5 **Reserved**		Char(3)	
• Recovery/resource activation		Char(2)	
Byte(s) *Bit(s)* *Meaning*			
0 **Status**			
0 Inoperative pending		Bit 1	
1 Normal pending		Bit 1	
2 Normal cancel		Bit 1	
3 Normal continue		Bit 1	
4 Normal activation pending		Bit 1	
5-6 Reserved		Bits 2	
7 Normal active		Bit 1	
1 Reserved		Char(1)	
• Reserved		Char(8)	
Session definition data	Char(32)		z002
Load/dump definition data	Char(16)		z004

Elements Contained in the Template (LUD Types 00, 10, 30) for Materialize LUD	Element Length	Sub-Element Length	Materialize Option Values
Specific characteristics	Char (y + 2)		1012
Retry value sets	Char (6y + 2)		z008
Error threshold sets	Char (8y + 2)		z010
Device-specific contents	Char (y + 4)		z020

y = Variable length of an element
z = Option value control digit. Valid values are:
z = 1 Materialize this individual element.
z = 4 Materialize this element as part of a group of modifiable elements.

The recovery/resource activation field can be materialized and is set by the machine when conditions that require error recovery action are encountered by this object or any forward object that this object is attached to.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Refer to the Modify Logical Unit Description instruction for details of the states in the LUD object that can be materialized and for the corresponding modify operations to these states.

Authorization Required

- Operational
  - Operand 2

Lock Enforcement

- Materialize
  - Operand 2

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

Exception	Operands			Other
	1	2	3	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X		
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state		X		
44 Partial system object damage		X		
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer address invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MATERIALIZE NETWORK DESCRIPTION (MATND)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04BF	Receiver template	Network description	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(2) scalar (fixed-length).

Description: Based on the materialization options specified by operand 3, elements of the ND (network description) object specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes of the template size specification entry contain a value that specifies the number of bytes that can be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the template size specification entry contain a value that specifies the number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exception (other than the materialization length exception) is signaled in the event that the receiver contains insufficient area for materialization.

The template identified by operand 1 must be 16-byte aligned.

Authorization is not set in materialized system pointers.

The scalar specified in operand 3 cannot be defined by a data pointer.

The following chart shows the elements within the ND materialization templates and the corresponding materialization option values that are used to select these elements. The materialization option value specified for operand 3 must contain a value as follows:

- Hex 8000–Causes a materialization of the entire contents of the ND as shown in the following chart.

- Hex znnn–Causes one of the following for z = 1 or z = 4:

  - Hex 1nnn–Causes a materialization of only the individual element within the ND that has the corresponding value for nnn.

  - Hex 4nnn–Causes a materialization of any members of the set of elements within the ND that are modifiable elements. The nnn value in operand 3 is formed by a logical OR of the individual nnn option values for the desired elements as shown in the following charts.

Elements Contained in the ND Template (ND Type 00) for Materialize ND	**Element Length**	**Sub-Element Length**	**Materialize Option Values**
Template size specification	Char(8)		
Reserved (for all materialize templates except ones including object header data)	Char(8)		
Object header data (includes template size)	Char(96)		1003
ND definition data	Char(16)		1007
Backward object pointer group	Char(48)		1006
Physical definition data	Char(16)		1009
State/status definition	Char(16)		z001
• State change/status		Char(6)	
Byte(s) *Bit(s)* *Meaning*			
0-1 **Status ND active count (number of CDs varied on)**		Bin(2)	
2 **Status**			
0 Network active		Bit 1	
1 Manual dial start state		Bit 1	
2 Manual answer start state		Bit 1	
3 Manual answer state		Bit 1	
4 Dial pending state		Bit 1	
5 Switched enabled state		Bit 1	
6 Varied on state		Bit 1	
7 Varied off state		Bit 1	
3 **Status**			
0 Diagnostic mode		Bit 1	
1 Diagnostic active indicator		Bit 1	
2-7 Reserved		Bits 6	
4-5 **Reserved**		Char(2)	
• Recovery/resource activation		Char(2)	
Byte(s) *Bit(s)* *Meaning*			
0 **Status**			
0 Inoperative pending		Bit 1	
1 Reserved		Bit 1	
2 Normal cancel		Bit 1	
3 Normal continue		Bit 1	
4-6 Reserved		Bits 3	
7 Normal active		Bit 1	
1 Reserved		Char(1)	
• Reserved		Char(8)	
Line definition data	Char(16)		100A
Communications initialization data	Char(16)		100B
Exchange identification data	Char(16)		100C
Selectable mode data	Char(16)		z002
Communications subsystem parameters data	Char(16)		z004

Elements Contained in the ND Template (ND Type 00) for Materialize ND	Element Length	Sub-Element Length	Materialize Option Values
Reserved group	Char(32)		z010
Specific characteristics	Char (y + 2)		100D
Retry value sets	Char (6y + 2)		z020
Line-specific contents	Char (y + 4)		z040
Backward object pointers	Variable number of system pointers	System pointers to CD or to LUD	1006

y = Variable length of an element
z = Option value control digit. Valid values are:
z = 1 Materialize this individual element.
z = 4 Materialize this element along with any other elements of the modifiable set by ORing together their option values.

The ND active count field indicates the number of CD objects associated with this ND that caused the ND to be in the network active state. For switched lines (point-to-point), this indicator can have values of only 0 or 1. The count will be incremented to 1 only when a switched connection exists and will be decremented to 0 when the switched connection is disconnected.

For nonswitched lines, this indicator can have any value from 0 to the maximum number of attached controllers. It indicates a count of the number of attached controllers for which a Modify CD (vary on) instruction was issued and a Modify CD (vary off) instruction was not. It is a count of the number of CDs which are in the vary on pending or the varied on states.

The recovery/resource activation field can be materialized and is set by the machine when conditions that require error recovery action are encountered by this object.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Refer to the Modify Network Description instruction for details of the states in the ND object that can be materialized as shown above and for the corresponding modify operations to these states.

Authorization Required

- Operational
  - Operand 2

Lock Enforcement

- Materialize
  - Operand 2

Events

0002 Authorization
0101 Object authorization violation

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X		
0A Authorization				
01 Unauthorized for operation		X		
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state		X		
44 Partial system object damage		X		X
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer address invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	
34 Source/Sink Management				
01 Source/sink configuration invalid				X
38 Template Specification				
03 Materialization length exception	X			

MODIFY CONTROLLER DESCRIPTION (MODCD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04C3	Controller description	Controller description modification template	Modification options

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: Character(2) scalar (fixed-length).

Description: This instruction modifies the CD (controller description) specified by operand 1 to the new values contained in the modification template specified by operand 2. The elements or groups of elements within the CD are modified based on the modification options specified by operand 3.

The scalar specified in operand 3 cannot be defined by a data pointer.

The template identified by operand 2 and any pointer list referenced by it must be 16-byte aligned.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

The following chart shows the modifiable elements that can be included in the template for operand 2. (Refer to the Create Controller Description instruction for detailed descriptions of the elements). The template can contain any combination of these elements as indicated by the option value in operand 3, by including only those elements in the order shown here.

The Set Diagnostic Mode and Reset Diagnostic Mode commands are for use by service personnel.

Elements Contained in the CD Template (CD Types 00 and 10) for Modify CD	Element Length	Sub-Element Length	Materialize Option Values
Template size specification	Char(8)		
Modify time-out value (for all modify templates)	Char(8)		
State/status definition	Char(16)		4001
• State change/status		Char(6)	
Byte(s) *Bit(s)* *Meaning*			
0-1 **Reserved**		Bin(2)	
2-3 **Reserved**		Bin(2)	
4 **Commands**			
0-1 Reserved		Bits 2	
2 Dial		Bit 1	
3 Abandon connection		Bit 1	
4 Reserved		Bit 1	
5 Vary on		Bit 1	
6 Vary off		Bit 1	
7 Power on		Bit 1	
5 **Commands**			
0 Power off		Bit 1	
1-3 Reserved		Bits 3	
4 Set diagnostic mode		Bit 1	
5 Reset diagnostic mode		Bit 1	
6-7 Reserved		Bits 2	
• Recovery/resource activation		Char(2)	
Byte(s) *Bit(s)* *Meaning*			
0 **Status**			
0-1 Reserved		Bits 2	
2 Normal cancel		Bit 1	
3 Normal continue		Bit 1	
4-7 Reserved		Bits 4	
1 Reserved		Char(1)	
• Reserved		Char(8)	
ND candidate list data	Char(32)		4002
Selected mode data	Char(16)		4004
Dial digits	Char(32)		4008
Unit-specific contents	Char (y + 4)		4010
ND candidate list	Variable number of system pointers	Either system pointers to NDs or null	4002

y = Variable length of an element

Note: A combination of elements can be modified on the same Modify instruction by supplying in operand 3 a value that is the result of performing a logical OR on the modify option values of the desired elements.

Each modifiable element within a CD can be successfully modified only when in certain operational states of the controller description.

Refer to Figures 17-1 through 17-4 for a description of the states, the status, and the elements of the CD that can be modified.

When the state of the CD does not allow the modification of a requested element, a source/sink object state invalid exception is signaled and modification is stopped. All elements that were modified before the exception remain successfully modified. The exception information identifies the element responsible for the exception.

Any modification options that invoke other changes to the object, along with changes to the recovery/resource activation field, cause a template value invalid exception to be signaled.

Modification options that include the state change/status element of the CD and involve a vary state change to this element have the following additional exceptions that can be signaled if conditions are not valid for the requested change:

- Source/sink object state invalid–This exception occurs because an associated ND (network description) or LUD (logical unit description) is not in the proper state for this controller to be varied on or varied off.

- Source/sink configuration invalid–This exception occurs because the CD does not have a required valid forward object pointer; therefore, the controller cannot be associated with any communications line or I/O port.

- Source/sink resource not available–This exception occurs because the appropriate physical hardware or machine support components are not installed on the system to match this CD. This exception can also occur because of a hardware failure occurring anywhere in the communications network while the system is attempting to establish the vary on function for the CD.

The following describes the vary on function and the effect it has on the CD object. The CD for a particular controller must be explicitly varied on by using the Modify Controller Description instruction. If the CD is attached to an ND (network description), the ND object must be varied on before varying on the CD. If the CD (controller description) has logical unit descriptions (LUDs) attached, the CD must be varied on before varying on any of the LUDs (this check is made in the Modify LUD instruction). However, the LUDs must be varied off before varying off the CD. If the above conditions are not met, a source/sink object state invalid exception is signaled, and the instruction is stopped at that point.

Whether the CD is logically or physically varied on depends on the attachment method used for this controller.

The following describe the different attachment methods and the resulting state of the CD object:

If the CD is type 00 (CD is not attached to an ND), the following conditions apply:

1. The physical connection is activated, and initial contact with the station is established. (If contact cannot be established, a resource not available exception is signaled, and the instruction is stopped at that point.)

2. The CD object is set to a vary on state.

3. A CD contact event is signaled.

If the CD is type 10 (CD is attached to an ND) and the CD represents a station on a nonswitched line, the following conditions apply:

1. The CD object is set to a vary on pending state and the Modify CD instruction is completed. (The remaining activity is performed asynchronously by the machine.)

2. If the CD object indicates that delayed contact control is not present and the station cannot be contacted, a CD contact event (unsuccessful) is signaled, and the CD remains in a vary on pending state.

3. If the CD object indicates that delayed contact control is present and the station can be contacted, the CD contact event (successful) is signaled, and the CD is modified to varied on state.

4. If the CD object indicates delayed contact control is present and the station cannot be contacted, the CD object remains in a vary on pending state and periodic attempts to contact the station continue until contact is established (CD goes to a vary on state) or the CD is varied off. The CD contact event is signaled when the station has been contacted, and vary on is completed.

If the attachment method indicates that the CD is attached to an ND and the CD represents a controller that supports communications via the switched network, the forward switched connection pointer in the CD does not contain the address of an ND object. The forward switched connection pointer is set to null (binary 0) when the CD object is created, when the CD is varied off, or when the Modify Controller Description Abandon Connection command sets the CD to a vary on pending state. When the CD is set to a vary on pending state, the Modify Controller Description instruction completes execution. The following describes how the CD goes from a vary on pending state to a vary on state and also how the forward address in the CD is set to address an ND object:

When a CD for a nonswitched network is being varied on and the resource is not available because of a network failure, the CD status is set to the same status as the ND object.

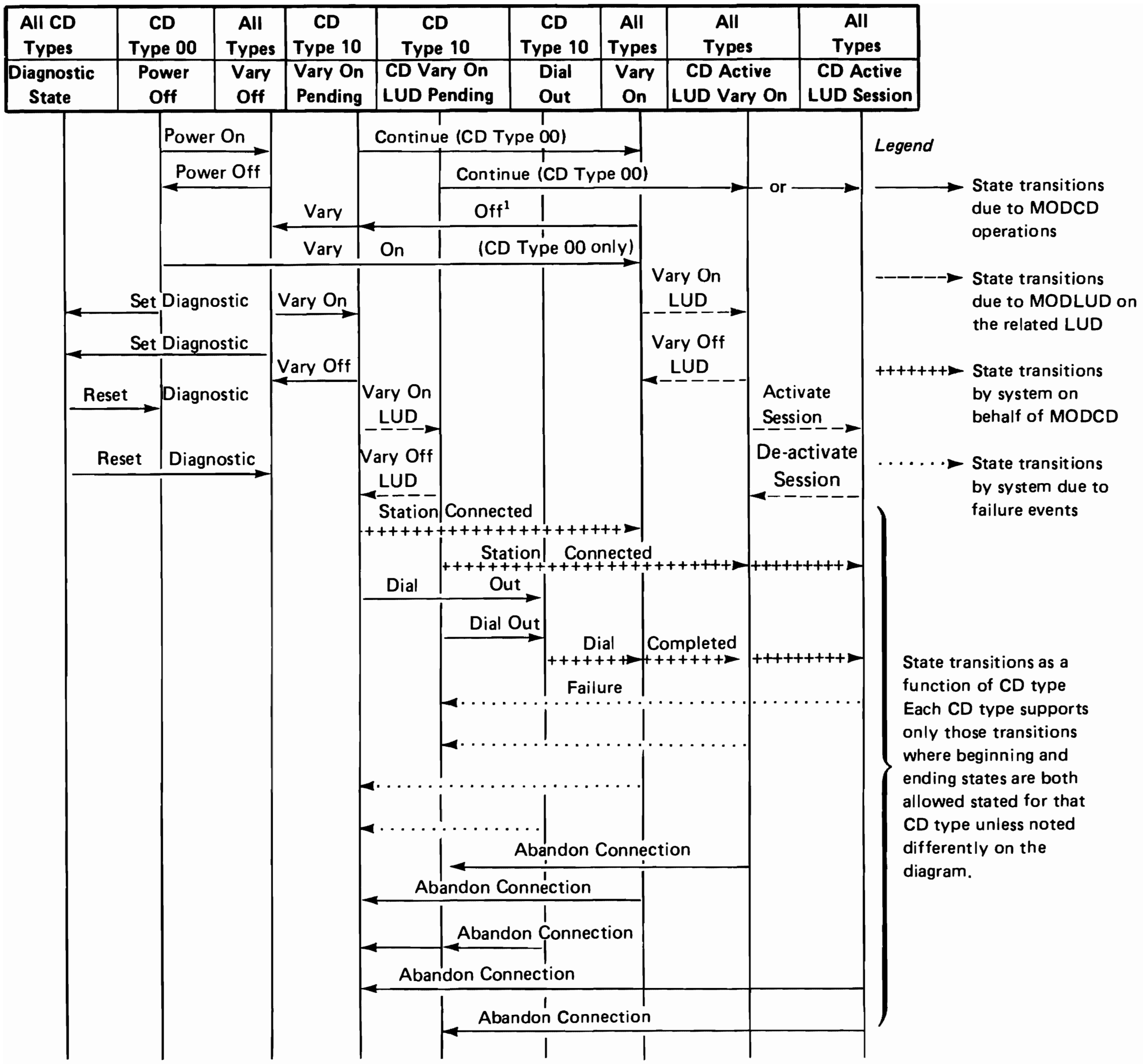
Dial In

For dial in devices, the vary on pending state exists until an activated line attachment accepts an incoming call. If the ND associated with the line is in the ND candidate list of this CD for SDLC lines or if this CD is in the eligibility list of the ND for BSC lines, the forward switched connection pointer in the CD is set to point to the ND object and the CD is set to varied on state. (If the specified ND is not in the list, the connection is abandoned, and an event is signaled.) The address of the CD object is put in the backward switched connection of the ND object, and the CD object is set to a vary on state. Any LUDs attached to this CD that are in a vary on pending state are set to a vary on state at this time. The CD contact event and LUD contact event(s) are signaled upon completion of the activity associated with the incoming call.

Dial Out

For dial out devices, the system initiates a dial procedure to establish a switched connection at the time a Modify CD Dial Connection command is issued to the CD. To complete a dial out connection, the ND candidate list in the CD is again referenced. An ND that is in the switched enabled state with continue status must be found in the list, must be enabled for dial out, and must not be in use. If an ND is not found, the connection is not made and a resource not available exception is signaled. Once an ND is found, the CD is updated to dial pending state, the switched connection forward pointer is set to point to the ND that was selected, the ND status is updated to dial pending state; the backward connection pointer in the ND is set to point to this CD; and the instruction completes execution. The actual connecting of the line is done asynchronously by the machine. A manual intervention event can be signaled during this interval if the connection requires manual dialing. When the connection is made, the CD goes to a vary on state, and the ND goes to the active state. Any LUDs attached to this CD that are in a vary on pending state are set to a vary on state at this time. The CD contact event and LUD contact event(s) are signaled upon completion of this dial out activity.

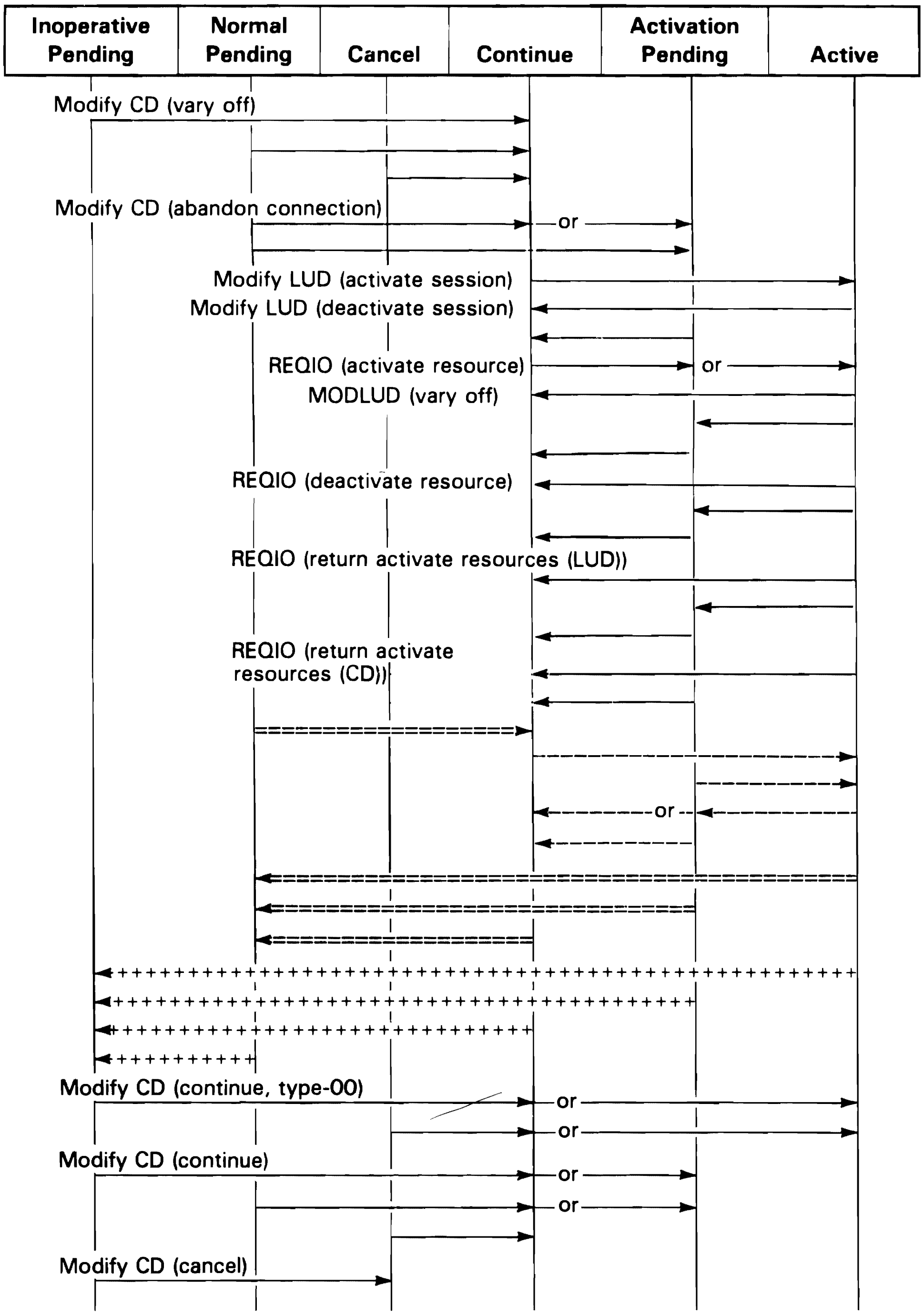
The modify time-out value field is used to specify the desired length of time (in standard time units) that the machine should allow for the modification operation to complete. The minimum time-out value is 10 seconds, and the maximum time-out value is 45 minutes. If the operation does not complete within the specified time, the operation is terminated and the partial system object damage exception is signaled. Error recovery procedures must be invoked to perform any shutdown or cleanup operations if this exception occurs. If no time-out value is specified in the modify template, a default time-out value of 85 seconds is used. Any nonzero time-out value supplied must fall within the time-out limits. This time-out value should not be construed as a maximum length of execution time for the modify instruction. The time-out is only used internally to time some arbitrary portion of the operation to prevent the Modify instruction from never completing. Time-out does not occur in less than the specified time-out value. However, execution may validly be much longer than the time-out value when several elements are included in one Modify instruction because each element operation is timed separately.


All CD Types Diagnostic State
CD Type 00 Power Off
All Types Vary Off
CD Type 10 Vary On Pending
CD Type 10 CD Vary On LUD Pending
CD Type 10 Dial Out
All Types Vary On
All Types CD Active LUD Vary On
All Types CD Active LUD Session
Power On
Power Off
Continue (CD Type 00)
Continue (CD Type 00)
or
Vary
Off¹
Vary
On
(CD Type 00 only)
Vary On LUD
Set Diagnostic
Vary On
Set Diagnostic
Vary Off LUD
Vary Off
Reset
Diagnostic
Vary On LUD
Activate Session
Reset
Diagnostic
Vary Off LUD
De-activate Session
Station Connected
Station Connected
Dial
Out
Dial Out
Dial
Completed
Failure
Abandon Connection
Abandon Connection
Abandon Connection
Abandon Connection
Abandon Connection
Legend
State transitions due to MODCD operations
State transitions due to MODLUD on the related LUD
State transitions by system on behalf of MODCD
State transitions by system due to failure events
State transitions as a function of CD type Each CD type supports only those transitions where beginning and ending states are both allowed stated for that CD type unless noted differently on the diagram.

[1] For CD type 10 (switched line), a vary off from this state will cause an implicit abandon connection and then a vary off.

Modify Option Values	CD States	Diagnostic State	Power Off	Vary Off	Vary On Pend-ing	CD Vary On Pending LUD Vary On Pending	Dial Out	Vary On	CD Active LUD Vary On	CD Active LUD Session
	Element checking sequence and allowable states for modification									
4002	1. ND candidate list	No	Yes	Yes	Yes	Yes	No	No	No	No
4004	2. Selected modes	No	Yes	Yes	No	No	No	No	No	No
4008	3. Dial digits	No	Yes	Yes	Yes	Yes	No	No	No	No
4010	4. Unit-specific content	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Figure 17-1. CD State Change Rules


Inoperative Pending
Normal Pending
Cancel
Continue
Activation Pending
Active
Modify CD (vary off)
Modify CD (abandon connection)
or
Modify LUD (activate session)
Modify LUD (deactivate session)
REQIO (activate resource)
or
MODLUD (vary off)
REQIO (deactivate resource)
REQIO (return activate resources (LUD))
REQIO (return activate resources (CD))
or
Modify CD (continue, type-00)
or
or
Modify CD (continue)
or
or
Modify CD (cancel)

Legend

——► or ◄—— Status transition caused by successful completion of a source/sink Modify or a synchronous Request I/O instruction.

- - - - - - - - -►
◄- - - - - - - - -
◄==========
==========►

Status transition asynchronous to a source/sink instruction.

◄++++++++++ Status transition asynchronous to a source/sink instruction that is caused by a failure.

Figure 17-2. CD Object Recovery/Resource Activation Status Transitions

Modify Instruction	Diagnostic State	Power Off	Power On Varied Off	CD Vary On Pending No LUDs Vary On Pending	CD Vary On Pending LUDs Vary On Pending	Dial Out	Vary On	CD Active LUD Vary On	CD Active LUD Session
Vary On	No	No	Yes	No	No	No	No	No	No
LUD Vary On	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
LUD Activate Session	No	No	No	No	Yes[1]	Yes[1]	No	Yes	Yes
Dial	No	No	No	Yes	Yes	No	No	No	No
Abandon Connection	No	No	No	No	No	Yes	Yes	Yes	Yes
LUD Vary Off	No	No	No	No	Yes	Yes	No	Yes	Yes
LUD Deactivate Session	No	No	No	No	Yes	Yes	No	No	Yes
Vary Off	No	No	No	Yes	No	No	Yes	No	No
Cancel	No	No	No	Yes	Yes	No	No	No	No
Continue	No	No	No	Yes	Yes	No	Yes[1]	Yes[1]	Yes[1]

[1] These states are conditionally supported. For the specific implementation, see *IBM System/38 Functional Reference Manual–Volume 2,* GA21-9800.

Figure 17-3. CD State Modification Rules

Modify Instruction	Inoperative Pending	Normal Pending	Cancel	Continue	Activation Pending	Active
Vary On	No	No	No	Yes	No	No
LUD Vary On	Yes	Yes	Yes	Yes	Yes	Yes
LUD Activate Session	Yes[1]	Yes[1]	Yes[1]	Yes	Yes	Yes
Dial	No	No	No	Yes	Yes	No
Abandon Connection	No	Yes	No	Yes	Yes	No
LUD Vary Off	Yes	Yes	Yes	Yes	Yes	Yes
LUD Deactivate Session	Yes	Yes	Yes	Yes	Yes	Yes
Vary Off	Yes	Yes	Yes	Yes	No	No
Cancel	Yes	No	No	No	No	No
Continue	Yes	Yes[1]	Yes	No	Yes[1]	No

[1] These states are conditionally supported. For the specific implementation, see *IBM System/38 Functional Reference Manual–Volume 2,* GA21-9800.

Figure 17-4. CD Status Modification Rules

Authorization Required

- Operational
  - Operand 1
  - System objects specified within the operand 2 space object, if any (ND candidate list entries)

Lock Enforcement

- Modify
  - Operand 1
  - The ND, which is specified by the forward object pointer of this CD, if any, and only when this forward object is to be modified by the synchronous execution of this Modify CD instruction on the status field of the CD object
  - The LUDs that are specified by the backward object pointer list in this CD, and only when this backward object is to be modified by the synchronous execution of the Modify CD instruction on the status field of the CD object

Note: The state change diagrams provided with the Modify Logical Unit Description and the Modify Network Description instructions show when the Modify Controller Description instruction causes these modifications. For operations that involve the recovery/resource activation status field, objects are not ensured by object locks and the status fields may be changed independently by the machine or other processes regardless of the lock state of the object.

Events

0002 Authorization
    0101 Object authorization violation

0004 Controller description
    0401 Controller description successful contact
    0402 Controller description invalid contact
    0403 Controller description unsuccessful contact
    0601 Controller description manual intervention

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X		
0A Authorization				
01 Unauthorized for operation	X			
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state	X	X	X	X
44 Partial system object damage	X			
1A Lock State				
01 Invalid lock state				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	
34 Source/Sink Management				
01 Source/sink configuration invalid				X
03 Source/sink object state invalid	X			X
04 Source/sink resource not available				X
38 Template Specification				
01 Template value invalid		X		
02 Template size invalid		X		

MODIFY LOGICAL UNIT DESCRIPTION (MODLUD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04CB	Logical unit description	Logical unit description modification template	Modification options

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: Character(2) scalar (fixed-length).

Description: This instruction modifies the LUD (logical unit description) specified by operand 1 to the new values contained in the modifications template specified by operand 2. The elements or groups of elements within the LUD are modified based on the modification options specified by operand 3.

Operand 2 must be 16-byte aligned. The scalar specified in operand 3 cannot be defined by a data pointer.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

The following chart shows the modifiable elements that can be included in the template for operand 2. (Refer to the Create Logical Unit Description instruction for detailed descriptions of the elements.) The template can contain any combination of these elements as indicated by the option value in operand 3, by including only those elements in the order shown here.

The Set Diagnostic Mode and Reset Diagnostic Mode commands are for use by service personnel.

Elements Contained in the LUD Template (LUD Types 00, 10, 30) for Modify LUD			**Element Length**	**Sub-Element Length**	**Modify Option Values**
Template size specification			Char(8)		
Modify time-out value			Char(8)		
State/status definition			Char(16)		4001
• State change/status				Char(6)	
Byte(s)	*Bit(s)*	*Meaning*			
0		**Commands**			
	0-6	Reserved		Bits 7	
	7	Activate session		Bit 1	
1		**Commands**			
	0	Suspend session		Bit 1	
	1	Quiesce session		Bit 1	
	2	Reset session		Bit 1	
	3	De-activate session		Bit 1	
	4	Vary on		Bit 1	
	5	Vary off		Bit 1	
	6	Power on		Bit 1	
	7	Power off		Bit 1	
2		**Commands**			
	0	Set diagnostic mode		Bit 1	
	1	Reset diagnostic mode		Bit 1	
	2-7	Reserved		Bits 6	
3-5		**Reserved**		Char(3)	
• Recovery/resource activation				Char(2)	
Byte(s)	*Bit(s)*	*Meaning*			
0		**Status**			
	0-1	Reserved		Bits 2	
	2	Normal cancel		Bit 1	
	3	Normal continue		Bit 1	
	4-7	Reserved		Bits 4	
1		Reserved		Char(1)	
• Reserved				Char(8)	
Session definition data			Char(32)		4002
Load/dump definition data			Char(16)		4004
Retry value sets			Char (6y + 2)		4008
Error threshold sets			Char (8y + 2)		4010
Device-specific contents			Char (y + 4)		4020

y = Variable length of an element

Note: A combination of elements can be modified on the same Modify instruction by supplying in operand 3 a value that is the result of performing a logical OR on the modify option values of the desired elements.

Each modifiable element within an LUD can be successfully modified only when in certain operational states of the LUD.

Refer to Figures 17-5 through 17-10 for a description of the states that exist for the LUD object, for the valid state changes that can be made by the Modify Logical Unit Description instruction, and for the valid relationship for modifying elements in the LUD.

When the state of the LUD does not allow the modification of a requested element, a source/sink object state invalid exception is signaled and modification is stopped. All elements that were modified before the exception remain successfully modified. The exception information identifies the element responsible for the exception.

Any modification options that invoke other changes to the object, along with changes to the recovery/resource activation field, cause a template value invalid exception to be signaled.

Modification options that include the state change/status element of the LUD and involve a power state, a vary state, or an activate session change to this element have the following additional exceptions that can be signaled if conditions are not valid for the requested change:

- Source/sink object state invalid–This exception occurs because the associated controller description or network description is not in the proper state for the logical unit to be varied on. This exception also occurs when the logical unit description itself is not in the proper state to allow a power on, power off, activate session, de-activate session, suspend, quiesce, or reset modification.

- Source/sink configuration invalid–This exception occurs because the LUD does not have a valid forward object pointer; therefore, the logical unit cannot be associated with any control unit or communications line as part of a vary on modification.

- Source/sink resource not available–This exception occurs because the appropriate hardware or machine support components are not installed on the system to match this LUD. This exception can also occur because of a hardware failure occurring anywhere in the system while the system is attempting to establish a power on, vary on, or activate session function for the LUD.

The LUD for a particular device must be explicitly varied on by using the Modify LUD instruction. If the LUD is attached to a CD (controller description), the CD must be in the varied on or the vary on pending state before varying on the LUD. If not, a source/sink object state invalid exception is signaled, and execution of the instruction stops.

Whether the LUD is logically or physically varied on depends on the attachment method used for this device. The following paragraphs describe the different attachment methods and the resulting state of the LUD object.

When the attachment method indicates that the LUD can be attached directly (LUD type 00) or attached only to a CD (LUD type 10), the device is initialized, and the LUD object is set to the vary on state. A LUD contact event is also signaled. If the device cannot be initialized, a resource not available exception is signaled, and execution of the instruction is stopped.

When the attachment method indicates that the LUD is attached to both a CD and an ND (LUD type 30) and the CD represents a station on a nonswitched line or local loop, the device is initialized, and the LUD is modified either to a vary on state if the CD was in a varied on state or to a vary on pending state if the CD was in a vary on pending state. If the device cannot be initialized, the resource not available exception is signaled, and execution of the instruction is stopped. If the LUD is modified to a vary on pending state, asynchronous to this Modify LUD instruction, the LUD is then modified to a varied on state; the LUD contact event is signaled whenever contact is made with the station, and the CD is modified to a varied on state.

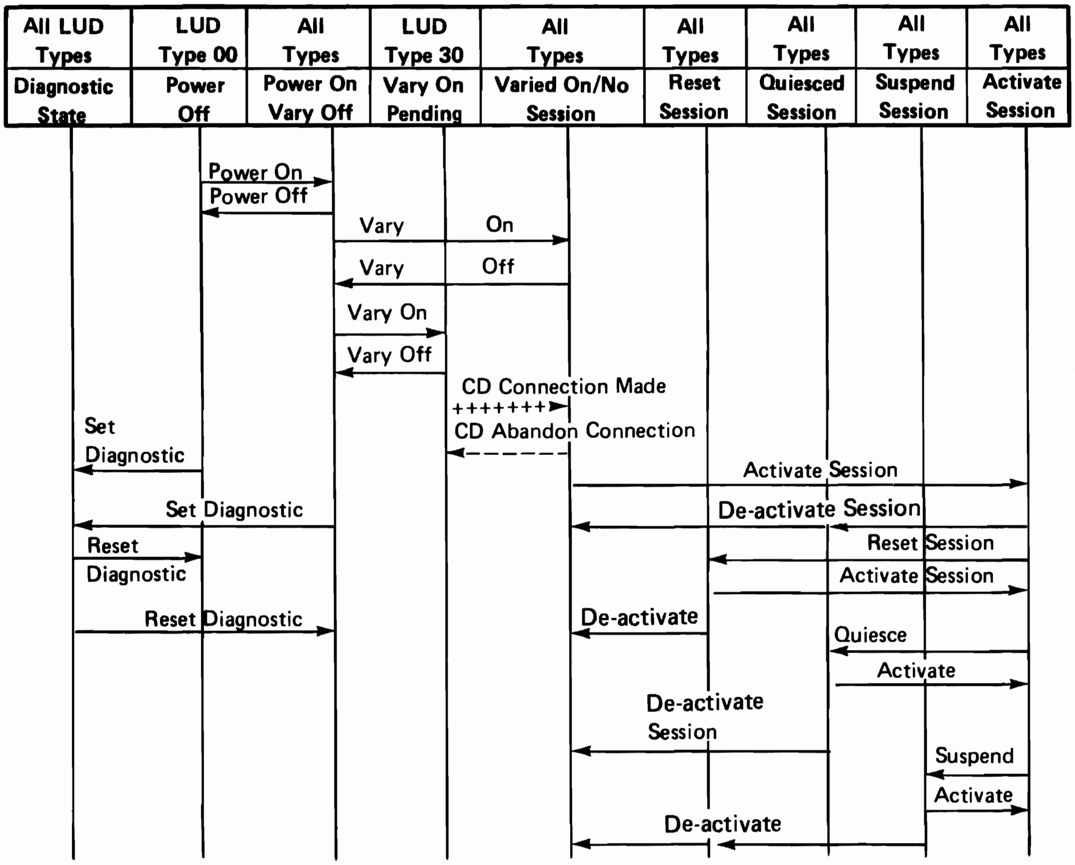
If the attachment method indicates that the LUD is attached to both a CD and an ND (LUD type 30) and the CD represents a station on a switched network, the following conditions apply:

- When the connection to the CD has not been established (CD is in a vary on pending state), the LUD is modified to a vary on pending state. The LUD is modified to a vary on state when the dial in or dial out function is completed, the CD is set to a vary on state, and the LUD contact event is signaled.

- If the connection to the CD has been established (CD is in a varied on state), the LUD is modified to a varied on state, and the LUD contact event is signaled.

- When the device cannot be initialized, a resource not available exception is signaled.

The modify time-out value field is used to specify the desired length of time (in standard time units) that the machine should allow for the modification operation to complete. The minimum time-out value is 10 seconds, and the maximum time-out value is 5 minutes. If the operation does not complete within the specified time, the operation is terminated and the partial system object damage exception is signaled. Error recovery procedures must be invoked to perform any shutdown or cleanup operations if this exception occurs. If no time-out value is specified in the modify template, a default time-out value of 30 seconds is used. Any nonzero time-out value supplied must fall within the time-out limits. This time-out value should not be construed as a maximum length of execution time for the Modify instruction. The time-out is only used internally to time some arbitrary portion of the operation to prevent the Modify instruction from never completing. Time-out will not occur in less than the specified time-out value. However, execution may validly be much longer than the time-out value when several elements are included in one Modify instruction because each element operation is timed separately.



Legend

——► State transitions due to MODLUD operations

-++++► State transitions due to MODCD on the related CD

– – – –► State transitions by system on behalf of MODCD

State transitions as a function of LUD type
Each LUD type supports only those transitions where beginning and ending states are both allowed states for that LUD type unless noted differently on the diagram.

Notes:

1. De-activate from active state causes a state change first to quiesced state and then to varied on state. De-activate from suspended state causes a state change first to reset state and then to varied on state.
2. For LUDs which are used for load/dump operations, de-activate is not allowed if load pending or dump pending conditions are set. See Figures 17-9 and 17-10 for load/dump change rules.

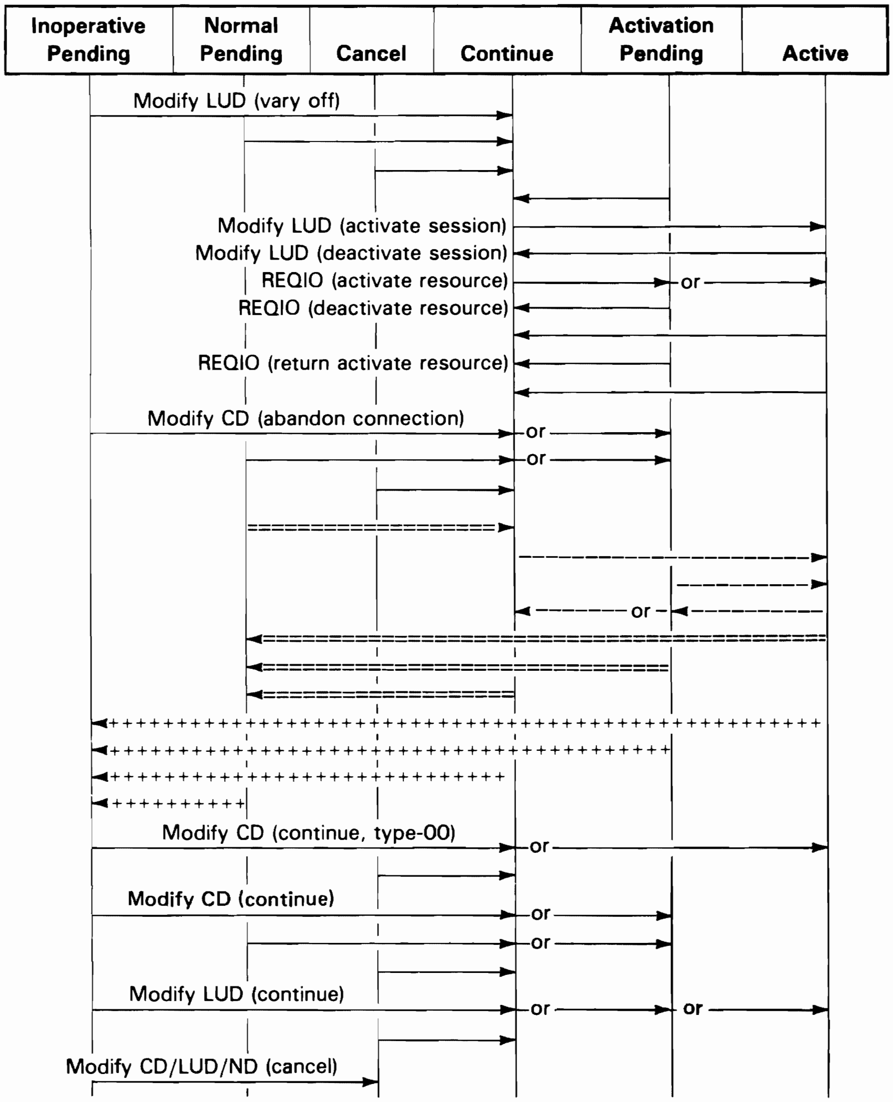
Figure 17-5 (Part 1 of 2). LUD State Change Rules

LUD States	**Diagnostic State**	**Power Off**	**Power On**	**Vary On Pending**	**Vary On**	**Reset Session**	**Quiesced Session**	**Suspend Session**	**Activate Session**
Element checking sequence and allowable states for modification									
1. Session information	No	Yes	Yes	No	No	No	No	No	No
2. Load/Dump indicator	No	Yes	Yes	Yes	Yes	Yes/No[1]	Yes/No[1]	Yes/No[1,2]	Yes/No[1]
3. Retry value sets	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
4. Error threshold sets	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
5. Device-specific contents	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

[1] **Transition is allowed only if the load/dump device is defined as an interruptible or exchangeable device.**

[2] **For interruptible devices only, transition from load or dump mode to normal mode is allowed, but transition from normal mode to load or dump mode is not allowed in suspended session state.**

Figure 17-5 (Part 2 of 2). LUD State Change Rules


Inoperative Pending
Normal Pending
Cancel
Continue
Activation Pending
Active
Modify LUD (vary off)
Modify LUD (activate session)
Modify LUD (deactivate session)
REQIO (activate resource)
or
REQIO (deactivate resource)
REQIO (return activate resource)
Modify CD (abandon connection)
or
or
or
Modify CD (continue, type-00)
or
Modify CD (continue)
or
or
Modify LUD (continue)
or
or
Modify CD/LUD/ND (cancel)

Legend

——► or ◄—— Status transition caused by successful completion of a source/sink Modify instruction or an asynchronous Request I/O instruction.

- - - - - - - - - -►
◄- - - - - - - - - -
◄==========
==========► Status transition asynchronous to a source/sink instruction.

◄+++++++++++ Status transition asynchronous to a source/sink instruction that is caused by a failure.

Figure 17-6. LUD Object Recovery/Resource Activation Status Transitions

Modify Instruction	Diagnostic State	Power Off	Power On Varied Off	Varied On Pending	Varied On No Session	Reset	Quiesced	Suspended	Active Session
Vary On	No	No	Yes	No	No	No	No	No	No
Reset	No	No	No	No	No	No	No	No	Yes
Quiesce	No	No	No	No	No	No	No	No	Yes
Suspend	No	No	No	No	No	No	No	No	Yes
Activate Session	No	No	No	No	Yes	Yes	Yes	Yes	No
Deactivate Session	No	No	No	No	No	Yes	Yes	Yes	Yes
CD Abandon Connection	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Vary Off	No	No	No	Yes	Yes	No	No	No	No
Cancel	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Continue	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes

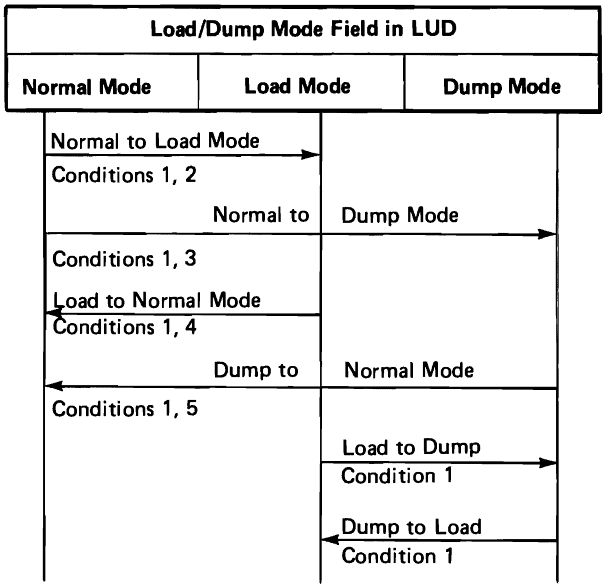
Figure 17-7. LUD State Modification Rules

Modify Instruction	Inoperative Pending	Normal Pending	Cancel	Continue	Activation Pending	Active
Vary On	No	No	No	Yes	No	No
Reset	Yes	Yes	Yes	Yes	Yes	Yes
Quiesce	Yes	Yes	Yes	Yes	Yes	Yes
Suspend	Yes	Yes	Yes	Yes	Yes	Yes
Activate Session	Yes	Yes	Yes	Yes	Yes	Yes
Deactivate Session	Yes	Yes	Yes	Yes	Yes	Yes
CD Abandon Connection	Yes	Yes	Yes	Yes	Yes	No
Vary Off	Yes	Yes	Yes	Yes	Yes	No
Cancel	Yes	No	No	No	Yes[1]	No
Continue	Yes	No	Yes	No	Yes[1]	No

[1] Modification is allowed when inoperative pending status is indicated along with activation pending status; otherwise No.

Figure 17-8. LUD Status Modification Rules

LUD Load/Dump Indicator Change Rules (LUD Types 00, 10)



Conditions:

1. Allowed if LUD status is powered off, powered on/varied off, vary on pending, or varied on.
2. Allowed if LUD status is reset or quiesced and load pending is on. This change is allowed only on interruptible load/dump devices and causes the load pending indicator to be reset.
3. Allowed if LUD status is reset or quiesced and dump pending is on. This change is allowed only on interruptible load/dump devices and causes the dump pending indicator to be reset.
4. Allowed if LUD status is reset, quiesced or suspended. This change is allowed only on interruptible load/dump devices and causes the load pending indicator to be reset.
5. Allowed if LUD status is reset, quiesced or suspended. This change is allowed only on interruptible load/dump devices and causes the dump pending indicator to be set.

Figure 17-9. LUD Load/Dump Indicator Change Rules

Load/Dump Indicator Field (Hex)	Diagnostic State	Power Off	Power On	Vary On Pending	Varied On	Reset Session	Quiesced Session	Suspended Session	Active Session
Load/Dump Device 00 = Not a Load/Dump Device 01 = Noninterruptible/ Nonexchangeable 11 = Interruptible 21 = Exchangeable	Not Modifiable Data (ignored by MODLUD instruction)								
Load/Dump Operating Mode (note 1) 00 = Normal 01 = Load Mode (primary device) 02 = Dump Mode (primary device) 21 = Load Mode (alternative device) 22 = Dump Mode (alternative device)	No	Yes				Note 2			No
Load/Dump Pending 0000 = Normal 0100 = Load Pending 0200 = Dump Pending	Not Modifiable Data (ignored by MODLUD instruction)								
Corresponding Primary Address (note 1) nnnn = Logical unit address of the primary device when this device is an alternative mode device	No	Yes				Ignored by MODLUD instruction			
Load/Dump Exchange Status (note 3) On Materialize: 010000 = This device is current. 000000 = This device is not current. On Modify: 01nnnn = Exchange to current when nnnn is the same as the logical unit addıess of the previous current device 000000 = No modification requested	No	No (template must contain hex 000000)							Yes
Load/Dump Process Performance Attributes	No	Yes				No			

Notes:

1. Load/dump mode settings and corresponding primary address settings must be compatible between this LUD and the corresponding LUD(s) at session activation.
2. Mode changes from primary device mode to alternative device mode or the reverse direction are not allowed for LUD states above varied on.
3. On modification, the other LUD with the logical unit address hex nnnn must be current, active, in a corresponding mode (load or dump), and must have a corresponding primary address that either indicates this LUD or indicates the same primary as indicated in this LUD. The other LUD will be changed to not current. Any unprocessed load/dump request I/O operations will be routed to the new current LUD. If the exchange to current occurs while request I/O operations are in process (no terminating errors such as EOV indicated), then the disposition of these requests is indeterminate.

Figure 17-10. LUD Load/Dump State Change Rules

The following conditions are required when sessions are activated or de-activated for exchanges:

- Activate session–If the LUD is in primary load mode or primary dump mode, the LUD will become current when activated. If the LUD is in alternative load mode or alternative dump mode, the corresponding primary LUD must be in active session and in a matching primary mode.

- De-activate session–If the LUD is in primary mode, it must be current and all alternative LUDs must already be de-activated. De-activation will cause the LUD to change to not current. If the LUD is in alternative mode it must be not current.

Authorization Required

- Operational
  - Operand 1

Lock Enforcement

- Modify
  - Operand 1
  - The CD that is specified by the forward object pointer of this LUD, if any, and only when this forward object is to be modified by the synchronous execution of this Modify LUD instruction on the status field of the LUD object

Note: The state change diagrams provided with the Modify Controller Description and Modify Network Description instruction show when the Modify Logical Unit Description instruction causes these modifications. For operations that involve the recovery/resource activation status field, objects are not ensured by object locks and the status fields may be changed independently by the machine or other processes regardless of the lock state of the object.

Events

0002 Authorization
    0101 Object authorization violation

000B Logical unit description
    0601 Logical unit description contact successful
        (for all Modify LUD-vary on instructions)
    0602 Logical unit description contact unsuccessful

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X		
0A Authorization				
01 Unauthorized for operation	X			
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state	X	X	X	
44 Partial system object damage	X			
1A Lock State				
01 Invalid lock state	X			X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	
34 Source/Sink Management				
01 Source/sink configuration invalid				X
03 Source/sink object state invalid	X			X
04 Source/sink resource not available				X
38 Template Specification				
01 Template value invalid		X		
02 Template size invalid		X		

MODIFY NETWORK DESCRIPTION (MODND)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04CF	Network description	Network description modification template	Modification options

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: Character(2) scalar (fixed-length).

Description: This instruction modifies the ND (network description) specified by operand 1 to the new values contained in the modification template specified by operand 2. The elements within the ND are modified based on the modification options specified by operand 3. Operand 2 must be 16-byte aligned.

The scalar specified in operand 3 cannot be defined by a data pointer.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

The following chart shows the modifiable elements that can be included in the template for operand 2. (Refer to the Create Network Description instruction for detailed descriptions of the elements.) The template can contain any combination of these elements as indicated by the option value in operand 3 and by including only those elements in the order shown here.

Elements Contained in the ND Template (ND Type 00) for Modify ND			Element Length	Sub-Element Length	Modify Option Values
Template size specification			Char(8)		
Modify time-out value			Char(8)		
State/status definition			Char(16)		4001
• State change/status				Char(6)	
Byte(s)	*Bit(s)*	*Meaning*			
0-1		**Reserved**		Bin(2)	
2		**Commands**			
	0	Reserved		Bit 1	
	1	Abandon call		Bit 1	
	2	Manual start data		Bit 1	
	3	Manual answer		Bit 1	
	4	Disable		Bit 1	
	5	Enable		Bit 1	
	6	Vary on		Bit 1	
	7	Vary off		Bit 1	
3		**Commands**			
	0	Set diagnostic mode		Bit 1	
	1	Reset diagnostic mode		Bit 1	
	2-7	Reserved		Bits 6	
4-5		**Reserved**		Char(2)	
• Recovery/resource activation				Char(2)	
Byte(s)	*Bit(s)*	*Meaning*			
0		**Status**			
	0-1	Reserved		Bits 2	
	2	Normal cancel		Bit 1	
	3	Normal continue		Bit 1	
	4-7	Reserved		Bits 4	
1		Reserved		Char(1)	
• Reserved				Char(8)	
Selectable mode data			Char(16)		4002
Communications subsystem parameters data			Char(16)		4004
Retry value sets			Char (6y + 2)		4020
Line-specific contents			Char (y + 4)		4040

y = Variable length of an element

Note: A combination of elements can be modified of the same Modify instruction by supplying a value in operand 3 that is the result of performing a logical OR on the modify option values of the desired elements.

Each modifiable element within an ND can be successfully modified only when in certain operational states of the network description.

Refer to Figures 17-11 through 17-14 for a description of the states, the status, and the elements of the ND that can be modified.

When the state of the ND does not allow modification of a requested element, a source/sink object state invalid exception is signaled, and modification is stopped. All elements that were modified before the exception remain successfully modified. The exception information identifies the element responsible for the exception.

Any modification options that invoke other changes to the object, along with changes to the recovery/resource activation field, cause a template value invalid exception to be signaled.

Modification options that include the state change/status element of the ND and involve a vary state change to this element have the following additional exceptions that can be signaled if conditions are not valid for the requested change:

- Source/sink object state invalid–This exception occurs because an associated CD (controller description) or LUD (logical unit description) is not in the proper state for this ND to be varied off.

  The ND for a particular line must be explicitly varied on before the CDs are varied on by using the Modify Network Description instruction. If the ND is not varied on, a source/sink object state invalid exception is signaled by the Modify Controller Description instruction. Likewise, before the ND can be varied off, the CDs must be varied off. If not, a source/sink object state invalid exception is signaled by this instruction.

- Source/sink resource not available–This exception occurs because the appropriate hardware or machine support components are not installed on the system to match this ND. This exception can also occur because of a hardware failure occurring anywhere in the communications network while the system is attempting to establish the vary on function for the ND.

  When the attachments for the line are activated, the ND object is set to a vary on state. If the attachments cannot be activated, a resource not available exception is signaled, and the instruction is stopped at that point.

  When the attachments for nonswitched lines and local loops are activated, the line is prepared for transmitting to the attached devices.

When the ND is associated with a communications attachment configured for switched network support, the attachment is activated and made ready to establish a switched connection. This connection can be established in the following two ways:

Dial In

When an incoming call is received, if the ND is in the ND candidate list of the CD that called in and if that ND is in switched enabled state with dial in allowed, the connection is made.

Dial Out

When a modify CD dial command is issued to the CD, the ND candidate list in the CD is referenced again. If an ND is found that is in a switched enabled state with dial out allowed in the switched connection method field and this ND is not in use, the connection is established.

For both dial in and dial out, an event is signaled upon completion of the activity. Also, the forward and backward switched connection pointers in the ND and CD are updated to complete the addressing chain.

The modify time-out value field is used to specify the desired length of time (in standard time units) that the machine should allow for the modification operation to complete. The minimum time-out value is 10 seconds, and the maximum time-out value is 5 minutes. If the operation does not complete within the specified time, the operation is terminated and the partial system object damage exception is signaled. Error recovery procedures must be invoked to perform any shutdown or cleanup operations if this exception occurs. If no time-out value is specified in the modify template, a default time-out value of 30 seconds is used. Any nonzero time-out value supplied must fall within the time-out limits. This time-out value should not be construed as a maximum length of execution time for the Modify instruction. The time-out is only used internally to time some arbitrary portion of the operation to prevent the Modify instruction from never completing. Time-out will not occur in less than the specified time-out value. However, execution may validly be much longer than the time-out value when several elements are included in one Modify instruction because each element operation is timed separately.

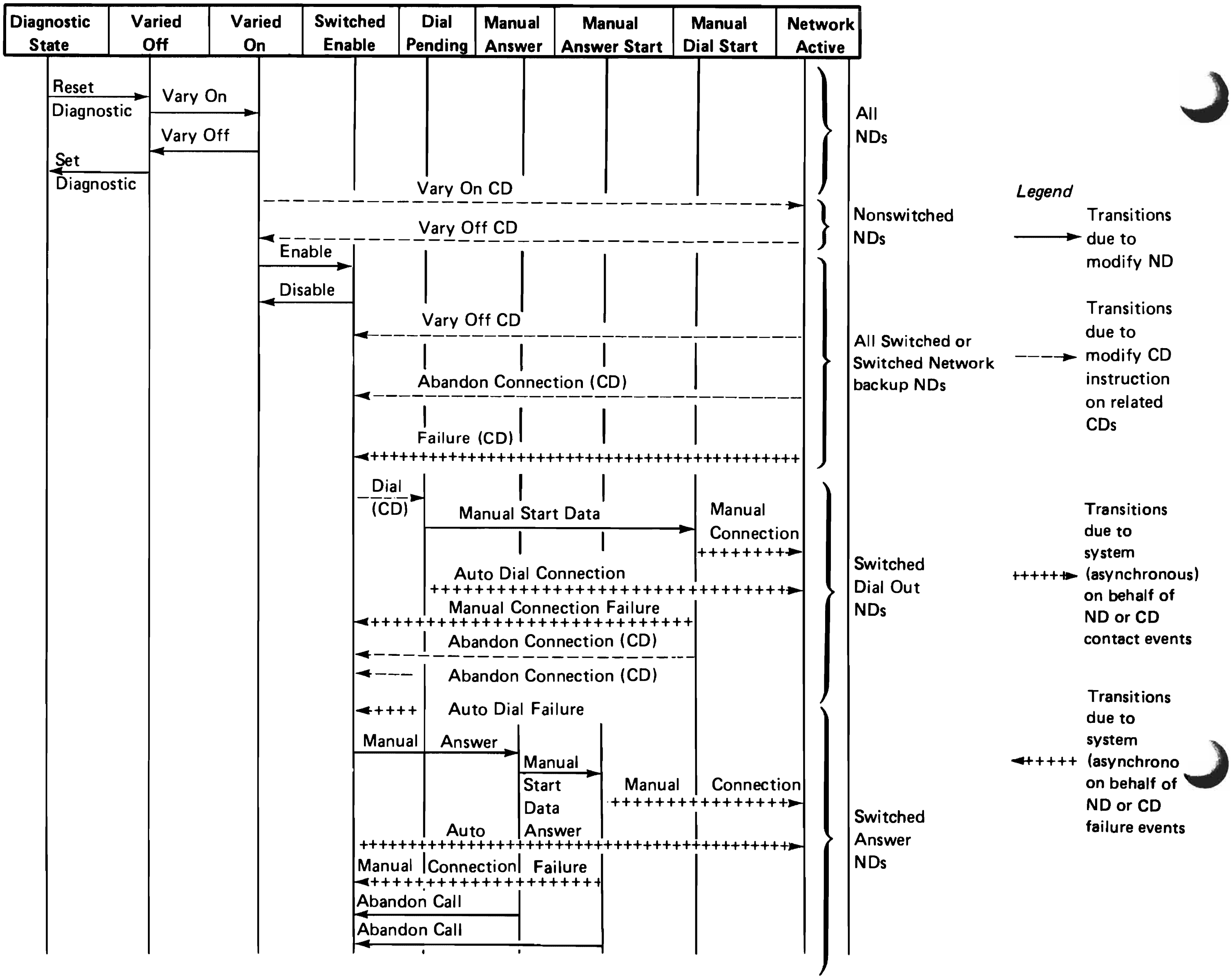
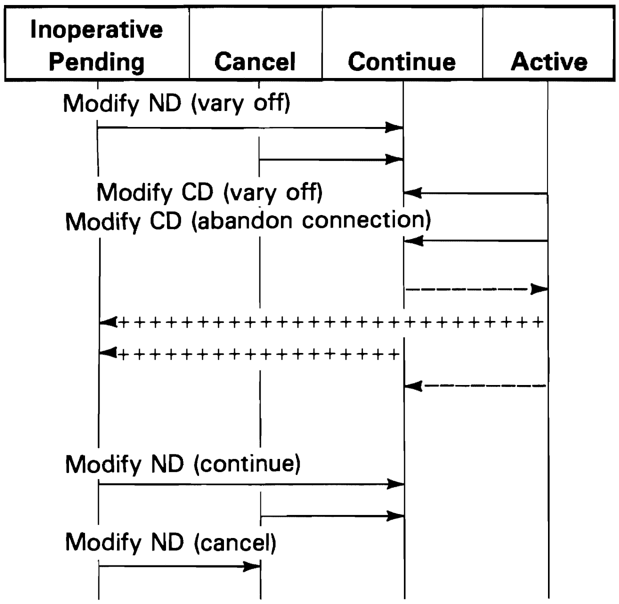


Figure 17-11 (Part 1 of 2). ND State Change Rules

ND States	Diagnostic State	Vary Off	Vary On	Switched Enable	Manual Answer	Manual Answer Start	Manual Dial Start	Network Active
Element checking sequence and allowable states for modification								
1. Selectable modes	No	Yes	No	No	No	No	No	No
2. Communications subsystem parameters	No	Yes	No	No	No	No	No	No
3. Eligibility list	No	Yes	Yes	No	No	No	No	No
4. Retry value sets	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
5. Line-specific contents	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Note: Transition from load or dump mode to normal mode is allowed, but transition from normal mode to load or dump mode is not allowed in suspended session state.								

Figure 17-11 (Part 2 of 2). ND State Change Rules



Legend

——► or ◄—— Status transition caused by successful completion of a source/sink Modify instruction.

– – – – – – – – – – –► Status transition asynchronous to a source/sink instruction. This status change is indicated by a CD contact successful event.

◄ – – – – – – – – – – – Status transition asynchronous to a source/sink instruction. This status change is indicated a CD contact unsuccessful event for a switched line, a CD loss of contact event, or a CD failure event for a switched line.

◄++++++++++ Status transition asynchronous to a source/sink instruction. This status change is indicated by a ND line failure event or a partial system object damage set event.

Figure 17-12. ND Object Recovery/Resource Activation Status Transitions

Modify Instruction	Diagnostic Mode	Varied Off	Varied On	Switched Enabled	Dial Pending	Manual Answer	Manual Answer Start Data	Manual Dial Start Data	Network Active
Vary On	No	Yes	No	No	No	No	No	No	No
Enable	No	No	Yes	No	No	No	No	No	No
CD Dial	No	No	No	Yes	No	No	No	No	No
Manual Answer	No	No	No	Yes	No	No	No	No	No
Manual Start Data	No	No	No	No	Yes	Yes	No	No	No
Abandon Call	No	No	No	No	No	Yes	Yes	No	No
Disable	No	No	No	Yes	No	No	No	No	No
CD Abandon Connection	No	No	No	No	Yes	No	No	Yes	Yes
Vary Off	No	No	Yes	No	No	No	No	No	No
Cancel	No	No	Yes[1]	Yes	No	No	No	No	Yes[1]
Continue	No	No	Yes[1]	Yes	No	No	No	No	Yes[1]

[1] Modification is allowed for a nonswitched line; otherwise No.

Figure 17-13. ND State Modification Rules

Modify Instruction	Inoperative Pending	Cancel	Continue	Active
Vary On	No	No	Yes	No
Enable	Yes	Yes	Yes	No
CD Dial	No	No	Yes	No
Manual Answer	No	No	Yes	No
Manual Start Data	No	No	Yes	No
Abandon Call	No	No	Yes	No
Disable	Yes	Yes	Yes	No
CD Abandon Connection	No	No	Yes	No
Vary Off	Yes	Yes	Yes	No
Cancel	Yes	No	No	No
Continue	Yes	Yes	No	No

Figure 17-14. ND Status Modification Rules

Authorization Required

- Operational
  - Operand 1
  - The CD which is specified by the lockword object pointer in this ND, if any, and only when this lockword object is to be modified by the synchronous execution of this Modify ND instruction

Lock Enforcement

- Modify
  - Operand 1
  - The CDs that are specified by the backward object pointer list of this ND, and only when these backward objects are to be modified by the synchronous execution of this Modify ND instruction on the status field of the ND object

Note: The state change diagrams provided with the Modify Controller Description instructions show when the Modify Logical Unit Description instruction will cause these modifications. For operations that involve the recovery/resource activation status field, objects are not ensured by object locks and the status fields may be changed independently by the machine or other processes regardless of the lock state of the object.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

000E Network description
    0401 XID exchange failure

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0201 Machine context damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X		
0A Authorization				
01 Unauthorized for operation	X			
10 Damage Encountered				
02 Machine context damage state				X
04 System object damage state	X	X	X	X
44 Partial system object damage	X			
1A Lock State				
01 Invalid lock state				X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer address invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	
03 Scalar value invalid			X	
34 Source/Sink Management				
01 Source/sink configuration invalid	X			
03 Source/sink object state invalid	X			X
04 Source/sink resource not available				X
38 Template Specification				
01 Template value invalid		X		
02 Template size invalid		X		

REQUEST I/O (REQIO)

Op Code (Hex)	Operand 1
0471	Source/sink request (SSR)

Operand 1: Space pointer.

Description: Operand 1 references an area in a space called the SSR (source/sink request). The SSR contains the pointers and data that are required to define the REQIO operation and must be 16-byte aligned.

The SSR contains three pointers. The first pointer specifies the source/sink description object for the I/O device or component to be used. The second pointer identifies the queue to which final disposition of the requested I/O operation is to be returned. The third pointer locates the SSD (source/sink data), which is the data area for the requested I/O operation.

The data contained in the SSR defines the type of REQIO function to be performed, certain controls, identification, sequencing functions, and the set of operational orders or commands for the I/O device.

Certain checks are made on the objects referenced by the SSR pointers and on the SSR data before the I/O operation is started. For example, the SSR data area must contain valid function and control fields but the device operational orders (called request descriptors) are not verified during the processing of the Request I/O instruction.

The first pointer in the SSR must represent a proper source/sink object that is authorized to this user. The object must be in a lock state that allows its use. For a normal or load/dump REQIO function, the object must be a LUD (logical unit description) in the active session state.

The second pointer must represent a queue that is authorized to this user. The queue must (1) be a keyed queue with a key length of 10 bytes or larger, (2) have a message size of 64 bytes of pointer and scalar data, and (3) have a message element available.

The third pointer may reference a space as a data area. If the preceding conditions are not satisfied, an appropriate exception is signaled and the instruction is terminated.

If the preceding conditions are satisfied, the requested I/O operation is scheduled for execution and the Request I/O instruction is complete.

The requested I/O operation is then processed asynchronously. The completion of this request I/O operation is indicated by the posting of a feedback record to the request I/O response queue specified in the SSR and also by the signaling of the request I/O completed event (only when such event signaling was specified in the SSR). Errors encountered during the machine processing of this requested operation are indicated in the feedback record. These errors include those encountered within the RDs (request descriptors) in the SSR, any authorization or lock enforcement violations encountered within load/dump operations, or any hardware errors detected while processing the I/O operation.

Some failures may occur during an I/O operation that may prevent the I/O operation from completion. Because the Request I/O instruction does not provide a time-out in these cases, indefinite waits or operator intervention recovery actions may occur. The user must prevent these waits or operator intervention recovery actions by providing a time-out. The time-out can be indicated in the Dequeue instruction by entering it in the dequeue-wait-time-out parameter (see Dequeue instruction in Chapter 12 for details). The time-out values to be used are device-dependent and are a function of the particular I/O operation being performed by the device.

The sequence of events is as follows:

1. Request I/O instruction is executed.

2. The I/O operation is completed.

3. The Dequeue instruction is issued to retrieve the feedback record.

4. Completion of the I/O operation is signaled by the retrieval of the feedback record.

The SSR space object contains the following:

Template size specification	Char(8)
– Size of template	Bin(4)
– Number of bytes available for materialization	Bin(4)
Request I/O time-out (synchronous only)	Char(8)
Source/sink object	System pointer
Response queue	System pointer
Source/sink data area (or binary 0)	Space pointer
Optional pointer area	Char(16)
Reserved (binary 0)	Char(8)
Request I/O timestamp	Char(8)
Request priority	Bin(2)
Request ID	Bin(2)
Function field	Char(1)
Request control field	Char(1)
Key length	Bin(2)
Offset to key field	Bin(2)
Request descriptor count	Bin(2)
Offset to request descriptor field	Bin(2)
Offset to the request I/O variable parameters	Bin(2)
Variable-length entries:	
– Key field (variable 10-256 bytes)	Char(*)
– Request descriptor field (modulo 16, 2-byte aligned; or modulo 96, 16-byte aligned for load/dump requests)	Char(*)
Request descriptor 1	Char(16) or (96)
.	.
.	.
.	.
Request descriptor n	Char(16) or (96)
Request I/O variable parameters (must be 16-byte aligned)	Char(*)
– Length of variable parameter area	Bin(4)
– Reserved	Char(28)
– Variable parameters (specific to the device or support used)	Char(*)

These entries are defined as described in the following paragraphs. The information associated with service request I/O, service functions, and service exceptions is for use by service personnel.

Template size specification–This entry defines the standard template header data. The size of the template field must indicate a sufficient number of bytes to contain all the following entries in the SSR including the lengths and positions of all the variable length items in the SSR. The number of bytes available for materialization field is not used by the Request I/O instruction.

Request I/O time-out–This field is used to specify the desired length of time (in standard time units for the system model being used) that the machine allows for the synchronous request I/O (with task switching) operation to complete. If the operation does not complete within the specified time, the operation is terminated and the partial system object damage set exception is signaled. Error recovery procedures must be used to perform any cleanup operations after the operation is terminated. If no time-out value is specified in the template and the function field specifies a synchronous operation, a default time is used. The default time is used only for internal failures and is not the maximum execution time for the synchronous Request I/O instruction. This field is ignored for asynchronous request I/O operations and for synchronous operations that do not have task switching.

Source/sink object–This entry can be a system pointer to an LUD for a normal or load/dump request I/O operation; can be a system pointer to an LUD, a CD, or an ND for MSCP (machine services control point) request I/O operations; or can contain binary 0 for service request I/O operations.

Response queue–This entry is a system pointer to the request I/O response queue. This pointer is ignored for synchronous requests.

Source/sink data area–This entry can be a space pointer to an SSD area for any request I/O operations, or it can be binary 0.

Optional pointer–This space must be null (binary 0) for all operations except service requests. When the SSR function field specifies service, this space will either contain a space pointer or be null.

Request I/O timestamp–This field is set by the machine to indicate (in standard time units for the system) the time when this request was processed. The request I/O response queue contains a standard enqueue timestamp that is also set to indicate the time of actual completion of the resulting I/O operation.

Request priority–This field defines the priority of each Request I/O instruction relative to other Request I/O instructions. As each Request I/O instruction is processed, this field is used to schedule the priority of each request with respect to any previously issued requests that are still stacked for processing. Priority values can be assigned in binary collating sequence with hex 0000 being the highest priority and hex FFFF being the lowest priority. This field is ignored for synchronous requests that do not have task switching.

Request ID–This field is used to assign unique identification to each source/sink request. This unique identification is copied into the feedback record associated with this Request I/O instruction and thus provides an external capability to correlate feedback records with the Request I/O instruction that generated them. The request ID field is also used to control the signaling of the request I/O completed event. When bit 0 in the request ID field is 1, the request I/O completed event is signaled when the feedback message is enqueued. This event indicates that the processing of this request is completed. When bit 0 in the request ID is 0, no event is signaled. This field is ignored for synchronous requests.

Function field–This field defines the type of request I/O as follows:

Bits 0-3= 1000–normal request I/O
= 0100–MSCP request I/O
= 0010–load/dump request I/O
= 0001–service request I/O

Bits 4-7 are function dependent and are defined for each device or function in the *IBM System/38 Functional Reference Manual–Volume 2*, GA21-9800.

If the function field indicates a load/dump request, then the load/dump indicator in the LUD must indicate load/dump mode or the source/sink invalid object state exception is signaled.

Request control field–This field defines request I/O control functions as follows:

Hex 10=Device specific control functions
Hex 20=Synchronous request I/O with task switching
Hex 21=Synchronous request I/O without task switching
Hex D5=Normal request I/O
Hex C3=Request I/O continue

Request I/O continue is used for error recovery situations. When a terminating error is posted in the feedback record, normal request I/O processing is inhibited until a request I/O continue command is issued. Normal requests can be issued before the request I/O continue command, but these requests remain enqueued for processing until the continue command is issued. The request I/O continue function is internally assigned a higher priority than normal requests, and, consequently, is processed before the normal requests that are enqueued for processing. When request I/O continue is indicated in the SSR, the RD count must contain 0. When request I/O normal is indicated, the RD count must contain a value greater than 0.

Synchronous request I/O specifies that the machine is to process the operation synchronously with the process that issued this instruction, suspend all other process activity until the completion of this instruction, and then continue processing the next sequential instruction in the process. Request I/O completion is indicated by the ending status of the instruction. No feedback record is posted in the response queue. Synchronous requests can be with or without task switching in the machine. When task switching is allowed, time-out considerations are the same as those for the modify instructions.

Device specific control functions specify that the request I/O for the device has task switching or that it does not have task switching.

Key length – This field indicates the length of the request key field in this SSR. This value must also match the key length attribute of the response queue specified in this SSR.

Offset to key field–This field indicates the location within the SSR where the request key field has been placed. This offset value is defined from the beginning of the SSR and must be a positive value.

Request descriptor count–Bin(2)–This field indicates the number of request descriptors contained in the request descriptor field in this SSR.

Offset to request descriptors–This field indicates the location within the SSR where the request descriptor field has been placed. This offset defines a positive value offset from the beginning of the SSR and must define either a 2-byte aligned location for normal MSCP or service requests, or a 16-byte aligned location for load/dump requests.

Offset to the request I/O variable parameters–This field indicates the location in the SSR where the variable parameters (if any) are placed. This offset is defined as a positive value offset from the beginning of the SSR and must define a 16-byte aligned location.

Key field–This field is used by the machine to post the feedback record onto the request I/O response queue. This is the key value to be used by the Dequeue instruction to retrieve the feedback record corresponding to this Request I/O instruction. Feedback records are posted to the response queue in binary collating sequence order so that standard dequeue keyed rules apply. Refer to the Dequeue instructions for details.

Request descriptor field–This part of the source/sink request contains the 16-byte RDs, which must be halfword aligned (or 96-byte RDs, 16-byte aligned for load/dump) for the RIUs (request information units) and/or system pointers involved in the source/sink operation. The RD is specifically tailored to a particular device type, method of attachment, and/or the mode of the Request I/O instruction. Refer to the *IBM System/38 Functional Reference Manual–Volume 2,* GA21-9800 for the contents of request descriptors for specific devices.

Request I/O variable parameters–This variable parameter area is used for additional data that is necessary to support certain devices or support mechanisms.

The SSD (source/sink data) located by the SSR, when it is present for an I/O request, represents the data area (I/O data buffer) associated with the particular request. The contents of the SSD are also defined for each device supported on a particular model of the system in the *IBM System/38 Functional Reference Manual–Volume 2,* GA21-9800. The significance concerning this SSD space is that it can be subdivided into segments called RIUs (request information units), which have a one-to-one correspondence with the RDs in the SSR so that feedback record subdivisions can be defined.

Unpredictable results can occur if the space object that contains the SSD is modified, destroyed, or truncated when the space is being used to complete the request I/O operation.

The message associated with a Dequeue instruction is called a feedback record only when the message resulted from a Request I/O operation associated with this response queue. The message operand on the Dequeue instruction has the following information inserted into it to form the feedback record:

Field	**Format**
Source/sink request address	Space pointer
Request ID	Bin(2)
Error summary	Bin(2)
RD number	Bin(2)
RIU segment count	Bin(2)
Device-dependent status	Char(40)

Definitions of these feedback record fields follow:

Source/sink request address–This pointer locates the SSR (source/sink request) that the issuer of the Request I/O instruction supplied as its operand. This SSR can optionally have new data inserted into it based on the Request I/O operation that was performed.

Request ID–This field contains the same value as the request ID field within the SSR of the Request I/O instruction that generated this feedback record. It is used to correlate responses to requests.

Error summary–This field indicates the final disposition of the request I/O operation. The contents of this field are:

Byte 0		**Byte 1**	
t n i s ffff		r d nnnnnn	
0	7	0	7

Byte 0–Error Attributes

Bits 0, 1 (t = terminate, n = not normal)

00 = Normal condition
01 = Not normal, nonterminating error
11 = Not normal, terminating error

Terminating errors are those for which processing of subsequent request I/O operations is suspended until higher-level Request I/O Control instructions or session state changes through the Modify LUD instruction are requested.

Bit 2 (i = included)

0 = Device-dependent data is not included.
1 = Device-dependent data is included in the device-dependent status area of this feedback message.

Bit 3 (s = specific error)

0 = Error code defined in byte 1.
1 = Device-specific error code is defined in byte 1, and none of the definitions for byte 1 apply.

Bits 4, 5, 6, 7 (f = function)

0000 = Normal function
01nn = Load/dump function; nn is defined in *IBM System/38 Functional Reference Manual –Volume 2 Load/Dump Object Management*
1000 = MSCP function
1100 = Service function

Byte 1–Error Type

Bits 0, 1 (r = SSR, d = SSD)

00 = Error type is not associated with the SSR or the SSD.
10 = Error type is associated with the SSR (source/sink request).
01 = Error type is associated with the SSD (source/sink data).
11 = Error type is associated with load/dump operations.

Bits 2-7 Bits 2 through 7 are combined with the r and d bits to provide the following byte 1 error type definitions.

Byte 1 Error Types Defined

Hex 00=No error conditions

Hex 08=Request I/O continue response

Hex 09=Partially processed request–terminated because of a reset session, error on quiesce session, or error on suspend session

Hex 0A=Unprocessed request–results from a reset session, error on quiesce session, error on suspend session, or a terminating error

Hex 0B=Partial damage encounter–The machine has encountered a partial damage situation that prevents successful completion of this request I/O operation. A partial system object damage event has been signaled independently.

Hex 0C=Data flow control sequence error

Hex 0F=Reserved for use above the Machine Interface

Hex 10=Unrecoverable error–LUD Type 00 or 10

Hex 11=Read terminated–device control error

Hex 12=Read completed–device control error

Hex 13=Data truncated–device control error

Hex 14=Command terminated–sequence error

Hex 15=Command terminated

Hex 16=End of file

Hex 17=End of volume

Hex 18=Command terminated–results from the conditions sensed

Hex 20=Unrecoverable error–LUD type 30

Hex 21=Line nonfunctional

Hex 22=Station nonfunctional

Hex 24=Send/receive error

Hex 26=Bind rejected

Hex 28=Invalid information unit

Hex 29=Bind host pacing parameter error

Hex 30=MSCP-invalid LUD type

Hex 31=MSCP-LUD not varied on

Hex 32=MSCP-invalid request header (RH)

Hex 33=MSCP-invalid transmission header (TH)

Hex 34=Cancel failure on ND, CD, or LUD

Hex 35=De-activate resource error

Hex 36=Return activate resource error

Hex 37=Return request I/O operation

Hex 40=Invalid source/sink data (SSD)

Hex 41=SSD object unusable (destroyed or suspended)

Hex 42=Invalid SSD data

Hex 43=Invalid SSD boundary alignment

Hex 44=SSD byte space too small

Hex 45=SSD byte space too large

Hex 46=Invalid number of pointers in SSD

Hex 47=Invalid pointer in SSD

Hex 48=Pointer in SSD references an unusable object (destroyed or suspended)

Hex 54=Respective session not active (SSCP to LU or LU to LU)

Hex 55=Data traffic session not active

Hex 80=Invalid source/sink request (SSR)

Hex 81=SSR object unusable (destroyed or suspended)

Hex 82=Invalid LUD pointer

Hex 83=Invalid response queue pointer

Hex 84=Invalid SSD pointer

Hex 85=Invalid function field

Hex 86=Invalid RD count field

Hex 87=Invalid RD

Hex 88=Invalid RD sequence

Hex 89=Invalid control field–continue out of sequence

Hex 8A=Conversation ID error

Hex 8B=Mode name error

Hex C0=Load/dump storage error

Hex C1=Insufficient user profile space for create and load

Hex C2=Invalid lock

Hex C3=Insufficient size of user profile or context for create and load

Hex C4=Duplicate object on create and load

Hex C5=Data space index sequence error on load or create and load

Hex C6=Load/dump object destroyed

Hex C7=Data space field descriptor mismatch on load or data space index key specification mismatch

Hex C8=Reserved

Hex C9=Object name, type, subtype mismatch on load

Hex CA=Data space or data space index is in use

Hex CB=Entry cannot be journaled

Hex CC=Data base linkage problem

Hex CD=Load/dump object damaged

Hex CE=Load/dump invalid version level

Hex CF=Same request I/O–SSR not returned after EOF (end of file), EOT (end of tape), or EOV (end of volume)

Hex Dn=Load/dump errors which are further defined in model dependent documentation

RD number–This number indicates the request descriptor that is within the Request I/O instruction and is appropriate for the ending status of that instruction. Normally, it is the last RD in the request, and in terminating error cases it is the RD on which the failure occurred.

RIU segment count–This count indicates a further breakdown to the segment within the RIU (request information unit) associated with the RD number if such a breakdown is meaningfully defined for each device type.

Device-dependent status–This field indicates further status associated with the error summary field. This field is uniquely defined for each type of device supported on the system.

Note: The Request I/O instruction normally initiates asynchronous I/O hardware operations that, under abnormal circumstances or hardware failures, may fail to complete. The Request I/O instruction does not provide any time-out mechanism for these cases as is provided by the Modify instructions or is provided for synchronous Request I/O instructions. Whenever possible, the user should provide time-out mechanisms for feedback records to prevent these I/O failures from causing indefinite waits, which ultimately require operator-initiated recovery actions. Because the Request I/O instruction execution is asynchronous to the actual hardware operations (that is, the instruction completes before the actual operation is started by the machine), timing must be done on the Dequeue instruction, which retrieves the feedback record that signals the actual completion of the I/O operation. This timing can be done by setting a time-out value for the dequeue–wait–time-out parameter on the Dequeue instruction. Time-out values to be used are device-dependent and are a function of the particular I/O operation being performed by that device.

Authorization Required

- Operational
  - LUD, CD, or ND specified in the SSR

- Insert
  - Queue specified in the SSR (request I/O response queue)

- Retrieve
  - Contexts referenced for address resolution

- Service–special authorization

  Specific authorization for load/dump operations is described in *IBM System/38 Functional Reference Manual–Volume 2 Load/Dump Object Management.*

Lock Enforcement

- Modify
  - The LUD, CD, or ND specified by the first system pointer in the SSR
  - The request I/O response queue specified by the second system pointer in the SSR

- Object control
  - Any system objects specified in the SSR for request I/O functions specifying load operations

- Materialize
  - Contexts referenced for address resolution
  - Any system objects specified in the SSR for Request I/O functions specifying dump operations

Events

0002 Authorization
0101 Object authorization violation

000B Logical unit description
0701 Operator intervention required (signaled asynchronously to execution of Request I/O instruction)
0801 Device failure (signaled asychronously)
0901 Request I/O completed (signaled asynchrono to execution of Request I/O instruction)

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0012 Queue
0401 Queue message limit reached
0501 Queue extended

0016 Machine observation
0101 Instruction reference

0017 Damage set
0201 Machine context damage set
0401 System object damage set
0801 Partial system object damage set

001A Journal port
0301 Entry not journaled
0401 Journal space attached to a journal port is not usable

001C Journal space
0301 Threshold reached

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
02 Machine context damage state		X
04 System object damage state		X
44 Partial system object damage		X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer address invalid object	X	
26 Process Management		
02 Queue full	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
2C Program Execution		
06 Instruction cancellation	X	
07 Instruction termination		X
2E Resource Control Limit		
02 Process storage limit exceeded	X	X
30 Journal Management		
06 Journal space not at a recoverable boundary	X	
32 Scalar Specification		
01 Scalar type invalid	X	

Exception	Operand 1	Other
34 Source/Sink Management		
01 Source/sink configuration invalid		X
03 Source/sink object state invalid		X
38 Template Specification		
01 Template value invalid	X	
02 Template size invalid	X	
3C Service		
01 Invalid service session state		X
02 Unable to start service session		X

REQUEST PATH OPERATION (REQPO)

Op Code (Hex)	Operand 1
0475	Path operation template

Operand 1: Space pointer.

Description: This instruction requests that the path operation described in the template be performed for the internal machine path function specified in the template. The template specifically describes the operation and contains the data involved with the operation. The template also specifies the queue (request path operation response queue) to which an asynchronous message called a feedback record is to be sent by the machine upon completion of the operation by this instruction.

Operand 1 references an area in a space called the path operation template, which contains the pointers and data required to define the REQPO operation. This area must be 16-byte aligned.

The template contains several pointers, the first specifies the requester object which is using the requested path operation. The second specifies the queue where final disposition of the requested path operation is to be returned. The third system pointer specifies the secondary object involved in establishing the path operation. This pointer is optional depending on the specific role and function requests being made so this pointer is located in the dependent portion of the template.

Other optional pointers may exist within the dependent portion of the template for other dependent operations.

Verifications are performed on the objects referenced by the pointers and on the data before execution of the path operation is started.

The path operation function to be performed is indicated by the value of the request role field in the path operation template as follows:

Hex Value	Path Operation
Hex 0	Passthru target node operation
Hex 1	Passthru intermediate node operation
Hex 2	Passthru source node operation
Hex 3	Passthru virtual workstation user operation
Hex 8	Load/dump dump space operation

The remaining request role field values are reserved and result in the signaling of the template value invalid exception if specified.

The first pointer, the requestor object pointer, must address an object for which path operations are supported or the instruction results in the signaling of the object not eligible for operation exception.

The first pointer must be a system pointer to a LUD and the LUD must be one of the types which can support the path operation as defined in model dependent documentation. This field may also contain binary zero if the specific role and function request does not require a LUD system pointer.

The second pointer must represent a queue, authorized to this user, in a lock state allowing its use, and must be a keyed queue with a key length of 16 bytes, a message size of 64 bytes of pointer and scalar data, and a message element available. If the above conditions are not satisfied, an appropriate exception is signaled and the instruction terminated.

The third (optional) LUD pointer has the same requirements as the first pointer.

Any additional optional pointers must satisfy the requirements for these pointers as stated in model dependent documentation.

If the above verifications are successful, the requested path operation is scheduled for execution by the machine and the Request Path Operation instruction is completed at this time.

The requested path operation is then processed asynchronously by the machine. The completion of this processing is indicated by the posting of a feedback record to the response queue specified in the template and signaling of the request path operation completed event if such event signaling was specified in the template. Errors encountered during the machine processing of this requested operation are indicated in the feedback record. These errors include those encountered within the template, any authorization or lock enforcement violations encountered, or any hardware errors detected while processing the path operation.

Contents of the Path Operation Template

The space object template contains the following:

• Template size specification	Char(8)
– Size of template	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Reserved	Char(8)
• Requester object	System pointer
• Response queue	System pointer
• Reserved	Char(16)
• Request key field	Char(16)
• Path identifier	Char(16)
• Request path timestamp	Char(8)
• Request operations	Char(8)
– Request priority	Bin(2)
– Request ID	Bin(2)
– Request control field	Char(4)
– Request role	Bit(4)
– Request functions	Bit(28)
• Variable parameters	Char(*)

Definition of these entries is as follows:

- Template size specification–This entry defines the standard template header data. The size of template field must indicate a sufficient number of bytes to contai all of the following entries in the template including the l and positions of all the variable length items. The numb available for materialization field is not used by the Request Path Operation instruction.

- Requester object–System pointer to a requester object for the operation.

- Response queue–System pointer to the request path response queue.

- Key field–This field is used by the machine to post the feedback record onto the request path operation response queue. This is the key value to be used by the Dequeue instruction to retrieve the feedback record corr to this Request Path Operation instruction. Feedback rec are posted to the response queue in binary collating seq order so that standard dequeue keyed rules apply. Refer Dequeue instruction for details.

- Path ID–This field may contain binary zeros on input ID that was returned in this field on previous executio of the Path Operation instruction. The request role and r function fields define the control over the use of the pat ID field as defined in model dependent documentation.

- Request path timestamp–This field is set by the mach indicate (in standard time units for the system) the ti at which this request was processed. The response que contains a standard enqueue timestamp that also will be to indicate the time of actual completion of the resulting path operation. Refer to the Enqueue instruction for loca of this other timestamp.

- Request priority–This field is used to establish the priority of each Request Path instruction relative to each other. As each Request Path instruction is processed, this field is used to schedule the priority of this request with respect to any previously issued requests that are still stacked for processing. Priority values can be assigned in binary collating sequence with hex 0000 being the highest priority and hex FFFF being the lowest priority.

- Request ID–This field can be used to assign any unique identification to each path operation. It is copied into the feedback record that results from this request path operation and thus provides an external capability to correlate feedback records with the REQPO which generated them. It is also used to control signaling of the request path operation completed event. If bit 0 in the request ID field is equal to 1, this event is signaled at the time the feedback message is enqueued. This indicates that the processing of this request is completed. If bit 0 in the request ID is equal to zero, no event is signaled.

- Request control field–The two subfields of this component are used to define the request role and request functions which are further defined for each specific usage in model dependent documentation. These fields control operation and use of the path ID and also the addition variable template parameters defined for each use of the instruction.

- Variable parameters–This variable extension to the template is defined in model dependent documentation and is controlled by the defined values for the request role and request function fields as described above. The secondary object LUD pointer, when appropriate, is the first pointer within this area.

Format and Contents of the Feedback Record

The message associated with a Dequeue instruction is called a feedback record whenever the message resulted from a request path operation associated with this response queue. The message operand on the Dequeue instruction has the following information inserted into it to form the feedback record:

• Path operation template	Space pointer
• Request ID	Bin(2)
• Error summary field	Bin(2)
• Reserved	Char(4)
• Device dependent status	Char(40)

Definitions of these feedback record fields are as follows:

- Path operation template–This pointer locates the same template supplied by the issuer of the REQPO instruction as its operand. This template can optionally have new data inserted into it based on the path operation which was performed.

- Request ID field–This 2-byte field contains the same value as the request ID field within the template of the Request Path Operation instruction which generated this feedback record for correlation of responses to requests.

- Error summary field–This 2-byte field indicates the final disposition of the request path operation. The contents of this field are:

Byte 0	**Byte 1**
t n i s ffff	nnnnnnnn
0 7	0 7

Byte 0 Error Attributes

Bits 0, 1 (t = terminate, n = not normal)

00 = Normal condition.
01 = Not normal, nonterminating error.
11 = Not normal, terminating error.

Terminating errors are defined as those for which processing of subsequent request path operations is suspended until such time as higher level request path control instructions are requested.

Bit 2 (i = included)

0 = Device-dependent data is not included.
1 = Device-dependent data has been included in the device-dependent status area of this feedback message.

Bit 3 (s = specific error)

0 = Error code defined in byte 1.
1 = Device-specific error code is defined in byte 1, and none of the definitions for byte 1 apply.

Bits 4, 5, 6, 7 (f = function)

0000 = Normal function.
All other values are reserved.

Byte 1 Error Type

There are currently no error codes for byte 1 under control of byte 0, bit 3 as defined above. Byte 0, bit 3 specified as '0'B and all accompanying values for byte 1 are reserved. Byte 0, bit 3 specified as '1'B results in model dependent error codes which are defined in model dependent documentation.

Request Path Operation Time-out Considerations

The Request Path Operation instruction normally initiates asynchronous I/O hardware operations that, under abnormal circumstances or hardware failures, may fail to complete. The REQPO, like the REQIO instruction, does not provide any time-out mechanism for these cases as is provided by the Modify instructions or for the synchronous request I/O. Whenever possible, the user should provide time-out mechanisms for feedback records to prevent these I/O failures from causing indefinite waits or ultimately requiring operator initiated recovery actions.

Since the Request Path Operation instruction execution is asynchronous to the actual hardware operations (that is, the instruction completes before the actual operation is started by the machine), timing must be done on the Dequeue instruction that retrieves the feedback record that signals the actual completion of the I/O operation. This timing can be done by setting a time-out value for the dequeue-wait-time-out parameter on the Dequeue instruction.

Time-out values to be used are device dependent and are a function of the particular I/O operation being performed by that device.

The path operation template for this instruction must not be modified, destroyed, or truncated during the time the machine is using this space to complete request path operation or unpredictable results including device failure or damage event signaling (LUD events) may occur.

Authorization Required

- Operational
  - LUD or dump space specified as requester object in the template
  - LUD specified as secondary object in the template (when appropriately supplied)
- Insert
  - Queue specified in the template (response queue)
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Modification
  - The logical unit description or dump space specified by the system pointer in the template
  - The response queue specified by the system pointer in the template
  - Any secondary system objects specified in the dependent portion of the template

- Object Control
  - None

- Materialize
  - Contexts referenced for address resolution
  - Any system objects specified in the template

Events

0002 Authorization
  0101 Authorization violation

000B Logical unit description
  0905 Request path operation completed (signaled asynchronously)
  0A01 Request path operation response queue destroyed (signaled asynchronously)

0012 Queue
  0401 Queue message limit reached
  0501 Queue extended

000C Machine resources
  0201 Machine auxiliary storage exceeded

000D Machine status
  0101 Machine check

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0201 Machine context
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X				X
02 Boundary alignment	X				X
03 Range	X				X
06 Optimized addressability invalid	X				X
08 Argument/Parameter					
01 Parameter reference violation	X				
0A Authorization					
01 Unauthorized for operation	X				X
04 Unauthorized for process control	X				X
10 Damage Encountered					
02 Machine context damage					X
04 System object damage	X				X
44 Partial system object damage	X				X
1A Lock State					
01 Invalid lock state	X				X
1C Machine Dependent Exception					
03 Machine storage limit exceeded					X
20 Machine Support					
02 Machine check					X
03 Function check					X
22 Object Access					
01 Object not found	X				X
02 Object destroyed	X				X
03 Object suspended	X				X
04 Object not eligible for operation	X				X
05 Object not available to process	X				X
24 Pointer Specification					
01 Pointer does not exist	X				X
02 Pointer type invalid	X				X
03 Pointer addressing invalid object	X				X
26 Process Management					
02 Queue full	X				X
2A Program Creation					
06 Invalid operand type	X				
07 Invalid operand attribute	X				
08 Invalid operand value range	X				
0C Invalid operand ODT reference	X				
0D Reserved bits are not zero	X				X
2C Program Execution					
06 Instruction cancellation					X
32 Scalar Specification					
01 Scalar type invalid	X				
34 Source/Sink Management					
01 Source/sink configuration invalid					X
03 Source/sink object state invalid					X
38 Template Specification					
01 Template value invalid	X				
02 Template size invalid	X				

Chapter 18. Machine Observation Instructions

This chapter describes all instructions used for machine observation. These instructions are arranged alphabetically. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CANCEL INVOCATION TRACE (CANINVTR)

Op Code (Hex)	Operand 1
0581	Trace options

Operand 1: Character(4) scalar.

Description: Based on the options specified in operand 1, this instruction causes the invocation reference event to no longer be signaled as a result of the creation of a new invocation or a return from an existing invocation. The instruction locates a specific invocation by its invocation number and allows cancellation of the trace of either the invocation of subsequent invocations or the return from the referenced invocation. No explicit control exists for simply turning off the propagation status; this is done implicitly by resetting the primary status.

Operand 1 contains the following:

- Trace status — Char(2)
  - Invocation trace — Bit 0
    - 0 = Do not cancel invocation trace
    - 1 = Cancel invocation trace
  - Return trace — Bit 1
    - 0 = Do not cancel return trace
    - 1 = Cancel return trace
- Invocation number — Bin(2)

Any currently existing invocation in the process may be the target of this instruction. No exception is signaled if no trace is in effect for the target invocation.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	X
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist		X
02 Pointer type invalid		X
2A Program Creation		
02 ODT syntax error	X	
04 Operation code invalid		X
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
02 Scalar attributes invalid	X	
03 Scalar value invalid	X	

CANCEL TRACE INSTRUCTIONS (CANTRINS)

Op Code (Hex)	Operand 1	Operand 2
0562	Program	Instruction lists

Operand 1: System pointer.

Operand 2: Space pointer or null.

Description: The instructions specified in operand 2 are removed from the instruction trace of the program referenced by operand 1.

The space pointer identified by operand 2 addresses a list of instructions that are to be removed from the instruction trace. If operand 2 is null, or if the number of instructions referenced is 0, then all instructions currently being traced in the program are removed from the instruction trace. If operand 2 is specified, its format must be as follows:

• Number of instructions referenced (N)	Bin(2)
• Instruction reference 1	Bin(2)
.	
.	
.	
• Instruction reference N	Bin(2)

Instruction references are binary values representing the address (number) of the instruction within the program on which the trace is to be canceled.

Instructions currently being traced but not referenced in the instruction list continue to be traced. References to instructions not currently being traced are ignored.

An exception is signaled if an instruction number that is not in the program being traced is specified.

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X		
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute		X	
08 Invalid operand value range		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid		X	
38 Template Specification			
01 Template value invalid		X	

MATERIALIZE INSTRUCTION ATTRIBUTES (MATINAT)

Op Code (Hex)	Operand 1	Operand 2
0526	Receiver	Selection information

Operand 1: Space pointer.

Operand 2: Character scalar.

Description: This instruction materializes the attributes of the instruction that are selected in operand 2 and places them in the receiver (operand 1).

Operand 2 is a 16-byte template. Only the first 16 bytes are used. Any excess bytes are ignored. Operand 2 has the following format:

• Selection template	Char(16)
– Invocation number	Bin(2)
– Instruction number	Bin(4)
– Reserved (binary 0)	Char(10)

The invocation number is a specific identifier for the target invocation, in the process, that is to be materialized. This program must be observable or the program not observable exception is signaled.

The instruction number specifies the instruction in the specified program invocation that is to be materialized.

Operand 1 is a space pointer that addresses a 16-byte aligned template where the materialized data is placed. The format of the data is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Object identification	Char(32)
– Program type	Char(1)
– Program subtype	Char(1)
– Program name	Char(30)
• Offset to instruction attributes	Bin(4)
• Reserved (binary 0)	Char(8)
• Instruction attributes	Char(*)
– Instruction type	Char(2)
– Instruction version Hex 0000=2-byte operand references Hex 0001=3-byte operand references	Bits 0-3
– Reserved (binary 0)	Bits 4-15
– Instruction length as input to Create Program	Bin(2)
– Offset to instruction form specified as input to Create Program	Bin(4)
– Reserved (binary 0)	Char(4)
– Number of instruction operands	Bin(2)
– Operand attributes offsets	Char(*)
– An offset is materialized for each of the operands of the instruction specifying the offset to the attributes for the operand	Bin(4)
• Instruction form specified as input to Create Program	Char(*)
– Instruction operation code	Char(2)
– Optional extender field and operand fields	Char(*)
• Operand attributes A set of attributes following this format is materialized for each of the operands of the instruction. Compound operand references result in materialization of only one set of attributes for the operand which describe the substring or array element as is appropriate. See the specific format described below for each operand type.	Char(*)
– Operand type 1 = Data object 2 = Constant data object 3 = Instruction number reference 4 = Argument list 5 = Exception description 6 = Null operand 7 = Space pointer machine object	Bin(2)
– Operand specific attributes See descriptions below for detailed formats. Nothing is provided for null operands.	Char(*)

• Data object	Char(32)
For a data object, the following operand attributes are materialized.	
– Operand type = 1	Bin(2)
– Data object specific attributes	Char(7)
Element type	Char(1)
Hex 00=Binary	
Hex 01=Floating-point	
Hex 02=Zoned decimal	
Hex 03=Packed decimal	
Hex 04=Character	
Hex 08=Pointer	
Element length	Char(2)
If binary, or character, or floating-point:	
Length	Bits 0-15
If zoned decimal or packed decimal:	
Fractional digits	Bits 0-7
Total digits	Bits 8-15
If pointer:	
Length = 16	Bits 0-15
Array size	Bin(4)
If scalar, then value of 0.	
If array, then number of elements.	
– Reserved (binary 0)	Char(6)
– Data object addressability	Char(17)
Addressability indicator	Char(1)
Hex 00=Addressability was not established	
Hex 01=Addressability was established	
Space pointer to the object if addressability could be established	Space pointer

• Constant data object	Char(*)
For a constant data object, the following operand attributes are materialized (immediate operands as constants, signed immediates as binary, and unsigned immediates as character).	
– Operand type = 2	Bin(2)
– Constant specific attributes	Char(7)
Element type	Char(1)
Hex 00=Binary	
Hex 01=Floating-point	
Hex 02=Zoned decimal	
Hex 03=Packed decimal	
Hex 04=Character	
Element length	Char(2)
If binary, or character, or floating-point:	
Length	Bits 0-15
If zoned decimal or packed decimal:	
Fractional digits	Bits 0-7
Total digits	Bits 8-15
Reserved (binary 0)	Bin(4)
– Reserved (binary 0)	Char(7)
– Constant value	Char(*)

• Instruction references	Char(*)
For instruction references, either through instruction definition lists or immediate operands, the following operand attributes are materialized.	
– Operand type = 3	Bin(2)
– Number of instruction reference elements	Bin(2)
1 = Single instruction reference	
>1 = Instruction definition list	
– Reserved (binary 0)	Char(12)
– Reference list	Char(*)
The instruction number of each instruction reference is materialized in the order in which they are defined.	

• Argument list	Char(*)
For an argument list, the following operand attributes are materialized.	
– Operand type = 4	Bin(2)
– Argument list specific attributes	Char(4)
Actual number of list entries	Bin(2)
Maximum number of list entries	Bin(2)
– Reserved (binary 0)	Char(10)
– Addressability to list entries	Char(*)
Space pointer to each list entry for the number of actual list entries. A value of all zeros is materialized if addressability could not be established.	Space pointer

• Exception description	Char(48)
For an exception description, the following operand attributes are materialized.	
– Operand type = 5	Bin(2)
– Reserved (binary 0)	Char(10
– Control flags	Char(2)
Exception handling action	Bits 0-2
000 = Ignore occurrence of exception and continue processing	
001 = Disabled exception description	
010 = Continue search for an exception description by resignaling the exception to the immediately preceding invocation	
100 = Defer handling	
101 = Pass control to the specified exception handler	
Reserved (binary 0)	Bits 3-15
– Compare value length	Bin(2)
– Compare value	Char(32)

• Space pointer machine object	Char(32)
For a space pointer machine object, the following operand attributes are materialized.	
– Operand type = 7	Bin(2)
– Reserved (binary 0)	Char(13)
– Pointer addressability	Char(17)
– Pointer value indicator	Char(1)
Hex 00= Addressability value is not valid	
Hex 01= Addressability value is valid	
– Space pointer data object containing the space pointer machine object value if addressability value is valid.	Space pointer

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then excess bytes are unchanged.

The materialization available for an instruction depends on the execution status of the program that the instruction is in. If the program has not executed to the point of the instruction, little or no meaningful information about the instruction can be materialized. If the program executes the instruction multiple times, the materialization will vary with each execution.

No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
- 0201 Machine auxiliary storage threshold exceeded

000D Machine status
- 0101 Machine check

0010 Process
- 0601 Exception signaled to process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference
- 0201 Object location reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage			X
44 Partial system object damage			X
1E Machine Observation			
01 Program not observable		X	
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found		X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid	X	X	
03 Scalar value invalid	X	X	
38 Template Specification			
01 Template value invalid		X	
03 Materialization length exception	X		

MATERIALIZE INVOCATION (MATINV)

Op Code (Hex)	Operand 1	Operand 2
0516	Receiver	Selection information

Operand 1: Space pointer.

Operand 2: Space pointer.

Description: The attributes of the invocation selected through operand 2 are materialized into the receiver designated by operand 1.

Operand 2 is a space pointer that addresses a template of the following form:

• Invocation number	Bin(2)
• Offset to list of parameters	Bin(4)
• Number of parameter ODV numbers	Char(2)
• Offset to list of exception descriptions	Bin(4)
• Number of exception description ODV (object definition table) numbers	Char(2)

The offset to the list of parameters and the offset to the list of exception descriptions are both relative to the start of the operand 2 template. Each list is an array of Char(2) ODV numbers. The number of parameter ODV numbers and the number of exception description ODV numbers define the sizes of the arrays.

Operand 2 is a space pointer that addresses a template that has the following format:

• Control information	Char(2)
– Template extension 0 = Template extension is not present. 1 = Template extension is present.	Bit 0
– Invocation number	Bits 1-15
• Offset to list of parameters	Bin(4)
• Number of parameter ODV numbers	Char(2)
• Offset to list of exception descriptions	Bin(4)
• Number of exception description ODV numbers	Char(2)
• Template extension (optional)	Char(14)
– Offset to list of space pointer machine objects	Bin(4)
– Number of space pointer machine object ODV numbers	Char(2)
– Reserved (binary 0)	Char(8)

The offset to the list of space pointer machine objects, offset to the list of parameters, and the offset to the list of exception descriptions are relative to the start of the operand 2 template. Each list is an array of Char(2) ODV numbers. The number of space pointer machine object ODV numbers, number of parameter ODV numbers, and the number of exception description ODV numbers define the sizes of the arrays.

Operand 1 is a space pointer that addresses a 16-byte aligned template into which the materialized data is placed. The format of the data is:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Object identification	Char(32)
– Program type	Char(1)
– Program subtype	Char(1)
– Program name	Char(30)
• Trace specification	Char(2)
– Invocation trace status 0 = Not tracing new invocations 1 = Tracing new invocations	Bit 0
– Return trace 0 = Not tracing returns 1 = Tracing returns	Bit 1
– Invocation trace propagation 0 = Not propagating invocation trace 1 = Propagating invocation trace	Bit 2
– Return trace propagation 0 = Not propagating return trace 1 = Propagating return trace	Bit 3
– Reserved (binary 0)	Bits 4-15
• Instruction number	Bin(2)

- Offset to parameter values Bin(4)

- Offset to exception description values Bin(4)

- Offset to space pointer machine object values (Optional–This data is present only if the template extension is present in the selection information.) Bin(4)

- Space pointer machine objects (Optional–This data is present only if the template extension is present in the selection information.) Char(*)
  - For each ODV number specified for a space pointer machine object, the value of the space pointer machine object is materialized as follows: Char(32)
    - Reserved (binary 0) Char(15)
    - Pointer value indicator Char(1)
      - 00 = Addressability value is not valid
      - 01 = Addressability value is valid
    - Space pointer data object containing the space pointer machine object value if addressability value is valid. Space pointer

- Parameters Char(*)
  - For each parameter ODT number specified, the address of the parameter data is materialized (If no parameter ODT numbers are materialized, this parameter is binary 0.) Space pointer

- Exception description Char(*)
  - For each exception description ODT number specified, the following is materialized: Char(36)
  - Control flags Char(2)
    - Exception handling action Bits 0-2
      - 000 = Ignore occurrence of exception and continue processing
      - 001 = Disabled exception description
      - 010 = Continue search for an exception description by resignaling the exception to the immediately preceding invocation
      - 100 = Defer handling
      - 101 = Pass control to the specified exception handler
    - Reserved (binary 0) Bits 3-15
  - Compare value length Bin(2)
  - Compare value Char(32)

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then excess bytes are unchanged.

No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The instruction number returned depends on how control was passed from the invocation:

Exit Type	Instruction Number
Call External	Locates the Call External instruction
Event	Locates the next instruction to execute
Exception	Locates the instruction that caused the exception

The space pointers that address parameter values are returned in the same order as the corresponding ODT numbers in the input array. The same is true for the exception description values.

If the offset to the list of parameters or the number of parameter ODT numbers is 0, no parameters are returned and the offset to parameters value is 0. If any parameters are returned, they are 16-byte aligned. If the offset to list of exception descriptions or the number of exception description ODT numbers is 0, no exception descriptions are returned and the offset to exception description values are 0.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1E Machine Observation			
01 Program not observable			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found		X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid	X	X	
38 Template Specification			
01 Template value invalid		X	
03 Materialization length exception	X		

MATERIALIZE INVOCATION ENTRY (MATINVE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0547	Receiver	Selection information	Materialization options

Operand 1: Character variable scalar.

Operand 2: Character(8) scalar or null.

Operand 3: Character(1) scalar or null.

Description: This instruction materializes the attributes of the specified invocation entry within the process issuing the instruction. The attributes specified by operand 3 of the invocation selected through operand 2 are materialized into the receiver designated by operand 1.

Operand 2 is an 8-byte template or a null operand. If operand 2 is null, it indicates that the attributes of the current invocation are to be materialized. If operand 2 is not null, it must be an 8-byte template which specifies the invocation to be materialized. Only the first 8 bytes are used. Any excess bytes are ignored. It has the following format:

• Selection information	Char(8)
– Relative invocation number	Char(2)
– Reserved	Char(6)

If operand 2 is not null, it is restricted to a constant with the relative invocation number field specifying a value of zero, which indicates that the attributes of the current invocation are to be materialized.

Operand 3 is a 1-byte value or a null operand. If operand 3 is null, it indicates that the attributes for a materialization option value of hex 00 are to be materialized. If operand 3 is not null, it must be a 1-byte value which specifies the type of materialization to be performed. Option values that are not defined below are reserved values and may not be specified. Only the first byte is used. Any excess bytes are ignored. It has the following format:

- Materialization options Char(1)
  - Hex 00=Long materialization
  - Hex 01=Short materialization type 1
  - Hex 02=Short materialization type 2
  - Hex 03=Short materialization type 3
  - Hex 04=Short materialization type 4

If operand 3 is not null, it is restricted to a constant character scalar or an immediate value.

Operand 1 specifies a receiver into which the materialized data is placed. It must specify a character scalar with a minimum length which is dependent upon the materialization option specified for operand 3. If the length specified for operand 1 is less than the required minimum, an exception is signaled. Only the bytes up to the required minimum length are used. Any excess bytes are ignored. For the materialization options which produce pointers in the materialized data, 16-byte space alignment is required for the receiver. The data placed into the receiver differs depending upon the materialization option specified. The following descriptions detail the formats of the optional materializations.

Long Materialization

For a materialization option value of hex 00, the minimum length for the receiver is 144 bytes. It has the following format:

Hex 00=Long materialization	Char(144)
• Reserved	Char(12)
• Mark counter	Bin(4)
• Reserved	Char(32)
• Associated program pointer (zero for data base select/omit program)	System pointer
• Invocation number	Bin(2)
• Invocation type	Char(1)

Hex 00=Data base select/omit program
Hex 01=Call external
Hex 02=Transfer control
Hex 03=Event handler
Hex 04=External exception handler
Hex 05=Initial program in process problem state
Hex 06=Initial program in process initiation state
Hex 07=Initial program in process termination state
Hex 08=Invocation exit

• Reserved (binary 0)	Char(1)
• Invocation mark	Bin(4)
• Reserved	Char(8)
• PASA entry pointer	Space pointer
• PSSA entry pointer	Space pointer
• Reserved	Char(32)

Short Materialization Type 1

For a materialization option value of hex 01, the minimum length for the receiver is 16 bytes. It has the following format:

Hex 01=Short materialization type 1	Char(16)
• Associated program pointer (null for data base select/omit program)	System pointer

Short Materialization Type 2

For a materialization option value of hex 02, the minimum length for the receiver is 4 bytes. It has the following format:

Hex 02=Short materialization type 2	Char(4)
• Invocation mark	Bin(4)

Short Materialization Type 3

For a materialization option value of hex 03, the minimum length for the receiver is 16 bytes. It has the following format:

Hex 03=Short materialization type 3	Char(16)
• PASA entry pointer	Space pointer

Short Materialization Type 4

For a materialization option value of hex 04, the minimum length for the receiver is 16 bytes. It has the following format:

Hex 04=Short materialization type 4	Char(16)
• PSSA entry pointer	Space pointer

The mark counter value represents the current value of a counter used by the machine to mark all activations and invocations created during the execution of a process with a unique value. This mark indicates the point at which the specific entry was allocated relative to the sequence of all activations and invocations that have been created over time within the process.

The associated program pointer is a system pointer that locates the program associated with the invocation entry.

The invocation number is a number that uniquely identifies each invocation in the PASA. When an invocation is allocated, the invocation number of the new invocation entry is one more than that in the calling invocation. The first invocation in the current process state has an invocation number of one.

The invocation type indicates how the associated program was invoked.

The invocation mark indicates the point at which this invocation entry was allocated relative to the sequence of all activations and invocations that have been created over time within the process. This is set from the incremented mark counter value for each new invocation added to the invocation stack.

The PASA entry pointer is a space pointer that is set to address the start of the PASAE (program automatic storage area entry) associated with the invocation. The associated program's automatic data starts 64 bytes after the area addressed by this pointer.

The PSSA entry pointer is a space pointer that is set to address the start of the PSSAE (program static storage area entry) associated with the invocation. The associated program's static data starts 64 bytes after the area addressed by this pointer. The first 64 bytes contain the header information for the PSSAE. Refer to the Create Activation instruction for a description of this header information. This pointer will be set to a value of all zeros if the invoked program does not have static data.

The fields labeled reserved in the descriptions of the optional materializations are currently reserved for future use. These fields may be altered by this instruction depending upon the particular implementation of the machine. Any values set into these fields are meaningless.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	**Operands**			**Other**
	1	**2**	**3**	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage				X
44 Partial system object damage				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range		X	X	
0A Invalid operand length	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid	X	X	X	
03 Scalar value invalid	X	X	X	

MATERIALIZE INVOCATION STACK (MATINVS)

Op Code (Hex)	Operand 1	Operand 2
0546	Receiver	Process

Operand 1: Space pointer.

Operand 2: System pointer or null.

Description: This instruction materializes the current invocation stack within the specified process.

The attributes of the invocation entries currently on the invocation stack of the process specified by operand 2 are materialized into the template specified by operand 1.

Operand 2 is a system pointer or a null operand. If operand 2 is null, it indicates that the invocation stack of the current process is to be materialized. If operand 2 is not null, it is a system pointer identifying the process control space associated with the process for which the invocation stack is to be materialized. If the subject process, identified by operand 2, is different from the process executing this instruction, the executing process must be the original initiator of the subject process or must have process control special authorization to the process control space associated with the subject process.

Operand 1 is a space pointer that addresses a 16-byte aligned template into which is placed the materialized data. The format of the data is:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Number of invocation entries	Bin(4)
• Mark counter	Bin(4)
• Invocation entries (An invocation entry is materialized for each of the invocations currently on the invocation stack of the specified process.)	Char(*)

The invocation entries materialized are each 128 bytes long and have the following format:

• Reserved	Char(32)
• Associated program pointer (null for data base select/omit program or a destroyed program)	System pointer
• Invocation number	Bin(2)
• Invocation type Hex 00=Data base select/omit program Hex 01=Call external Hex 02=Transfer control Hex 03=Event handler Hex 04=External exception handler Hex 05=Initial program in process problem state Hex 06=Initial program in process initiation state Hex 07=Initial program in process termination state Hex 08=Invocation exit	Char(1)
• Reserved	Char(1)
• Invocation mark	Bin(4)
• Instruction number	Bin(4)
• Reserved	Char(68)

The number of invocations value specifies the number of invocation entries provided in the materialization.

The mark counter value represents the current value of a counter used by the machine to mark all activations and invocations created during the execution of a process with a unique value. This mark indicates the point at which the specific entry was allocated relative to the sequence of all activations and invocations that have been created over time within the process.

The associated program pointer is a system pointer that locates the program associated with the invocation entry.

The invocation number is a number that uniquely identifies each invocation in the PASA. When an invocation is allocated, the invocation number of the new invocation entry is one more than that in the calling invocation. The first invocation in the current process state has an invocation number of one.

The invocation type indicates how the associated program was invoked.

The invocation mark indicates the point at which this invocation entry was allocated relative to the sequence of all activations and invocations that have been created over time within the process. This is set from the incremented mark counter value for each new invocation added to the invocation stack.

The instruction number specifies the number of the instruction last being executed when the invocation passed control to the next invocation on the stack.

The fields labeled reserved are currently reserved for future use. These fields may be altered by this instruction depending upon the particular implementation of the machine. Any values set into these fields are meaningless.

The first 4 bytes of the materialization identifies the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identifies the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, the excess bytes are unchanged.

No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

When the materialization is performed for a process other than the one executing this instruction, the instruction attempts to interrogate, snapshot, the invocation stack of the other process concurrently with the ongoing execution of that process. In this case, the interrogating process and subject process may be interleaving usage of the processor resource. Due to this, the accuracy and integrity of the materialization is relative to the state, static or dynamic, of the invocation stack in the subject process over the time of the interrogation. If the invocation stack in the subject process is in a very static state, not changing over the period of interrogation, the materialization may represent a good approximation of a snapshot of its invocation stack. To the contrary, if the invocation stack in the subject process is in a very dynamic state, radically changing over the period of interrogation, the materialization is potentially totally inaccurate and may describe a sequence of invocations that was never an actual sequence that occurred within the process. In addition to the above exposures to inaccuracy in attempting to take the snapshot, the ongoing status of the invocation stack of the subject process may substantially differ from that reflected in the materialization, due to its continuing execution after completion of this instruction.

When the materialization is performed for the process executing this instruction, it does provide an accurate reflection of the status of the process' invocation stack. In this case, concurrent execution of this instruction with execution of other instructions in the process is precluded.

Authorization Required

- Process control special authorization
  - For materializing a different process than the one executing this instruction

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution

Events

000C Machine resources
  0201 Machine auxiliary storage exceeded

000D Machine status
  0101 Machine check

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage Encountered			
04 System object damage			X
44 Partial system object damage			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
28 Process State			
02 Process control space not associated with a process		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid	X	X	
38 Template Specification			
03 Materialization length	X		

MATERIALIZE POINTER (MATPTR)

Op Code (Hex)	Operand 1	Operand 2
0512	Receiver	Pointer

Operand 1: Space pointer.

Operand 2: System pointer, space pointer data object, data pointer, or instruction pointer.

Description: The materialized form of the pointer object referenced by operand 2 is placed in operand 1.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The format of the materialization is:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Pointer type	Char(1)
Hex 01 = System pointer	
Hex 02 = Space pointer	
Hex 03 = Data pointer	
Hex 04 = Instruction pointer	

Pointer value materialization depends on the pointer type. One of the following pointer type formats is used.

• System pointer description	Char(66)

The system pointer description identifies the object addressed by the pointer and the context which the object specifies as its addressing context.

– Context identification	Char(32)
Context type	Char(1)
Context subtype	Char(1)
Context name	Char(30)
– Object identification	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Pointer authorization	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorization pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Ownership	Bit 8
Reserved (binary 0)	Bits 9-15

Note: If the object addressed by the system pointer specifies that it is not addressed by a context or if the context is destroyed, the context entry is hex 00. If the object is addressed by the machine context, a context type entry of hex 81 is returned. No verification is made that the specified context actually addresses the object.

The following lists the object type codes for system object references:

Value (Hex)	Object Type
01	Access group
02	Program
04	Context
07	Journal space
08	User profile
09	Journal port
0A	Queue
0B	Data space
0C	Data space index
0D	Cursor
0E	Index
0F	Commit block
10	Logical unit description
11	Network description
12	Controller description
19	Space
1A	Process control space

Note: Only the authority currently stored in the system pointer is materialized.

- Data pointer description Char(75)

  The data pointer description describes the current scalar and array attributes and identifies the space addressability contained in the data pointer.

– Scalar and array attributes	Char(7)
Scalar type	Char(1)
Hex 00 = Binary	
Hex 01 = Floating-point	
Hex 02 = Zoned decimal	
Hex 03 = Packed decimal	
Hex 04 = Character	
Scalar length	Char(2)
If binary, character, or floating-point:	
Length	Bits 0-15
If zoned decimal or packed decimal:	
Fractional digits	Bits 0-7
Total digits	Bits 8-15
Reserved (binary 0)	Bin(4)
– Data pointer space addressability	Char(68)
Context identification	Char(32)
Context type	Char(1)
Context subtype	Char(1)
Context name	Char(30)
Object identification	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
Offset into space	Bin(4)

Note: If the object containing the space addressed by the data pointer is not addressed by a context, the context entry is hex 00. If the object is addressed by the machine context, a context type entry of hex 81 is returned.

• Space pointer description	Char(68)

The space pointer description describes space addressability contained in the space pointer.

– Context identification	Char(32)
Context type	Char(1)
Context subtype	Char(1)
Context name	Char(30)
– Object identification	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Offset into space	Bin(4)

Note: If the object containing the space addressed by the space pointer is not addressed by a context, the context entry is hex 00. If the object is addressed by the machine context, a context type entry of hex 81 is returned.

- Instruction pointer description

The instruction pointer description describes instruction addressability contained in the instruction pointer.

– Context identification	Char(32)
Context type	Char(1)
Context subtype	Char(1)
Context name	Char(30)
– Program identification	Char(32)
Program type	Char(1)
Program subtype	Char(1)
Program name	Char(30)
– Instruction number	Bin(4)

If the program containing the instruction currently being addressed by the instruction pointer is not addressed by a context, the context entry is hex 00.

If the pointer is a system pointer or a data pointer and is initialized but unresolved, the pointer is resolved before the materialization occurs.

This instruction will tolerate a damaged object referenced by operand 2 when operand 2 is a resolved pointer. The instruction will not tolerate a damaged context(s) or damaged programs when resolving pointers. Also, as a result of damage or abnormal machine termination, this instruction can indicate that an object is addressed by a context, when in fact the context will not show this as an addressed object. The Modify Addressability instruction can be used to correct this problem.

A space pointer machine object cannot be specified for operand 2.

Events

000C Machine resource
 0201 Machine auxiliary storage threshold exceeded

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
04 External data object not found		X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X		
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE POINTER LOCATIONS (MATPTRL)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0513	Receiver	Source	Length

Operand 1: Space pointer.

Operand 2: Space pointer.

Operand 3: Binary scalar.

Description: This instruction finds the pointers in a subset of a space and produces a bit mapping of their relative locations.

The area addressed by the operand 2 space pointer is scanned for a length equal to that specified in operand 3. A bit in operand 1 is set for each 16 bytes of operand 2. The bit is set to binary 1 if a pointer exists in the operand 2 space, or the bit is set to binary 0 if no pointer exists in the operand 2 space.

Operand 1 is a space pointer addressing the receiver area. One bit of the receiver is used for each 16 bytes specified by operand 3. If operand 3 is not a 16-byte multiple, then the bit position in operand 1 that corresponds to the last (odd) bytes of operand 2 is set to 0. Bits are set from left to right (bit 0, bit 1,...) in operand 1 as 16-byte areas are interrogated from left to right in operand 2. The number of bits set in the receiver is always a multiple of 8. Those rightmost bits positions that do not have a corresponding area in operand 2 are set to 0.

The format of the operand 1 receiver is:

• Template size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Pointer locations	Char(*)

Operand 2 must address a 16-byte aligned area; otherwise, a boundary alignment exception is signaled. If the value specified by operand 3 is not positive, the scalar value invalid exception is signaled.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for materialization.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
10 Damage Encountered				
04 System object damage state	X	X	X	X
44 Partial system object damage	X	X	X	X
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found			X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length			X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception			X	

MATERIALIZE SYSTEM OBJECT (MATSOBJ)

Op Code (Hex)	**Operand 1**	**Operand 2**
053E	Receiver	Object

Operand 1: Space pointer.

Operand 2: System pointer.

Description: This instruction materializes the identity and size of a system object addressed by the system pointer identified by operand 2. It can be used whenever addressability to a system object is contained in a system pointer.

The first 4 bytes of the materialization identify the total number of bytes that may be caused by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 raises the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The format of the materialization is:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Object state attributes	Char(2)
– Suspended state	Bit 0
0 = Not suspended	
1 = Suspended	
– Damage state	Bit 1
0 = Not damaged	
1 = Damaged	
– Partial damage state	Bit 2
0 = No partial damage	
1 = Partial damage	
– Existence of addressing context	Bit 3
0 = Not addressed by a temporary context	
1 = Addressed by a temporary context	
– Reserved (binary 0)	Bits 4-15
• Context identification	Char(32)
– Context type	Char(1)
– Control subtype	Char(1)
– Context name	Char(30)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Timestamp of creation	Char(8)
• Size of associated space	Bin(4)
• Object size	Bin(4)
• Owning user profile identification	Char(32)
– User profile type	Char(1)
– User profile subtype	Char(1)
– User profile name	Char(30)
• Timestamp of last modification	Char(8)

The timestamp field is materialized as an 8-byte unsigned binary number in which bit 41 is equal to 1024 microseconds. The timestamp of creation field is implicitly set when an object is created.

The timestamp of last modification field is explicitly set by the Modify System Object instruction or by any operation that modifies or attempts to modify an object attribute value or an object state. When implicitly set, the timestamp of last modification field is ensured as part of the normal ensuring of objects. The timestamp of last modification field can also be implicitly updated for some IMPL functions to the last time known to the system.

If the object addressed by the system pointer specifies that it is not addressed by a context or if the context is destroyed, the context type entry is hex 00. If the object is addressed by the machine context, a context type entry of hex 81 is returned. No verification is made that the specified context actually addresses the object.

If the object is a temporary object and is, therefore, owned by no user profile, the user profile type entry is assigned a value of hex 00.

This instruction will tolerate a damaged object referenced by operand 2 when operand 2 is a resolved pointer. The instruction will not tolerate a damaged context(s) or damaged programs when resolving pointers. Also, as a result of damage or abnormal machine termination, this instruction can indicate that an object is addressed by a context, when in fact the context will not show this as an addressed object. The Modify Addressability instruction can be used to correct this problem. The existence of addressing context attribute indicates whether the previously (or currently) addressing context was (is) temporary. This field is 0 if the object was (is) not addressed by a temporary context.

Valid object type fields and their meanings are:

Value (Hex)	**Object Type**
01	Access group
02	Program
04	Context
07	Journal space
08	User profile
09	Journal port
0A	Queue
0B	Data space
0C	Data space index
0D	Cursor
0E	Index
0F	Commit block
10	Logical unit description
11	Network description
12	Controller description
19	Space
1A	Process control space

Authorization Required

- Retrieve
  – Contexts referenced for address resolution

Lock Enforcement

- Materialize
  – Operand 2
  – Contexts referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorization for operation		X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
03 Materialization length exception	X		

MODIFY SYSTEM OBJECT (MODSOBJ)

Op Code (Hex)	Operand 1	Operand 2
053E	Target object	Modification options

Operand 1: System pointer.

Operand 2: Character(16) scalar.

Description: Based on the options specified by operand 2, selected state attributes of the system object addressed by the system pointer identified by operand 1 are modified.

• Object state modification options	Char(16)
– Modification selection	Char(1)
Modification timestamp	Bit 0
Reserved (binary 0)	Bits 1-7
– Reserved (binary 0)	Char(15)

The modification timestamp option, if set, indicates that the modification timestamp attribute of the object is to be set to the 8-byte unsigned binary number for the current time-of-day clock value, where bit 41 is equal to 1024 microseconds. This field is also implicitly set for certain instructions that alter an object attribute value or a state of an object. Only the first 16 bytes of the modification option operand are used.

Implicit changing of the modification timestamp will only occur through common functions for the following objects:

- Logical unit description
- Controller description
- Network description
- Access group
- Queue

No modification timestamp is provided for a process control space and a value of zero will be returned in the materialization template.

Only the first 16 bytes of the modification option operand are used.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Object management
  - Operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution

- Modification
  - Operand 1

Events

0002 Authorization
    0101 Authorization violation

000C Machine resource
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
10 Damage Encountered			
04 System object damage		X	X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state	X		
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	

TRACE INSTRUCTIONS (TRINS)

Op Code (Hex)	Operand 1	Operand 2
0552	Program	Instruction list

Operand 1: System pointer.

Operand 2: Space pointer or null.

Description: This instruction causes the execution of the program referenced by operand 1 within the current process monitored for specific instruction executions. When one of the instructions specified by operand 2 starts execution, an instruction reference event is signaled. The event is signaled before any operands of the instruction are accessed.

The space pointer identified by operand 2 addresses an area that defines the instructions to be traced in a format as follows:

• Number of instructions to be traced (N)	Bin(2)
• Instruction reference 1	Bin(2)
.	
.	
.	
• Instruction reference N	Bin(2)

The value of each instruction reference is interpreted as the address (number) of an instruction to be traced. If a value of 0 is specified for the number of instructions to be traced entry or if operand 2 is null, all program instructions are traced.

A template value invalid exception is signaled if any specified instruction number is not in the program being traced. If instructions in the referenced program are already being traced, the instructions referenced in operand 2 are added to those being traced. References to instructions already being traced are ignored.

Any number of programs may be traced within the process at the same time.

This instruction may not be performed in a process when a service machine trace is in progress for the process. A machine-dependent request invalid exception (hex 1C01) is signaled. The exception is also signaled if the service machine trace is requested when the trace instruction is in progress.

Authorization Required

- Retrieve
  - Operand 1
  - Context referenced for address resolution

Lock Enforcement

- Materialize
  - Context referenced for address resolution

Events

0002 Authorization
    0101 Object authorization violation

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X		
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
01 Machine dependent-request invalid	X		
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid		X	
38 Template Specification			
01 Template value invalid		X	

TRACE INVOCATIONS (TRINV)

Op Code (Hex)	**Operand 1**
0551	Trace specification

Operand 1: Character(4) scalar.

Description: The instruction causes the invocation reference event to be signaled upon invocation of a program or upon termination of the invocation of a program. The following conditions may be traced:

- Call external
- Transfer control
- Invocation of an external exception handler
- Invocation of an event handler
- Invocation of an internal or branch point exception handler
- Control given to an invocation exit routine
- Return external
- Return from exception
- Termination of an invocation to pass control to an internal exception handler or to a branch point exception handler in a previous invocation
- Termination of an invocation to pass control from an external exception handler to an invocation other than the invocation in which the exception occurred
- Termination of an invocation to terminate a phase of a process

This instruction references only a single invocation within the process and causes the invocation reference event to be signaled when that invocation returns or when an invocation subsequent to it is created. The instruction also allows the trace control attributes to be propagated to subsequently created invocations. Currently existing invocations within a process may be designated through multiple executions of this instruction. Specification of trace propagation in a currently existing invocation does not cause propagation to other currently existing invocations.

Operand 1 contains the following information:

• Trace specification	Char(2)
– Invocation trace	Bit 0
0 = Do not cancel new invocations	
1 = Trace new invocations	
– Return trace	Bit 1
0 = Do not cancel trace return	
1 = Trace returns	
– Trace propagation	Bit 2
0 = Do not propagate trace to subsequent invocations	
1 = Propagate trace to subsequent invocations	
– Reserved (binary 0)	Bits 3-15
• Invocation number	Bin(2)

If the referenced invocation is currently being traced, the invocation trace, the return trace, or both may be added. No exception is signaled if either or both are currently being traced. If propagation of the trace control indicator to lower-level invocations is desired, then trace new invocations, trace return, or both must also be set. The propagated trace applies only to the trace action specified by this instruction, not to the current trace action in the referenced invocation.

Propagating of trace to a lower-level invocation means that any immediately subordinate invocations that are created have trace controls that are identical to those of the designated invocation. The only exception is that the invocation trace is not propagated to an invocation reference event handler.

On transfer control conditions, the new invocation overlays the old invocation, and the invocation reference event is signaled if either the trace new invocations or the trace returns option is in effect.

When the initial invocation in a process phase returns, the initial program in the next process phase is invoked, and the trace status of the returning invocation becomes the trace status of the new invocation.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Events

000C Machine resource
    0201 Machine auxiliary storage threshold exceeded

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	**Operand 1**	**Other**
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage Encountered		
04 System object damage state	X	X
44 Partial system object damage	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	X
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
02 ODT syntax error	X	
04 Operation code invalid		X
06 Invalid operand type	X	
07 Invalid operand attribute	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
02 Scalar attributes invalid	X	
03 Scalar value invalid	X	

Chapter 19. Machine Interface Support Functions Instructions

This chapter describes all instructions used for machine interface support functions. These instructions are arranged in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

DIAGNOSE (DIAG)

Op Code (Hex)	**Operand 1**	**Operand 2**
0672	Function code	Function-dependent information

Operand 1: Binary scalar.

Operand 2: Space pointer.

Description: This instruction invokes diagnostic functions and is intended for use by personnel who service System/38. Each function has a separate and unique purpose and is identified by the value in operand 1. Operand 2 identifies a template that contains either information specified for the function or information to be received from the function.

The instruction is a privileged instruction and its use must be authorized to the user profile under which it is executing.

Authorization Required

- Privileged instruction

Events

0002 Authorization
 0201 Privileged instruction violation

000C Machine resources
 0201 Machine auxiliary storage exceeded

000D Machine Status
 0101 Machine check

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
02 Privileged instruction			X
10 Damage Encountered			
04 System object damage state			X
44 Partial system object damage			X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
01 Diagnose	X	X	
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
03 Scalar value invalid	X		
38 Template Specification			
01 Template value invalid		X	

MATERIALIZE MACHINE ATTRIBUTES (MATMATR)

Op Code (Hex)	Operand 1	Operand 2
0636	Materialization	Machine attributes

Operand 1: Space pointer.

Operand 2: Character(2) scalar (fixed-length).

Description: The instruction makes available the unique values of machine attributes. The values of various machine attributes are placed in the receiver. Operand 2 options specify the type of information to be materialized.

The machine attributes are divided into nine groups. Byte 0 of the attribute selection operand specifies the group from which the machine attributes are to be materialized. Byte 1 of the options operand selects a specific subset of that group of machine attributes.

The first 4 bytes of the materialization (operand 1) identify the total number of bytes that can be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested for materialization, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

Data-pointer-defined scalars are not allowed as a primary operand for this instruction. An invalid operand type exception is signaled if this occurs.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

The format of the materialization is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Attribute specification (as defined by the attribute selection)	Char(*)

The machine attributes defined by operand 2 are materialized according to the following selection values:

Selection Value **Attribute Description**

Hex 0000 MCR (machine configuration record)

The MCR contains the internal configuration of the machine. The MCR machine attribute is provided for machine maintenance only and has no meaning or value to the user. The MCR is materialized as a contiguous character string of binary data.

Hex 0004 Machine serial identification (can be materialized but cannot be modified)

The machine serial identification that is materialized is an 8-byte character field that contains the unique machine identifier.

Hex 0100 Time-of-day clock (can be materialized and modified)

The time-of-day clock provides a consistent measure of elapsed time. The maximum elapsed time the clock can indicate is approximately 143 years.

The time-of-day clock is a 64-bit unsigned binary counter with the following format:

0................41 42 reserved 63

The bit positions of the clock are numbered from 0 to 63.

Selection Value **Attribute Description**

Hex 0100 (continued)

The clock is incremented by adding a 1 in bit position 41 every 1024 microseconds. Bit positions 42 through 63 are used by the machine and have no special meaning to the user. Note that these bits (42-63) may contain either binary 1's or binary 0's.

Unpredictable results occur if the time of day is materialized before it is set.

The maximum unsigned binary value that the time of day clock can be modified to contain is hex DFFFFFFFFFFFFFFF.

Hex 0104 Initial process definition template (can be materialized and modified)

The initial process definition template is used by the machine to perform an initial process load. The initial process definition template has the same format as the process definition template defined by the Initiate Process instruction. See Chapter 11. *Process Management Instructions*.

No check is made and no exception is signaled if the values in the template are invalid; however, the next initial process load will not be successful.

Hex 0108 Machine initialization status record (can be materialized and modified)

The MISR (machine initialization status record) is used to report the status of the machine. The status is collected at IMPL (initial microprogram load) or IMPLA (initial microprogram load abbreviated).

Selection Value **Attribute Description**

Hex 0108 (continued)

Modifying the MISR causes it to be reset. The values in the operand 1 template of the Modify Machine Attributes instruction are ignored when this selection value is specified. The materialize format of the MISR is as follows:

- MISR status — Char(2)
  - Termination status — Bit 0
    0 = Normal (TERMMPR)
    1 = Abnormal
  - IMPL — Bit 1
    Reference bit 14 to determine if the IMPL is initiated because of Terminate Machine Processing instruction with the restart option. If bit 14 is set, bit 1 should be ignored.
    0 = Normal (Manual depression of the Power-on or Load push button)
    1 = Auto-IMPL (Auto power-on of system followed by an IMPL)
  - Primary console status — Bit 2
    0 = Normal
    1 = Inoperative
  - Primary load/dump — Bit 3
    0 = Normal
    1 = Inoperative
  - Power status of Operator/Service panel sequence indicators — Bit 4
    0 = Normal
    1 = Inoperative
  - Duplicate user profile (AIPL only) — Bit 5
    0 = Not duplicate, new user profile created
    1 = Duplicate found and used by AIPL
  - Reserved (binary 0) — Bit 6

Selection Value **Attribute Description**

Hex 0108 (continued)

  - Damaged machine context — Bit 7
    0 = Not damaged
    1 = Machine context damaged
  - Power control initialization — Bit 8
    0 = Successful
    1 = Failed
  - Object recovery list status — Bit 9
    0 = Complete
    1 = Incomplete
  - Recovery phase completion — Bit 10
    0 = Complete
    1 = Incomplete
  - Most recent machine termination — Bit 11
    0 = Objects ensured
    1 = Object(s) not ensured at most recent machine termination
  - Last MISR reset — Bit 12
    0 = Object(s) ensured on every machine termination
    1 = Object(s) not ensured on every machine termination since last MISR reset
  - Console data storage test — Bit 13
    0 = Successful
    1 = Failed
  - Restart initiated from the Terminate Machine Processing instruction without powering down the system — Bit 14
    0 = IMPL was not initiated by the Terminate instruction
    1 = IMPL was initiated by the Terminate instruction
  - Reserved (binary 0) — Bit 15

Selection Value	Attribute Description	
Hex 0108 (continued)		
	• Number of damaged main storage units	Bin(2)
	• Number of entries in object recovery list	Bin(4)
	• Address of object recovery list	Space pointer
	• Process control space created as the result of IPL or AIPL	System pointer
	• Process static storage area space	System pointer
	• Process automatic storage area space	System pointer
	• Recovery object list (located by recovery object list pointer)	Char(*)
	– Recovery entry (repeated for number of entries)	Char(32)
	Object pointer	System pointer
	Object type	Char(1)
	Object status	Char(15)

Termination status indicates how the previous IMPL was terminated. If normal, the Terminate Machine Processing instruction successfully terminated the previous IMPL. If abnormal, the Terminate Machine Processing instruction did not successfully terminate the previous IMPL. This also implies that some cleanup of permanent objects may be required by the user.

IMPL indicates that the machine was automatically powered on and an IMPL was initiated because the previous IMPL was terminated as a result of a loss of the machine's primary power supply.

Primary console status indicates that the primary console is functioning normally or that it is inoperative.

Primary load/dump device status indicates that the load/dump device is functioning normally or that it is inoperative. This indicator is valid only if an IPL has been performed with the IMPL or IMPLA. If the primary load/dump device is inoperative and an AIPL is to be done, the machine terminates machine processing because the data needed to perform the AIPL is read from the load/dump device.

The power status for Operator/Service panel sequence indicators (light emitting diodes) indicates whether the sequence indicators on the Operator/Service panel are operational or not.

The duplicate user profile is valid only for AIPL and indicates if a user profile that is the same as the AIPL user profile to be created already exists in the machine context. The machine in this instance does not create the user profile for AIPL but rather uses the one located with the same name.

Damaged AIPL user profile indicates if the currently existing user profile was detected as damaged and a new user profile was created as specified in the AIPL user profile creation template.

Damaged machine context indicates if damage was detected in the machine context when an attempt was made to locate the duplicate user profile or to insert addressability to a newly created user profile. In either case, all current addressability is removed from the machine context, the new AIPL user profile is created, its addressability is inserted into the machine context, and the AIPL continues. Objects whose addressability was removed may have it reinserted using the Reclaim instruction for all objects or the Modify Addressability instruction for a specific object.

Power control initialization indicates if the power controller is operative or not.

The object recovery list status entry indicates that the status is complete unless one of the following conditions is true:

- The recovery list was lost.
- More objects were to be placed in the list but there was insufficient space.

The recovery phase completion entry indicates that the status is complete unless one of the following conditions occurs:

- An object to be recovered and/or inserted into the object recovery list no longer exists.
- The objects to be recovered could not be determined due to loss of internal machine indicators that specified which objects were in use at machine termination.

The most recent machine termination entry is set to 0 unless all objects were not ensured at the most recent machine termination.

The last MISR reset entry is set to 0 if all objects were ensured at every machine termination since the MISR was last reset (to 0) using the Modify Machine Attributes instruction.

The console data storage test indicates whether the console data storage is usable or not. If this test fails, the storage used by the IOC to operate the console is not operating properly and attempts to perform console operations may produce unpredictable results.

The number of damaged main storage transfer blocks entry indicates the number of main storage transfer blocks that were detected as damaged by the machine during IMPL.

The number of entries in the object recovery list entry indicates how many objects are listed in the space located by the address of object recovery list entry.

The address of object recovery list entry contains a space pointer to the list of the potentially damaged objects that were identified during machine initialization. The machine maintains this list of objects until a Modify Machine Attribute instruction for the MISR is executed. The number of such objects is indicated by the number of entries in the object recovery list entry.

The process control space created results from IPL or AIPL and is identified by a system pointer returned in this field.

Process static storage space system pointer addresses the space object that contains the PSSA created and initialized at IPL time. The space containing the PSSA is a temporary space and is not addressed by a context. This field contains binary 0's if the machine to programming transition is done via an IPL.

Process automatic storage area system pointer addresses the space object that contains the PASA created and initialized at IPL time. The space containing the PASA is a temporary space and is not addressed by any context. This field contains binary 0's if the machine to programming transition is done via an IPL.

The recovery object list identifies objects that required some activity performed on the object(s) during IPL. The list is located by the recovery object list pointer. Each entry in the list has the following general format:

- Object System pointer
- Object type Char(1)
- Object status Char(15)
  - General status Char(2)

    Damaged Bit 0

    0 = Object not damaged

    1 = Object damaged

    Reserved Bit 1

    Suspended Bit 2

    0 = Object not suspended

    1 = Object suspended

    Partially damaged Bit 3

    0 = Object not partially damaged

    1 = Object partially damaged

    Journal synchronization Bit 4

    0 = Synchronization complete or not necessary

    1 = Synchronization failure

    Reserved Bits 5-6

    IPL detected damage Bit 7

    0 = Any indicated damage was not detected by directory recovery

    1 = Indicated damage was detected by directory recovery

    Reserved Bits 8-15
  - Object specific status Char(13)
    (The format for the IPL recovery status for this portion of the object recovery list entries is different for each object type. A description of each follows by object type.)

    Commit block status Char(2)

Decommit Bit 0

0 = The journal has successfully been read backwards until either a start commit or a decommit entry was found. An attempt has been made to decommit all the data base changes but the attempt may not have been successful if the data space is damaged or if the function check flag is on.

1 = The journal has not successfully been read backwards to a start commit or decommit entry and all the changes have not been decommitted.

Journal read errors Bit 1

0 = No journal read errors

1 = Journal read errors occurred during decommit

Journal write errors Bit 2

0 = No journal write errors

1 = Journal write errors occurred during decommit

Partial damage to data space Bit 3

0 = No partial damage encountered

1 = Partial damage encountered on 1 or more data spaces

Damage to data space Bit 4

0 = No damage encountered

1 = Damage encountered on 1 or more data spaces

Function check Bit 5

0 = No function check encountered

1 = Function check encountered

Reserved Bit 6

Data space during IMPL	Bit 7
0 = Data space is synchronized with the journal	
1 = Data space is not synchronized with the journal. All changes may not be decommitted.	
Decommit reason code	Bits 8-10
000= Decommit not performed	
001= Decommit at IPL	
010= Process termination	
100= Decommit instruction (all other values reserved)	
Reserved	Bits 11-15
Reserved (binary 0)	Char(7)
Start commit journal Sequence number	Bin(4)

• Data space	
– Status	Char(13)
Indexes detached from data space	Bit 0
0 = Indexes remain attached	
1 = All indexes detached from this data space	
Reserved (binary 0)	Bits 1-15
– Reserved (binary 0)	Char(7)
– Ordinal entry number of last entry	Bin(4)

• Data space index	
– Status	Char(13)
Invalidated	Bit 0
0 = Not invalidated	
1 = Invalidated	
Reserved (binary 0)	Bits 1-15
– Reserved (binary 0)	Char(11)

• Journal port	
– Status	Char(13)
Synchronization status	Bit 0
0 = All objects synchronized	
1 = Not all objects synchronized	
Reserved	Bits 1-7
– Reserved	Char(10)
– Number of journal spaces attached	Bin(2)

• Journal space	
– Status	Char(13)
Journal space usable	Bit 0
0 = Journal space is usable	
1 = Journal space is not usable	
Threshold reached	Bit 1
0 = Threshold has not been reached	
1 = Threshold has been reached	
Reserved	Bits 2-7
– Reserved	Char(4)
– First journal sequence number	Bin(4)
– Last journal sequence number	Bin(4)
– Reserved (binary 0)	Char(13)

All objects–Any damage detected during IPL is reported in the general status information. If this damage is detected as a result of special processing performed during directory rebuild, it is indicated in the IPL detected damage bit. A journal synchronization failure indicates the designated object was not made current with respect to the journal. Subsequent attempts to apply journal changes from the journal to this object will not be allowed.

Commit block–All commit blocks that were attached to an active process during the previous IPL are interrogated at the following IPL. The system attempts to decommit any uncommitted changes referenced through these commit blocks. The results of this attempted decommit is reported in the status field. The system also returns the journal entry sequence number of the start commit journal entry (hex 0500) last created for this commit block if there were any uncommitted changes. If the number is not returned, a value of binary zero is returned.

Data space–If object damage was detected during IPL, the object is marked as damaged, damage is indicated in the object status field, and an event is signaled. In this case, the highest ordinal entry number is 0. In certain situations, the data space indexes over the data space become detached and therefore must be recreated. If the object is not damaged, the data space is usable and the highest ordinal entry number is set. The ordinal entry number of last entry indicates the last entry in the data space. Updates are not guaranteed. Updates may be out of sequence or partially applied and must be verified by the user for correctness.

Data space index–If object damage was detected during IPL, the object is marked as damaged, damage is indicated in the object status field, and an event is signaled. If the object was invalidated because changes were made in a data space addressed by the data space index, the data space index is included in the list and marked as invalidated. The associated data space is also included elsewhere in the object recovery list. Only damaged or invalidated data space indexes are included in the list.

Journal port–Each journal port in the system is interrogated at IPL. The status field contains the result of this checking and also the result of the attempt to synchronize the objects (if necessary) being journaled through the indicated journal port. The system also returns the number of journal spaces attached to the journal port after IPL is complete.

Journal space–Each journal space that was attached to a journal port or used by the system to synchronize an object which was being journaled at the time of the previous machine termination is interrogated during IPL. The status field reports the results of this interrogation and synchronization use. Journal spaces are only referenced by the object recovery list if this IPL was preceded by an abnormal failure or some unexpected condition was discovered during the IPL. The first journal sequence number on the journal space is returned. The last usable entry on the journal space is also identified. If the journal space is damaged, these fields will contain zeroes.

Selection Value **Attribute Description**

Hex 0110 Machine system name (can be materialized and modified)

The machine system name is used as an identifier for certain source/sink APPC applications. It is defaulted at IMPL time to the machine serial number (with the first character changed, if necessary, to an alphabetic character). Thereafter, it may be materialized or modified to any value of 1 through 8 characters with the first character alphabetic.

The format of the template for materializing or modifying the system name is as follows (including the usual 8-byte prefix):

• Number of bytes available	Bin(4)
• Number of bytes provided (ignored) on modify)	Bin(4)
• System name (with count)	Char(10)
– Count of system name	Bin(2)
– System name	Char(8)

For the materialize system name, the number of bytes available should be set to 18 to include the system name in case it is 8 bytes long. In any case, it must be at least 8 bytes long; otherwise, an exception is signaled.

For the modify system name, the count of the system name must be 1 through 8 with the first character alphabetic and the number of bytes available must be at least 11. The number of bytes available must be at least 10 more than the count of the system name. If an error is detected, an exception is signaled.

The modify system name may be done at any time; however, unpredictable results may occur if done while certain source/sink APPC applications are active.

Selection Value **Attribute Description**

Hex 0114 Verify new system name and materialize new system name

The new system name is a tentative new value chosen for the machine system name.

The verify new system name (the modify form) determines whether the new value of the system name given in the template is legitimate and does not conflict with the LU names of any existing APPC LUD(s). If there is no conflict, it is saved as the new system name; the actual machine system name is not altered at this time. If there is a conflict, an exception is signaled.

The materialize new system name materializes the saved value of the new system name (if any), similar to the materialization of the actual machine system name.

The format of the template for verify new system name or materialize new system name (including the usual 8-byte prefix) is as follows:

• Number of bytes available	Bin(4)
• Number of bytes provided (ignored on verify new system name)	Bin(4)
• New system name (with count)	Char(10)
– Count of new system name	Bin(2)
– New system name	Char(8)

Syntax rules for materialize new system name are the same as for materialize machine system name. If the new system name does not exist, the count of new system name field will be zero.

Selection Value **Attribute Description**

Hex 0118 Uninterruptible power supply connection type (can be materialized and modified)

The uninterruptible power supply connection type provides the machine with information about how the uninterruptible power supply is connected to the system and the disks. If the uninterruptible power supply is connected to the system and the disks, the normal power warning feature (full uninterruptible power supply) will be activated when utility power fails. If the uninterruptible power supply is connected to only the System/38 processing unit and its feature disk frame, the basic uninterruptible power supply will be activated when utility power fails. In this case, the delay timer value determines a delay time interval after utility power fails. The delay time interval is the amount of time the system waits for the return of utility power. The delay time interval is equal to the delay timer value less the time required to copy main storage. The delay timer value will not be used by the machine as a delay if the uninterruptible power supply connection type machine attribute does not indicate a basic uninterruptible power supply.

The format of the template for the uninterruptible power supply connection type (including the 8-byte prefix) is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Connection type Char(2)

  Hex 0000–Indicates that the System/38 and the disks are connected to the full uninterruptible power supply. When utility power fails, control is passed to CPF or the system is powered down by a machine shutdown.

  Hex 0010–Indicates that only the System/38 processing unit and its feature frame are connected to the basic uninterruptible power supply. When utility power fails, the delay timer value is used to determine the delay time interval the system will wait before the system is powered down. If power does not stabilize within the delay time interval, main storage is saved and the machine is powered down. This power down will allow an auto-IMPL when power is restored if the auto-IMPL switch is set correctly.

- Delay timer value (seconds) Bin(2)

Events

000C Machine resource
0201 Machine auxiliary storage threshold exceeded

0010 Process
0701 Maximum processor time exceeded
0801 Process storage limit exceeded

0016 Machine observation
0101 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

Exceptions

	Operands		
Exception	1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	
38 Template Specification			
03 Materialization length exception	X		

MODIFY MACHINE ATTRIBUTES (MODMATR)

Op Code (Hex)	Operand 1	Operand 2
0646	Source value	Attribute selection

Operand 1: Space pointer.

Operand 2: Character(2) scalar (fixed-length).

Description: The instruction alters the value of a specific machine attribute. The value of the specified machine attribute is altered to the value specified by operand 1. Operand 2 options specify the type of information to be modified.

The machine attributes that may be modified are divided into nine groups. Byte 0 of the attribute selection operand specifies the group from which the machine attributes are to be modified. Byte 1 of the operand selects a specific subset of that group of machine attributes.

The groups are indicated as follows:

Group	Group Value (Hex)	Function
1	00	General attributes
2	80	Machine defined
3	40	Machine defined
4	20	Machine defined
5	10	Machine defined
6	08	Machine defined
7	04	Machine defined
8	02	Machine defined
9	01	Machine defined

Modification of attributes in groups two through nine requires that the user profile controlling execution of the instruction must have modify machine attributes authority for the specific group to be modified.

The format of the source value modification template defined by operand 1 is as follows:

• Template size specification	Char(8)
– Number of bytes provided	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Attribute specifications as defined by the attribute selection operand	Char(8)

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

The machine attributes defined by operand 2 are modified according to the following selection values:

Selection Value **Attribute Description**

Hex 0000 MCR (machine configuration record)

The MCR contains the internal configuration of the machine. The MCR machine attribute is provided for machine maintenance only and has no meaning or value to the user. The MCR is materialized as a contiguous character string of binary data.

Hex 0100 Time-of-day clock (can be materialized and modified)

The time-of-day clock provides a consistent measure of elapsed time. The maximum elapsed time the clock can indicate is approximately 143 years.

The time-of-day clock is a 64-bit unsigned binary counter with the following format:

```
0................41 42 reserved 63
```

The bit positions of the clock are numbered from 0 to 63.

The clock is incremented by adding a 1 in bit position 41 every 1024 microseconds. Bit positions 42 through 63 are used by the machine and have no special meaning to the user. Note that these bits (42-63) may contain either binary 1's or binary 0's.

Unpredictable results occur if the time of day is materialized before it is set.

The maximum unsigned binary value that the time of day clock can be modified to contain is hex DFFFFFFFFFFFFFFF.

Selection Value **Attribute Description**

Hex 0104 Initial process definition template (can be materialized and modified)

The initial process definition template is used by the machine to perform an initial process load. The initial process definition template has the same format as the process definition template defined by the Initiate Process instruction. See *Chapter 11. Process Management Instructions*.

No check is made and no exception is signaled if the values in the template are invalid; however, the next initial process load will not be successful.

Hex 0108 Machine initialization status record (can be materialized and modified)

The MISR (machine initialization status record) is used to report the status of the machine. The status is collected at IMPL (initial microprogram load) or IMPLA (initial microprogram load abbreviated).

Modifying the MISR causes it to be reset. The values in the operand 1 template of the Modify Machine Attributes instruction are ignored when this selection value is specified.

Selection Value **Attribute Description**

Hex 0114 Verify new system name and materialize new system name

The new system name is a tentative new value chosen for the machine system name.

The verify new system name (the modify form) determines whether the new value of the system name given in the template is legitimate and does not conflict with the LU names of any existing APPC LUD(s). If there is no conflict, it is saved as the new system name; the actual machine system name is not altered at this time. If there is a conflict, an exception is signaled.

The materialize new system name materializes the saved value of the new system name (if any), similar to the materialization of the actual machine system name.

The format of the template for verify new system name or materialize new system name (including the usual 8-byte prefix) is as follows:

• Number of bytes available	Bin(4)
• Number of bytes provided (ignored on verify new system name)	Bin(4)
• New system name (with count)	Char(10)
– Count of new system name	Bin(2)
– New system name	Char(8)

Selection Value **Attribute Description**

Hex 0114 (continued)

Syntax rules for the verify new system name are similar to those for modify machine system name. In addition to the syntax checking, a check is made of all existing APPC LUDs to ensure that the new system name does not match the remote LU name of any APPC LUD that has a null local LU name and that the new system name would not cause any two APPC LUDs on the same CD to have matching LU names.

If such a problem is detected, a source/sink configuration invalid exception (hex 3401) is signaled with a subcode of hex 5307.

A verify new system name is performed to check a tentative new system name. This may be done once or several times. The next time the system IPL is performed, the most recent new system name is used as input to modify machine system name.

Selection Value **Attribute Description**

Hex 0118 Uninterruptible power supply connection type (can be materialized and modified)

The uninterruptible power supply connection type provides the machine with information about how the uninterruptible power supply is connected to the system and the disks. If the uninterruptible power supply is connected to the system and the disks, the normal power warning feature (full uninterruptible power supply) will be activated when utility power fails. If the uninterruptible power supply is connected to only the System/38 processing unit and its feature disk frame, the basic uninterruptible power supply will be activated when utility power fails. In this case, the delay timer value determines a delay time interval after utility power fails. The delay time interval is the amount of time the system waits for the return of utility power. The delay time interval is equal to the delay timer value less the time required to copy main storage. The delay timer value will not be used by the machine as a delay if the uninterruptible power supply connection type machine attribute does not indicate a basic uninterruptible power supply.

The format of the template for the uninterruptible power supply connection type (including the 8-byte prefix) is as follows:

- Number of bytes available Bin(4)

- Number of bytes provided Bin(4)

Selection Value	Attribute Description
Hex 0118 (continued)	

- Connection type Char(2)

  Hex 0000–Indicates that the System/38 and the disks are connected to the full uninterruptible power supply. When utility power fails, control is passed to CPF or the system is powered down by a machine shutdown.

  Hex 0010–Indicates that only the System/38 processing unit and its feature frame are connected to the basic uninterruptible power supply. When utility power fails, the delay timer value is used to determine the delay time interval the system will wait before the system is powered down. If power does not stabilize within the delay time interval, main storage is saved and the machine is powered down. This power down will allow an auto-IMPL when power is restored if the auto-IMPL switch is set correctly.

- Delay timer value (seconds) Bin(2)

Hex 011C Uninterruptible power supply power off type (modifiable only)

The uninterruptible power supply power off type is used to inform the machine that a higher level user plans to terminate machine processing. The machine uses this information to determine how to turn the machine power off when utility power fails before the Terminate Machine Processing instruction is issued with uninterruptible power supply connection type machine attribute set to hex 0010 (basic uninterruptible power supply). This value is always reset at IMPL to allow auto-IMPL for subsequent power off.

Selection Value	Attribute Description
Hex 0118 (continued)	

The format of the template for uninterruptible power supply power off type (including the 8-byte prefix) is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Power off type Char(2)

  Hex 4000–Indicates that power off should inhibit auto-IMPL.

  Hex 8000–Indicates that power off should allow auto-IMPL.

Authorization Required

- Special authorization

Events

0002 Authorization
 0301 Special authorization violation

000C Machine resources
 0201 Machine auxiliary storage threshold exceeded

0010 Process
 0701 Maximum processor time exceeded
 0801 Process storage limit exceeded

0016 Machine observation
 0101 Instruction reference

0017 Damage set
 0401 System object damage set
 0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
0A Authorization			
08 Special authorization required			X
0C Computation			
10 Weak key not valid	X		
11 Key parity invalid	X		
10 Damage Encountered			
04 System object damage state	X	X	X
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	
38 Template Specification			
02 Template size invalid	X		

RECLAIM LOST OBJECTS (RECLAIM)

Op Code (Hex)	Operand 1	Operand 2
0686	Reclaimed objects list	Reclaim options

Operand 1: Space pointer.

Operand 2: Character(2) scalar.

Description: The instruction finds permanent objects, which have been lost from their owning user profiles, and optionally rebuilds the machine context. The machine searches storage for permanent objects and checks that the owning user profile specified by the object actually exists and considers itself to own the object. If not, the object is lost and an entry is returned in the reclaimed objects list.

Any storage areas not identifiable as valid system objects are destroyed. Following abnormal system or instruction termination, there may be portions of objects which continue to occupy storage space. The Reclaim Lost Objects instruction can be used to free up this storage space.

The machine context may optionally be updated to ensure that it locates all objects that are to be addressed by the machine context (contexts, user profiles and source/sink objects). This option should be used when the machine context loses entries or is damaged. The machine initialization status record machine attribute indicates whether AIMPL (alternate initial microprogram load) detected machine context damage.

When addressability to an object is to be inserted into the machine context, it is possible that an object of the same name, type and subtype is now addressed by the machine context. This can occur if the newer object was created after the currently existing object was lost. If this occurs, a pointer is returned to the object and its addressability is not inserted into the machine context. The Rename Lost Objects instruction can be used to change the name of the object and to insert addressability into the machine context.

Operand 2 specifies the verification of the machine context. The format is as follows:

• Reclaim options	Char(2)
– Machine context rebuild	Bit 0
0 = Do not rebuild	
1 = Rebuild	
– Reserved (binary 0)	Bits 1-15

Operand 1 identifies the reclaimed objects. This includes objects with improper ownership entries as well as those that are to be inserted into the machine context but cannot be because the object identification is in conflict.

• Materialization size specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization	Bin(4)
• Number of objects lost from user profile	Bin(4)
• Number of machine context duplicates	Bin(4)
• Number of bytes reclaimed	Bin(4)
• Reserved	Char(12)
• Reclaimed object entry (repeated for each object)	Char(32)
– Object pointer	System pointer
– Entry type	Char(1)
User profile	Bit 0
0 = Not lost from user profile	
1 = Lost from user profile	
Machine context	Bit 1
0 = Not a machine context duplicate	
1 = Machine context duplicate	
Reserved (binary 0)	Bits 2-7
– Reserved (binary 0)	Char(15)

No authorization is returned in the system pointers.

Information in each referenced object can be used to restore the ownership of the object to a user profile (transfer ownership).

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This number is supplied as input to the instruction and is not modified by the instruction. A number less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is larger than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Special (all objects authority)

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation		X
10 Damage Encountered		
04 System object damage state		X
44 Partial system object damage		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	
03 Scalar value invalid	X	
38 Template Specification		
03 Materialization length exception	X	

TERMINATE MACHINE PROCESSING (TERMMPR)

Op Code (Hex)	Operand 1	Operand 2
0622	Terminate options	Termination reason information for maintenance use

Operand 1: Character(2) scalar.

Operand 2: Space pointer or null.

Description: This instruction terminates machine processing by destroying or suspending all processes in the machine including the process that issued the instruction. The values of the termination options (operand 1) determine the functions to be performed. The following is the format of operand 1.

- Termination options — Char(2)
  - Machine termination options — Bits 0-3
    - 0001 = Terminate machine processing and enter the check-stop state.
    - 0010 = Terminate immediately and leave existing processes in an internal machine state that will retain information for diagnostic purposes.
    - 0100 = Destroy all processes, turn off the machine power supply, and do not allow an auto-IMPL.
    - 1000 = Destroy all processes and re-IMPL. If the machine is running on utility power, an IMPL of the machine will be performed automatically. If the machine is running on auxiliary power, the machine is powered down and an auto-IMPL occurs when utility power is restored (if the auto-IMPL feature is activated).

    All other values are reserved; if any other values are specified, they cause an exception.

– Reserved (binary 0)	Bits 4-5
– Regenerate permanent addresses	Bit 6
0 = No	
1 = Yes	
– Address regeneration	Bit 7
0 = No	
1 = Yes	
– Termination code	Bits 8-15

If the machine termination option is 0001 or 0010, then the termination code in bits 8-15 is displayed on the sequence indicators of the operator/CE panel. The allowed value is in the range from hex 80 through hex FF. Any other value causes a default value of hex 00 to be displayed. A hex 00 value indicates that an invalid termination code was specified.

If any other machine termination option is specified, this entry is ignored by the instruction.

The address regeneration and regenerate permanent addresses termination options control whether or not internal machine addresses used for unique identification of system objects are to be regenerated during the termination of machine processing. These internal machine addresses are a machine resource which can be used up over time through successive creation and destruction of system objects. When yes is specified for these options, a regeneration of these addresses is performed during the termination of machine processing to renew the supply of addresses available for usage by the machine.

The resource of internal machine addresses is separated into two groups, one used for temporary system objects and one used for permanent system objects. Address regeneration always renews the supply of internal machine addresses available for usage with temporary system objects. Address regeneration renews the supply of internal machine addresses available for usage with permanent system objects only if yes is specified for the regenerate permanent addresses option. The process of regenerating the supply of internal machine addresses can be lengthy, on the order of hours, depending upon the number of objects which exist on the machine and the amount of auxiliary storage on the machine. The system console condition indicators will display the pattern for hex 3F30 while address regenerating processing is being performed. Regeneration of permanent addresses will cause the machine to perform authority recovery processing on the next IMPL as well.

Yes can only be specified for the address regeneration option in conjunction with machine termination options 0100 and 1000. Yes can only be specified for the regenerate permanent addresses option in conjunction with a specification of yes for the address regeneration option. Violation of these restrictions results in the signaling of the scalar value invalid exception. When no is specified for these options, the corresponding address regeneration is not performed.

Operand 2 identifies a space pointer that addresses an area in a space. The space pointer locates information that further defines the reason for machine termination. If the space is not a permanent object, the information will be destroyed by the machine because all temporary objects allocated are destroyed when machine processing is terminated.

Machine termination causes the following:

- The process is terminated and no additional instructions are allowed to execute. The process does not enter the termination phase.

- All permanent system objects are written to auxiliary storage.

- If the power supply is to be turned off, an attempt is made to turn off the power supply for all devices associated with source/sink objects that have the power control attribute. If one or more of the devices associated with source/sink objects cannot be powered off, the machine is placed in the check stopped state. If all source/sink objects are powered off, the power supply for the machine is turned off.

When more than one process exists in the machine, execution of the instruction causes termination of each of the processes at the next instruction boundary. The normal process termination functions as defined by the Terminate Process instruction are not performed.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Privileged instruction

Events

0002 Authorization
    0201 Privileged instruction violation

0017 Damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
02 Privileged instruction			X
10 Damage Encountered			
04 System object damage state	X	X	
44 Partial system object damage	X	X	X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X		
02 Scalar attributes invalid	X		
03 Scalar value invalid	X		

Chapter 20. Journal Management Instructions

This chapter describes all instructions used for journal management. These instructions are arranged in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

APPLY JOURNALED CHANGES (APYJCHG)

Op Code (Hex)	**Operand 1**	**Operand 2**
05AA	Object list	Apply template

Operand 1: Space pointer.

Operand 2: Space pointer.

Description: This instruction applies the changes specified in the apply template (operand 2) to the object list specified in operand 1, if requested.

The format of the object list is as follows:

• Number of objects	Bin(2)
• Total number of entries applied	Bin(4)*
• Last entry interrogated	Bin(4)*
• Reserved (binary 0)	Char(22)
• List of objects (1 to n)	
– Object	System pointer
– Starting sequence number	Bin(4)
– Ending sequence number	Bin(4)
– Number of entries applied	Bin(4)*
– Reserved (binary 0)	Char(4)
– Object dependent status	Char(16)*

The object list must be aligned on a multiple of 16 bytes.

Note: The values of the entries annotated with an asterisk (*) are ignored by this instruction.

The number of objects field must contain a value greater than 0 and indicate how many objects are contained in the following list.

The total number of entries applied field contains the total number of journal entries whose changes have been applied as a result of this execution of the instruction. This value is returned whenever the instruction is executed unless an exception occurs while the system is processing the template, in which case it is not modified. This field, along with the starting sequence number or ending sequence number, determines the range of entries processed by this invocation of the instruction.

The last entry interrogated field contains the sequence number of the last entry that was checked for eligibility for application to one of the indicated objects.

The object field must contain a system pointer to a valid (not damaged, destroyed, or suspended) object. The object identified must have a journal ID specified (was once journaled); however, it need not currently be journaled. No duplicate journal IDs are allowed in the list of objects; that is, two or more objects in the list may not have the same journal ID. If so, a journal ID no unique exception is signaled and the operation is terminated.

The starting sequence number contains the journal sequence number of the first journal entry which is to be applied to the object. This sequence number must be contained on the journal spaces provided in operand 2, or a template value invalid exception is signaled.

The ending sequence number contains the journal sequence number of the last journal entry which is to be applied to the object. This sequence number must be contained on the journal spaces provided in operand 2, or a template value invalid exception is signaled. If this sequence number is not equal to or greater than the starting sequence number, a template value invalid exception is signaled.

The number of entries applied field contains the total number of journal entries whose changes have been applied to this object as a result of this execution of the instruction. This value is returned whenever the instruction is executed unless an exception occurs while processing the template.

Only changes made to data spaces may be applied to the object list through this instruction. The format for the object dependent information returned in the object list of the Apply Journaled Changes instruction for a data space is as follows:

• Entry count	Bin(4)
• Deleted entry count	Bin(4)
• Entry limit	Bin(4)
• Reserved (binary 0)	Char(4)

The format of the apply template is as follows:

• Apply options	Char(4)
– Apply changes indicator 0 = Apply changes 1 = Do not apply changes	Bit 0
– Retrieval direction 0 = Ascending 1 = Descending	Bit 1
– Selection list indicator 0 = No selection list specified 1 = Selection list specified	Bit 2
– Selection list processing indicator 0 = Do not apply those changes referenced in the selection list 1 = Apply only those changes referenced in the selection list	Bit 3
– Transaction list indicator 0 = Do not create a transaction list 1 = Create a transaction list	Bit 4
– List uncommitted transactions indicator 0 = Do not list uncommitted transactions 1 = List uncommitted transactions	Bit 5
– List committed transactions indicator 0 = Do not list committed transactions 1 = List committed transactions	Bit 6
– List decommitted transactions indicator 0 = Do not list decommitted transactions 1 = List decommitted transactions	Bit 7
– Reserved (binary 0)	Bits 8-31
• Number of entries in the selection list	Bin(4)
• Number of entries in the start transaction list	Bin(4)
• Maximum entries in the start transaction list	Bin(4)
• Number of journal spaces	Bin(4)
• Reserved (binary 0)	Char(12)
• Selection list	Space pointer
• Transaction list	Space pointer
• Journal space (1 to n)	System pointer

The apply template must be aligned on a multiple of 16 bytes.

A value of binary 0 in the apply changes indicator field specifies that all journal entries that are within the range specified for any object in the operand 1 object list are to be applied to that object. A value of binary 1 in this field results in a transaction list being generated (if requested). Any errors detected during this scan (apply changes indicator field has a value of binary 1) of the journal entries will result in the termination of the instruction with the appropriate exception.

If the retrieval direction indicator field has a value of binary 0, the journal entries will be retrieved from the journal spaces in ascending order (starting sequence number to ending sequence number) and a determination of whether or not to apply the indicated change is made. If the retrieval direction indicator field has a value of binary 1, the journal entries will be retrieved from the journal spaces in descending order (ending sequence number to starting sequence number) and a determination of whether or not to apply the indicated change is made.

A value of binary 0 in the selection list indicator field specifies that no selection list is to be processed. If this field has a value of binary 0, the selection list processing indicator, the number of entries in the selection list, and the selection list fields are ignored and all journal entries are applied that pertain to the objects listed in operand 1. A value of binary 1 in the selection list indicator field specifies that the selection list is to be processed according to the options specified in the selection list processing indicator field.

A value of binary 0 in the selection list processing indicator field specifies that the journal entries that are indicated in the selection list are not to be applied to the indicated objects in the operand 1 object list but all other transactions are to be applied. A value of binary 1 in the selection list processing indicator field specifies that only journal entries that are indicated in the selection list are to be applied to the indicated objects in the operand 1 object list and no other transactions are to be applied.

A value of binary 1 in the transaction list indicator field specifies that entries are to be generated in the transaction list according to the options specified in the list uncommitted transactions indicator, the list committed transactions indicator, and the list decommitted transactions indicator fields. A value of binary 0 in the transaction list indicator field specifies that entries are not to be generated in the transaction list, the list uncommitted transactions indicator, the list committed transactions indicator, the list decommitted transactions indicator, the number of entries in the transaction list, the maximum number of entries in the transaction list, and the transaction list fields are ignored.

A value of binary 1 in the list uncommitted transactions indicator specifies that entries that did not start and/or complete processing in the range indicated in operand 1 are placed in the transaction list for identification. A transaction is a group of changes identified with the same commit ID in a set of journal entries. A value of binary 0 in the list uncommitted transactions indicator specifies that entries that did not start and/or complete in the range specified in operand 1 are not placed in the transaction list.

A value of binary 1 in the list committed transactions indicator specifies that entries that are committed within the range indicated in operand 1 are placed in the transaction list for identification. These entries are indicated by a start commit journal entry and a corresponding commit journal entry. A value of binary 0 in the list committed transactions indicator specifies that entries that are committed within the range indicated by operand 1 are not generated in the transaction list.

A value of binary 1 in the list decommitted transactions indicator specifies that entries that are decommitted within the range indicated in operand 1 are placed in the transaction list for identification. These entries are indicated by a start commit journal entry and a corresponding commit journal entry. A value of binary 0 in the list committed transactions indicator specifies that entries that are decommitted within the range indicated by operand 1 are not generated in the transaction list.

The number of entries in the selection list indicates the number of entries that are contained in the selection list provided. If this field contains a value of 0, the selection list is ignored and is assumed to contain no entries. If this field contains a value less than 0, a template value invalid exception is signaled.

The number of entries in the transaction list indicates the number of entries that are contained in the transaction list provided. If this field contains a value of 0, the transaction list is assumed to have no entries in it and entries generated by the instruction will be returned in the first entry position provided. If this field contains a value greater than zero, the entries currently in the transaction list may be retained or dropped and additional entries, if generated, will be added to the list. The transaction list returned at instruction completion will contain entries provided as input to the instruction, if they still meet the criteria for creating entries in the transaction list, but not necessarily in the order provided to the instruction. If this field contains a value less than 0, a template value invalid exception is signaled.

The maximum number of entries in the transaction list value indicates the maximum number of entries the system generates in the transaction list provided. This value includes the number of entries provided to the system plus the number generated by the system. When this number is reached during execution of the instruction, a transaction list limit reached exception is signaled and the instruction is terminated. If this field contains a value less than 0 or is less than the value contained in the number of entries in the transaction list, a template value invalid exception is signaled.

The selection list and transaction list fields, if provided, both identify a space that contains a list of multiple entries with the following format:

• Commit ID	Bin(4)
• Commit or decommit journal sequence number	Bin(4)
• Status flags	Char(2)
– Entry valid indicator	Bit 0
0 = Ignore this entry	
1 = Valid entry	
– Start commit	Bit 1
0 = No start commit entry was found	
1 = Start commit journal entry was found	
– Modification indicator	Bit 2
0 = No object modification was found for this transaction	
1 = Object modification(s) were found for transaction	
– Commit identified	Bit 3
0 = No commit entry was found for this transaction	
1 = A commit entry was found for this transaction	
– Decommit identified	Bit 4
0 = No decommit entry was found for this transaction	
1 = A decommit entry was found for this transaction	
– Reserved (binary 0)	Bits 5-15

The commit ID field identifies a transaction and is the value of the commit ID field in the prefix portion of the journal entries. This value corresponds to the journal sequence number of the start commit journal entry. All subsequent changes made under commitment control, through the same commit block, to the objects and the corresponding commit or decommit entry are identified with this commit ID. A value of 0 in this field indicates changes made to the indicated objects were not done under commitment control (the commit ID in the prefix of the journal entry is binary 0). The system does not verify that this field contains a valid commit ID when input to the transaction list or selection list as contained in the prefix of one or more journal entries.

The commit or decommit sequence number field contains the journal entry sequence number of the journal entry for the commit or decommit of the transaction, if it is in the range of entries processed by the instruction. If it is not in this range, the field contains a value of 0.

An entry valid indicator value of binary 1 specifies that this entry is a valid entry in the list where it resides. A value of binary 0 in this field is used to identify entries that are ignored (not considered part of the valid entries) in the list.

A value of binary 1 in the start commit field indicates that a start commit journal entry was found for the indicated transaction within the range of entries processed by the instruction. A value of binary 0 in this field indicates that a start commit journal entry was not found for the indicated transaction within the range of entries processed by the instruction.

A value of binary 1 in the modification indicator field indicates that at least one change was processed for the indicated transaction (for one or more of the objects listed in operand 1) within the range of entries processed by the instruction. A value of binary 0 in this field indicates that no change journal entries were found for the indicated transaction within the range of entries processed by the instruction.

A value of binary 1 in the commit identified field indicates that a commit journal entry was found for the indicated transaction within the range of entries processed by the instruction. A value of binary 0 in this field indicates that a commit journal entry was not found for the indicated transaction within the range of entries processed by the instruction.

A value of binary 1 in the decommit identified field indicates that a decommit journal entry was found for the indicated transaction within the range of entries processed by the instruction. A value of binary 0 in this field indicates that a decommit journal entry was not found for the indicated transaction within the range of entries processed by the instruction.

The number of journal spaces field must contain a value greater than 0 which indicates how many journal spaces are to be searched for journal entries that satisfy the starting and ending sequence numbers. The journal space list is a list of system pointers which contains addressability to valid journal spaces (not damaged or suspended and have previously been attached to a journal port).

Note: Partial damage on a journal space does not invalidate it. The list contains as many entries as is specified in the number of journal spaces field. Each journal space must logically precede the next journal space in the list (the journal sequence numbers are continuous and ascending across the journal spaces). The journal sequence number must not be reset to 1 on any but the first journal space identified. If any exceptions, except partial damage to a particular journal space, are identified, an invalid journal space exception is signaled and no journal entries are applied.

The indicated starting journal entry is interrogated, and if the entry pertains to the designated object, the corresponding change will be made to the object. If any changes exist for the designated object before the indicated starting entry but they were not committed until after the starting point, these changes are applied and an entry to indicate this is generated in the transaction list, if requested. Each subsequent entry on the journal spaces is checked, and if it pertains to the designated object, the corresponding change is made. This procedure is repeated until the indicated ending entry has been processed or an error is encountered. If any uncommitted changes are encountered on the journal spaces that were not committed until after the indicated ending point, they are applied and the incomplete transaction is identified in the transaction list, if requested.

If the journal entries are to be processed in descending order, the system removes all of the changes from the objects in the operand 1 object list, indicated in the journal entries in the range processed. If a selection list is not provided, all changes for the objects identified in the operand 1 object list are processed. If a selection list is provided, only those transactions identified in the list are processed. A commit ID with a value of 0 in a valid selection list causes all changes that were not made under commitment control to be removed in the range processed. Processing the selection list and application of changes is subject to the selection list processing indicator. The system requires that before images reside in the journal to allow this type of processing. If a before image is required and is not present on the journal, an apply journal failure exception is signaled and processing is terminated.

Changes are made to all objects identified in operand 1 in the order the journal entries are retrieved and processed. If an exception is encountered while processing the journal entries, the following conditions occur:

- The appropriate exception is signaled.

- All return fields contain the correct status and return values to that point in the processing.

- The last entry that was attempted to be processed will be returned in the last entry interrogated field.

- The transaction list, if requested, contains a complete list of the requested entries.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
  - Journal spaces indicated in the apply template

- Object management
  - Objects in operand 1 object list

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Implicit locks
  - Objects in the operand 1 object list are locked implicitly: LEAR in the ascending direction and LENR in the descending direction.
  - Also the addition of certain journal entries cause an implicit LENR lock while the entry is being applied.

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port became unusable

001C Journal space
    0301 Threshold reached

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage			
04 System object	X	X	X
44 Partial system object damage			X
12 Data Base			
09 Duplicate key value	X		
21 Unable to perform index maintenance	X		
23 DSI selection routine failure	X		
27 Data space index key with invalid floating-point field	X		
34 Non-user exit selection routine failure	X		
38 Derived field operation error	X		
1A Lock State			
01 Invalid lock state	X	X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded	X		
06 Machine lock limit exceeded	X		
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
04 Object not eligible for operation	X		
05 Object not available to process	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X

Exception	Operands 1	 2	Other
2C Program Execution			
06 Instruction cancellation			X
07 Instruction termination			X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
02 Process storage limit exceeded			X
30 Journal Management			
01 Apply journal failure			X
02 Entry not journaled			X
04 Invalid journal space		X	
05 Invalid selection/transaction list entry		X	
07 Journal ID not unique	X		
09 Transaction list limit reached			X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid	X	X	
03 Scalar value invalid	X	X	
38 Template Specification			
01 Template value invalid	X	X	

CREATE JOURNAL PORT (CRTJP)

Op Code (Hex)	Operand 1	Operand 2
05A2	Journal port	Journal port template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction creates a journal port according to the description in operand 2 and returns addressability in the system pointer identified by operand 1. A journal port must be created as a permanent object.

The format of the journal port template is as follows:

• Template size	Char(1)
– Number of bytes provided by user	Bin(4)*
– Number of bytes that can be materialized	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Reserved (binary 0)	Bit 0
– Space attributes	Bit 1
0 = Fixed-length	
1 = Variable-length	
– Initial context	Bit 2
0 = No addressability insert	
1 = Insert addressability	
– Reserved (binary 0)	Bits 3-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)

- Performance class Char(4)
  - Space alignment Bit 0
    - 0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    - 1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.
  - Reserved (binary 0) Bits 1-4
  - Main storage pool selection Bit 5
    - 0 = Process default main storage pool is used for object.
    - 1 = Machine default main storage pool is used for object.
  - Transient storage pool selection Bit 6
    - 0 = Default main storage pool (process default or machine default as specified for main storage pool selection) is used for object.
    - 1 = Transient storage pool is used for object.
  - Block transfer on implicit access state modification Bit 7
    - 0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
    - 1 = Transfer the machine default storage transfer size. This value is 8 storage units.
  - Unit number Bits 8-15
  - Reserved (binary 0) Bits 16-31

- Reserved (binary 0) Char(7)

- Context System pointer

- Reserved (binary 0)Char(16)

- Length of timestamp in prefix (0 to 8) Bin(2)

- Length of process name in prefix (0 to 30) Bin(2)

- Length of user profile name in prefix (0 to 30) Bin(2)

- Length of program name in prefix (0 to 30) Bin(2)

The journal port template must be aligned on a multiple of 16 bytes.

Note: The values of the entries annotated with an asterisk (*) are ignored by this instruction.

The journal port is owned by the user profile governing process execution. The owning user profile is implicitly assigned all authority states to the journal port. The storage occupied by the journal port is charged to this owning user profile.

The object identification specifies the symbolic name that identifies the journal port within the machine. A type code of hex 09 is implicitly supplied by the machine. The object identification is used to identify the journal port on materialize instructions as well as to locate the object in a context that addresses the object.

A space may be associated with the created object. This space may be fixed or variable. Initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated is dependent on an algorithm defined by a specific implementation. A fixed-size space request of zero length results in no space being allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this byte value is also used to initialize the new allocation.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created object is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

The performance class parameter provides information allowing the machine to more effectively manage the object, considering the overall performance objective of operations involving the journal port.

The length of timestamp in prefix field must contain a value equal to or greater than 0 and less than or equal to 8. It indicates the length of the timestamp field that is to be contained in the journal prefix of every entry created through this journal port. The timestamp that is placed in the journal prefix indicates the time that the journal entry was generated and placed on a journal space. If this field contains a value of 0, the timestamp will not appear in journal prefix of entries created through this journal port. If a value less then 8 is provided in this field, the leading portion of the timestamp is placed in the prefix.

The length of process name in prefix field must contain a value equal to or greater than 0 and less than or equal to 30. It indicates the length of the process name field which is to be contained in the journal prefix of every entry created through this journal port. The process name that is placed in the journal prefix is the name of process (as indicated by the Create Process Control Space or Rename Object instruction) performing the changes that result in journal entries being created. If this field contains a value of 0, the process name will not appear in journal prefix of entries created through this journal port. If a value less than 30 is provided in this field, the leading portion of the process name is placed in the prefix.

The length of user profile name in prefix field must contain a value equal to or greater than 0 and less than or equal to 30. It indicates the length of the user profile name field that is to be contained in the journal prefix of every entry created through this journal port. The user profile name that is placed in the journal prefix is the name of user profile (as indicated by the Create User Profile or Rename Object instruction) the process is running under when performing the changes that result in journal entries being created. If this field contains a value of 0, the user profile name will not appear in journal prefix of entries created through this journal port.

The length of program name in prefix field must contain a value equal to or greater than 0 and less than or equal to 30. It indicates the length of the program name field which is to be contained in the journal prefix of every entry created through this journal port. The program name that is placed in the journal prefix is the name of program (as indicated by the Create Program or Rename Object instruction) that is the first program in the invocation stack with a value of binary 0 specified on the creation template for the program in the system program attribute field. The search for the first program with this value is made from the bottom of the invocation stack to the top. If no program is found with the correct value in this field, the top program in the invocation stack is identified in the journal prefix. If this field contains a value of 0, the program name will not appear in journal prefix of entries created through this journal port.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Insert
  - Access group identified in operand 2
  - User profile of creating process
  - Context identified in operand 2

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Modify
  - Access group identified in operand 2
  - Context identified in operand 2
  - User profile of creating process

Events

0002 Authorization
- 0101 Authorization violation

000C Machine resources
- 0201 Machine auxiliary storage exceeded
- 0501 Machine address threshold exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

Exception	**Operands**		**Other**
	1	**2**	
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context Operation			
01 Duplicate object identification		X	
10 Damage			
04 System object damage		X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
01 Template value invalid		X	

CREATE JOURNAL SPACE (CRTJS)

Op Code (Hex)	Operand 1	Operand 2
05AE	Journal space	Journal space template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction creates a journal space according to the description in operand 2 and returns addressability in the system pointer identified by operand 1. A journal space must be created as a permanent object.

The format of the journal space template is as follows:

• Template size	Char(8)
– Number of bytes provided by user	Bin(4)*
– Number of bytes that can be materialized	Bin(4)*
• Object identification	Char(2)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char (30)
• Object creation options	Char(4)
– Reserved (binary 0)	Bit 0
– Space attributes	Bit 1
0 = Fixed-length	
1 = Variable-length	
– Initial context	Bit 2
0 = No addressability insert	
1 = Insert addressability	
– Reserved (binary 0)	Bits 3-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
– Space alignment	Bit 0
0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.	
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.	
– Reserved (binary 0)	Bits 1-4
– Main storage pool selection	Bit 5
0 = Process default main storage pool is used for object.	
1 = Machine default main storage pool is used for object.	
– Transient storage pool selection	Bit 6
0 = Default main storage pool (process default or machine default as specified for main storage pool selection) is used for object.	
1 = Transient storage pool is used for object.	
– Block transfer on implicit access state modification	Bit 7
0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.	
1 = Transfer the machine default storage transfer size. This value is 8 storage units.	
– Unit number	Bits 8-15
– Reserved (binary 0)	Bits 16-31
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Reserved (binary 0)	Char(16)
• Threshold	Bin(4)

The journal space template must be aligned on a multiple of 16 bytes.

Note: The values of the entries annotated with an asterisk (*) are ignored by this instruction.

The journal space is owned by the user profile governing process execution. The owning user profile is implicitly assigned all authority states to the journal space. The storage occupied by the journal space is charged to this owning user profile.

The object identification specifies the symbolic name that identifies the journal space within the machine. A type code of hex 07 is implicitly supplied by the machine. The object identification is used to identify the journal space on materialize instructions as well as to locate the object in a context that addresses the object.

A space may be associated with the created object. This space may be fixed or variable. Initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated is dependent on an algorithm defined by a specific implementation. A fixed-size space request of zero length results in no space being allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this byte value is also used to initialize the new allocation.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created object is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

The performance class parameter provides information allowing the machine to more effectively manage the object, considering the overall performance objective of operations involving the journal space.

The threshold field contains a value that indicates the threshold value for this journal space. It must be a value equal to or greater than 0. A journal space threshold reached event is signaled when a journal entry is placed in the journal space and causes more than the space indicated in the threshold value to be used. This size includes the total object size for the journal space as provided by the Materialize System Object instruction. The event is signaled only once per journal space. If a value of 0 is provided in this field, the event will not be signaled.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
- Insert
  - Access group identified in operand 2
  - User profile of creating process
  - Context identified in operand 2

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Access group identified in operand 2
  - Context identified in operand 2
  - User profile of creating process

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage threshold exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

001C Journal space

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
0E Context Operation			
01 Duplicate object identification		X	
10 Damage			
04 System object damage		X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
01 Template value invalid		X	

DESTROY JOURNAL PORT (DESJP)

Op Code (Hex)	Operand 1
05AD	Journal port

Operand 1: System pointer.

Description: This instruction destroys the specified journal port and deletes addressability to the journal port from any context that currently addresses the object. The pointer identified by operand 1 is not modified by the instruction.

If any objects are currently being journaled through the journal port identified in operand 1, an object not eligible for destruction exception is signaled and the journal port is not destroyed.

If a journal space is attached to the journal port identified in operand 1, an object not eligible for destruction exception is signaled and the journal port is not destroyed.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Object control
  - Operand 1

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

- Object control
  - Operand 1

- Modify
  - Access group that contains operand 1
  - Context that addresses operand 1
  - User profile owning operand 1

Events

0002 Authorization
    0401 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0401 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation		X
0A Authorization		
01 Unauthorized for operation	X	
10 Damage		
04 System object damage	X	X
1A Lock State		
01 Invalid lock state	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	

DESTROY JOURNAL SPACE (DESJS)

Op Code (Hex)	Operand 1
05A1	Journal space

Operand 1: System pointer.

Description: This instruction destroys the specified journal space and deletes addressability to the journal space from any context that currently addresses the object. The pointer identified by operand 1 is not modified by the instruction.

If the journal space is currently attached to a journal port as a receiver, an object not eligible for destruction exception is signaled.

If any journal entries exist on the journal space that are needed to recover an object that has journaled changes, a journal space not at a recoverable boundary exception is signaled and the journal space is not destroyed.

If any journal entries that exist on the journal space were generated under commitment control and the changes have not been committed or decommitted, a journal space not at a recoverable boundary exception is signaled and the journal space is not destroyed.

Events

0002 Authorization
- 0401 Authorization violation

000C Machine resources
- 0201 Machine auxiliary storage exceeded

0010 Process
- 0601 Exception signaled to process
- 0701 Maximum processor time exceeded

0016 Machine observation
- 0401 Instruction reference
- 0201 Object location reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

001A Journal port
- 0301 Entry not journaled
- 0401 Journal space attached to a journal port became unusable

001C Journal space
- 0301 Threshold reached

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation		X
0A Authorization		
01 Unauthorized for operation	X	
10 Damage		
04 System object damage	X	X
1A Lock State		
01 Invalid lock state	X	X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
30 Journal Management		
06 Journal space not at recoverable boundary	X	
32 Scalar Specification		
01 Scalar type invalid	X	

JOURNAL DATA (JRNLD)

Op Code (Hex)	Operand 1	Operand 2
05B2	Journal port	User data template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction places the data specified in the template identified by operand 2 in the journal spaces attached to the journal port specified by operand 1.

The format of the entry template is as follows:

• Subtype of entry	Char(2)
• Length of entry	Bin(2)
• Journal data options	Char(2)
– Force journal spaces	Bit 0
0 = Do not force journal spaces.	
1 = Force journal spaces.	
– Reserved (binary 0)	Bits 1-15
• Journal ID	Char(10)
• Reserved (binary 0)	Char(16)
• Entry	System pointer

If the journal ID field contains a value of nonzero, the value will be placed in the journal ID field of the journal entry. If this value is binary 0's, it will be ignored, and binary 0's will appear in the generated journal entry. This journal ID may correspond to any journal ID currently used by an object being journaled through the indicated journal port; however, it need not correspond to any of these objects. The indicated journal ID can be used in the input template for the Retrieve Journal Entries instruction, and if it matches the journal ID of an object that has changes being journaled through the indicated journal port, it will be returned with entries for the object when the appropriate selection options are specified.

The entry will be placed in the user-defined portion of the journal entry for a length defined in the length of entry field.

Authorization Required

- Retrieve
  - – Contexts referenced for address resolution
- Insert
  - – Operand 1

Lock Enforcement

- Materialization
  - – Contexts referenced for address resolution
- Modify
  - – Operand 1

Events

0002 Authorization
  0101 Authorization violation

000C Machine resources
  0201 Machine auxiliary storage exceeded

0010 Process
  0701 Maximum processor time exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage
  0401 System object damage set
  0801 Partial system object damage set

001A Journal port
  0301 Entry not journaled
  0401 Journal space attached to a journal port became unusable

001C Journal space
  0301 Threshold reached

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage			
04 System object damage	X		X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state	X	X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
30 Journal Management			
02 Entry not journaled			X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	
38 Template Specification			
01 Template value invalid		X	

JOURNAL OBJECT (JRNLOBJ)

Op Code (Hex)	Operand 1	Operand 2
05BA	Object to be journaled	Journal object template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction indicates that the changes to the specified object are to be journaled or that no more changes are to be journaled to the specified object. If a journal port is specified in operand 2, the object specified in operand 1 is modified to the journal changes and journal ID through the specified journal port. If the object already has changes being journaled, an object already being journaled exception is signaled. Operand 2 can also stop journaling changes to the indicated object if the journal port field is null (binary 0's).

The format of the journal object template is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for mat	Bin(4)*
– Number of bytes available for mat	Bin(4)*
• Object journal attributes	Char(8)
– Object is being journaled	Bit 0*
– Before images 0 = Do not journal before images whenever the entry/image is optional 1 = Journal before images whenever the entry is optional	Bit 1
– After images 0 = Do not journal after images whenever the entry/image is optional 1 = Journal after images whenever the entry is optional	Bit 2
– Synchronized with journal	Bit 3*
– Journal optional entries 0 = Journal all entries defined for this object type 1 = Do not journal optional entries defined for this object type	Bit 4
– Reserved	Bits 5-63
• Journal port	System pointer
• Journal ID	Char(10)

The journal port template must be aligned on a multiple of 16 bytes.

Note: The values of the entries annotated with an asterisk (*) are ignored by this instruction.

If the before images bit has a value of binary 1, the system will generate before images whenever a change is made to an object that is being journaled and the change has an architected before image. If this field has a value of binary 0, the system will not generate the before images on the journal spaces.

If the after images bit has a value of binary 1, the system will generate after images whenever a change is made to an object that is being journaled and the change has an architected after image. This field must contain a value of binary 1, or a template value invalid exception is signaled.

If the do not journal optional entries bit has a value of binary 1, the system will not generate the optional journal entries defined for this object type. If the bit has a value of binary 0, all journal entries defined for this object type are generated as needed.

If the journal port field contains a null value (binary 0's), the object journal attributes and journal ID fields are ignored. If the journal port field contains a null value (binary 0's), the object will no longer be journaled through any journal port. If the object is not currently being journaled, the operation completes normally (no exception); however, the journal ID that was assigned to the object remains with the object and can be determined through the Materialize Journaled Object Attributes instruction. If the operand 1 object is damaged, the operation (stop journaling activity) will be completed and the object will no longer be journaled through any journal port. If there are no journal spaces attached to the journal port through which the object's changes were being journaled or if the journal port or journal spaces are damaged, the operation is performed, but no journal entries are placed in the user's journal identifying the operation performed.

The journal ID will be used as the journal ID for each journal entry for this object. A value of binary 0's for the journal ID will result in a template value invalid exception. The journal ID that is established for the object will remain with the object until it is reestablished through another execution of this instruction or the object is destroyed. The journal ID must be unique for each object being journaled through the journal port. If the journal ID is not unique, a journal ID not unique exception is signaled.

If operand 2 indicates that the object is to no longer be journaled, this will be done regardless of whether or not a journal entry can be generated and placed on the proper journal receiver(s).

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
- Object management
  - Operand 1
  - Journal port referenced in operand 2

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution
- Object control
  - Operand 1
- Modify
  - Journal port referenced in operand 2

Events

0002 Authorization
0401 Authorization violation

000C Machine resources
0201 Machine auxiliary storage exceeded

0010 Process
0701 Maximum processor time exceeded

0016 Machine observation
0401 Instruction reference

0017 Damage set
0401 System object damage set
0801 Partial system object damage set

001A Journal port
0301 Entry not journaled
0401 Journal space attached to a journal port became unusable

001C Journal space
0301 Threshold reached

Exceptions

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Subscript range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage			
04 System object damage	X	X	X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object type invalid	X		
04 Object not eligible for operation	X		
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
30 Journal Management			
02 Entry not journaled			X
03 Journaled object limit exceeded			X
07 Journal ID not unique		X	
08 Object already being journaled	X		
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
01 Template value invalid		X	

MATERIALIZE JOURNAL PORT ATTRIBUTES (MATJPAT)

Op Code (Hex)	Operand 1	Operand 2
05A6	Receiver	Journal port

Operand 1: Space pointer.

Operand 2: System pointer.

Description: This instruction materializes the creation attributes of the journal port specified by operand 2 and places the attributes in the receiver specified by operand 1.

The format of the materialization data is as follows:

• Materialization length	Char(8)
– Number of bytes provided by user	Bin(4)
– Number of bytes that can be materialized	Bin(4)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes (binary 1)	Bit 0
– Space attributes 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context	Bit 2
– Access group	Bit 3
– Replace option	Bit 4
– Reserved	Bits 5-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
• Reserved	Char(7)
• Context	System pointer
• Reserved (binary 0)	Char(16)

• Length of timestamp in prefix	Bin(2)
• Length of process name in prefix	Bin(2)
• Length of user profile name in prefix	Bin(2)
• Length of program name in prefix	Bin(2)
• Number of journal spaces attached to the journal port	Bin(2)
• Reserved	Char(6)
• Journal spaces (0 to n)	System pointer

The first 4 bytes of the receiver must be aligned on a 16-byte boundary. The first 8 bytes of the materialization output identify the total quantity of bytes provided by the user for the materialization and the total quantity of bytes available to be materialized. If fewer than 8 bytes are available in the space identified by the receiver, operand 1, a materialization length exception is signaled. The instruction materializes as many bytes as can be contained in the receiver's space. If the space of the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient space for the materialization other than the materialization length exception described previously.

The attributes materializable by the instruction are described in the Create Journal Port instruction.

Each journal space currently attached to the journal port will be identified in the journal space list. This list has as many entries as are identified in the number of journal spaces attached to the journal port field.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Operational
  - Operand 2

Lock Enforcement

- Materialization
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage			
04 System object damage		X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE JOURNAL SPACE ATTRIBUTES (MATJSAT)

Op Code (Hex)	Operand 1	Operand 2
05BE	Receiver	Journal space

Operand 1: Space pointer.

Operand 2: System pointer.

Description: This instruction materializes the current attributes of the journal space specified by operand 2 and places the attributes in the receiver specified by operand 1.

The format of the materialization data is as follows:

• Materialization length	Char(8)
– Number of bytes provided by user	Bin(4)
– Number of bytes materializable	Bin(4)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes (binary 1)	Bit 0
– Space attributes 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context	Bit 2
– Access group	Bit 3
– Replace option	Bit 4
– Reserved	Bits 5-31
• Reserved (binary 0)	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Reserved (binary 0)	Char(16)
• Threshold	Bin(4)

• Reserved (binary 0)	Char(12)
• Journal port	System pointer
• Number of journal entries	Bin(4)
• First sequence number	Bin(4)
• Last sequence number	Bin(4)
• Reserved (binary 0)	Char(4)
• Time journal space attached to journal port	Char(8)
• Time journal space detached to journal port	Char(8)
• Length of timestamp	Bin(2)
• Length of process name	Bin(2)
• Length of user profile name	Bin(2)
• Length of program name	Bin(2)
• Length of longest journal entry	Bin(4)
• Last journal entry dumped	Bin(4)
• Journal space status	Char(2)
– Operable journal space 0 = Journal space is operable 1 = Journal space is not operable	Bit 0
– Missing journal entries 0 = No entries missing 1 = 1 or more entries missing	Bit 1
– Journal space size extension 0 = Journal space could be extended 1 = Journal space could not be extended	Bit 2
– Maximum sequence number reached 0 = Maximum sequence number has not been reached 1 = Maximum sequence number has been reached	Bit 3
– Journal failure 0 = No journal failure has occurred 1 = A journal failure has occurred	Bit 4
– Reserved (binary 0)	Bits 5-15

The first 4 bytes of the receiver must be aligned on a 16-byte boundary. The first 8 bytes of the materialization output identify the total quantity of bytes provided by the user for the materialization and the total quantity of bytes available to be materialized. If fewer than 8 bytes are available in the space identified by the receiver (operand 1), a materialization length exception is signaled. The instruction materializes as many bytes as can be contained in the receiver's space. If the space of the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient space for the materialization other than the materialization length exception described previously.

The attributes that can be materialized by the instruction are described in the Create Journal Space instruction.

The journal port field contains a system pointer to the journal port to which the designated journal space is currently attached. If the journal space is not currently attached to a journal port, this field will contain binary 0's.

The number of journal entries field contains the number of journal entries currently in the journal space. If this field contains a value of 0, the journal space has never been attached to a journal port.

The first sequence number field contains the journal sequence number of the first journal entry contained in this journal space. If the number of journal entries field contains a value of 0, this field will also contain a value of 0.

The last sequence number field contains the journal sequence number of the last journal entry contained in this journal space. If the number of journal entries field contains a value of 0, this field will also contain a value of 0.

The time journal space attached to journal port field contains a timestamp that indicates the time the journal space was attached to a journal port. If the journal space has never been attached to a journal port, a value of 0 will be returned in this field.

The time journal space detached from journal port field contains a timestamp that indicates the time the journal space was detached from a journal port. If the journal space has never been attached to a journal port or is currently attached to a journal port, a value of 0 will be returned in this field.

The length of timestamp field contains the length of the timestamp field in the journal prefix of journal entries contained on the journal space. If the journal space has never been attached to a journal receiver (number of journal entries is equal to 0), this field will contain a value of 0.

The length of process name field contains the length of the process name field in the journal prefix of journal entries contained on the journal space. If the journal space has never been attached to a journal receiver (number of journal entries is equal to 0), this field will contain a value of 0.

The length of user profile name field contains the length of the user profile name field in the journal prefix of journal entries contained on the journal space. If the journal space has never been attached to a journal receiver (number of journal entries is equal to 0), this field will contain a value of 0.

The length of program name field contains the length of the program name field in the journal prefix of journal entries contained on the journal space. If the journal space has never been attached to a journal receiver (number of journal entries is equal to 0), this field will contain a value of 0.

The length of longest journal entry field contains the length of the longest journal entry currently contained on the journal space. This is the length of the entry as it would appear through the Retrieve Journal Entries instruction. If the journal space has never been attached to a journal port or there are no entries on the journal space, a value of 0 will be returned in this field.

The last journal entry dumped field contains the journal sequence number of the last complete journal entry that has been dumped from this journal space. If no dump operation has been performed on this journal space, a value of 0 will be returned.

The journal space status fields indicate whether or not the journal space is currently actively receiving journal entries or successfully received all journal entries while it was attached to a journal port. These fields also indicate the reason journal entries were not placed on the journal space.

The operable journal space field indicates whether or not journal entries are being placed in the journal space while it is attached to the indicated journal port. If the journal space is no longer attached to a journal port, this field indicates the status of the journal space when it was detached from the journal port.

The missing journal entries field indicates whether or not journal entries have been created, while this journal space was attached to a journal port, that were not recorded on this journal space. If this field contains a value of binary 1, the Apply Journaled Changes and Retrieve Journal Entries instructions will not continue beyond the last journal entry recorded on this journal space.

The journal space size extension field indicates whether or not the journal space can be extended. A value of binary 1 in this field indicates a storage limit exceeded exception was encountered while trying to extend the journal space.

The journal failure field indicates whether or not a journal failure occurred while this journal space was attached to a journal port.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Operational
  - Operand 2

Lock Enforcement

- Materialization
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage			
04 System object		X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE JOURNALED OBJECT ATTRIBUTES (MATJOAT)

Op Code (Hex)	Operand 1	Operand 2
05B6	Receiver space	Object

Operand 1: Space pointer.

Operand 2: System pointer.

Description: This instruction returns the journal ID and the address of the journal port for an object if it is being journaled. If the object is not being journaled, it returns an indication that it is not journaled (binary 0 for the address of the journal port) and the journal ID the object had when it was last journaled. If the journal ID is binary 0's, then the object has never been journaled.

The format of the receiver space is as follows:

• Materialization size specification	Char(8)
– Number of bytes provided for mat	Bin(4)
– Number of bytes available for mat	Bin(4)
• Object journal attributes	Char(8)
– Object is being journaled 0 = Not journaled 1 = Journaled	Bit 0
– Before images 0 = Journal does not contain before images whenever the entry/image is optional 1 = Journal contains before images whenever the entry is optional	Bit 1
– After images 0 = Journal does not contain after images whenever the entry or image is optional 1 = Journal contains after images whenever the entry is optional	Bit 2
– Synchronized with journal 0 = Object is currently synchronized with the indicated journal 1 = Object is not currently synchronized with the indicated journal	Bit 3
– Journal optional entries 0 = Journal all entries defined for this object type 1 = Do not journal optional entries defined for this object type	Bit 4
– Reserved	Bits 5-63
• Journal port	System pointer
• Journal ID	Char(10)

The receiver space must be aligned on a 16-byte boundary. The first 8 bytes of the materialization output identify the total quantity of bytes provided by the user for the materialization and the total quantity of bytes available to be materialized. If fewer than 8 bytes are available in the space identified by the receiver (operand 1) a materialization length exception is signaled. The instruction materializes as many bytes as can be contained in the receiver's space. If the space of the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions are signaled when the receiver contains insufficient space for the materialization other than the materialization length exception described previously.

If the object is being journaled field has a value of binary 0, all other fields except the journal ID and the materialization specification will contain binary 0's.

An attempt is made to synchronize the journal and the objects being journaled to it at each IPL. If this cannot be done, the object is marked as not synchronized with the journal. Either the load or dump operation will reestablish an object as synchronized with the journal.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Operational
  - Operand 2

Lock Enforcement

- Materialization
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage			
04 System object damage		X	
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
03 Materialization length exception	X		

MATERIALIZE JOURNALED OBJECTS (MATJOBJ)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
05A7	Receiver	Journal port	Materialize option

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar.

Description: This instruction materializes the system pointers to the objects that are currently being journaled through the journal port specified by operand 2 and places them in the receiver (operand 1). It also returns the object ID and journal ID of the object if these options are specified by operand 3.

The materialization options are as follows:

• Materialize options	Char(1)
– Materialize pointer to object	Bit 0
– Materialize object ID	Bit 1
– Materialize journal ID	Bit 2
– Reserved (binary 0)	Bits 3-7

If at least one materialize option is not requested, a scalar value invalid exception is signaled.

The receiver identified by operand 1 must be 16-byte aligned in the space and have the following format:

• Materialization size specification	Char(8)
– Number of bytes provided	Bin(4)
– Number of bytes available	Bin(4)
• Number of objects being journaled	Bin(4)
• Reserved (binary 0)	Char(4)
• Object data (0 to n)	Char(*)

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the bytes provided is greater than the bytes available, the excess bytes are unchanged.

For each journaled object there is an entry. These entries are in arbitrary order.

If materialize pointer to object is specified, then the system pointer is returned first in each entry.

• Journaled object	System pointer

If materialize object ID is specified, then the system pointer (if requested) is immediately followed by the object ID of the form:

• Object ID	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)

If materialize journal ID is specified, then the journal ID follows the object ID (if requested) and/or the system pointer (if requested). The journal ID is of the form:

• Journal ID	Char(10)
• Reserved (binary 0)	Char(6)

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
  - Operand 2

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution
  - Operand 2

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X		
02 Boundary alignment	X	X		
03 Range	X	X		
06 Optimized addressability invalid	X	X		
08 Argument/Parameter				
01 Parameter reference violation	X	X		
0A Authorization				
01 Unauthorized for operation		X		
10 Damage				
04 System object damage		X		
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X		
02 Object destroyed	X	X		
03 Object suspended	X	X		
24 Pointer Specification				
01 Pointer does not exist	X	X		
02 Pointer type invalid	X	X		
03 Pointer addressing invalid object	X	X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X		
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MODIFY JOURNAL PORT (MODJP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
05AB	Detached object template	Options template	Journal port

Operand 1: Space pointer or null.

Operand 2: Space pointer.

Operand 3: System pointer.

Description: This instruction attaches journal spaces to the journal port identified in operand 3 according to the options specified in operand 2.

All journal spaces previously attached to the journal port will be detached. Addressability to any journal spaces detached from the journal port are returned in the operand 1 template unless a null value is provided in operand 1. The journal sequence may also be reinitialized when the journal spaces are attached to the journal port.

The receiver identified by operand 1 must be 16-byte aligned in the space and have the following format:

• Materialization size specification	Char(8)
– Number of bytes provided	Bin(4)
– Number of bytes available	Bin(4)
• Number of journal spaces detached	Bin(2)
• Journal entry sequence number of last entry	Bin(4)
• Reserved (binary 0)	Char(2)
• Journal space (0 to n)	System pointer

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the number of bytes provided is greater than the number of bytes available, then the excess bytes are unchanged.

The number of journal spaces detached field contains the number of journal spaces that were previously attached to the journal port. This field also indicates the number of entries contained in the journal space field.

The journal entry sequence number of last entry field contains the journal entry sequence number of the last journal entry generated on the journal spaces that were detached from the journal port.

A system pointer to each journal space detached is returned in the journal space field.

The format of the options template is as follows:

• Options	Char(2)
– Reset journal sequence number	Bit 0
0 = Do not reset journal sequence number	
1 = Reset journal sequence number	
– Reserved (binary 0)	Bits 1-15
• Number of journal spaces to attach	Bin(2)
• Reserved (binary 0)	Char(12)
• Journal space to be attached (0 to n)	System pointer

The options template must be aligned on a multiple of 16 bytes.

The number of journal spaces to attach field must contain a value equal to or greater than 0 and indicate how many journal spaces are to be attached to the designated journal port. The journal space to be attached system pointer addresses a journal space that is to be attached as a receiver. If this field has a value of binary 0, no journal spaces are attached to the journal port as receivers and the journal space to be attached field is ignored. If any objects that have changes being journaled through the referenced journal port are in use and this field has a value of binary 0, a template value invalid exception is signaled.

The journal space to be attached system pointer addresses a journal space that is to be attached as a receiver. If the system journal space is referenced more than once in the template, a template value invalid exception is signaled.

If the reset journal sequence number contains a value of binary 1, the journal sequence number of the first entry on the attached journal space(s) will be 1. If this field contains a value of binary 1, all objects currently being journaled through the journal port must be at a recoverable boundary. A recoverable boundary is that point at which no entries for the object on the detached journal space are needed for any subsequent IPL synchronization. If not, a journal space not at recoverable boundary exception is signaled. If this field contains a value of binary 0, the first journal entry on the attached journal spaces will be 1 greater than the journal sequence number of the last entry on the detached journal space(s).

A journal entry will be generated and added to the journal spaces identifying the attached and detached journal spaces. Once a journal space has been attached to a journal port, it may never be attached to any journal port again. If an attempt to perform this operation is made, an object not eligible for operation exception is signaled.

Authorization Required

- Retrieve
  – Contexts referenced for address resolution

- Update
  – Operand 3

- Object management
  – Operand 1 (journal spaces to be detached)
  – Operand 2

Lock Enforcement

- Materialization
  – Contexts referenced for address resolution
  – Operand 1 (journal spaces to be detached)

- Modify
  – Operand 3
  – All journal spaces whose status as journal receivers is to be modified

Events

0002 Authorization
    0401 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage reached

0010 Process
    0701 Maximum processor time exceeded

0016 Machine observation
    0401 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

001A Journal port
    0301 Entry not journaled
    0401 Journal space attached to a journal port became unusable

001C Journal space
    0301 Threshold reached

Exceptions

Exception	Operands			Other
	1	2	3	
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation	X	X		
10 Damage				
04 System object damage	X	X	X	
44 Partial system object damage	X	X		
1A Lock State				
01 Invalid lock state	X			
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
04 Object not eligible for operation			X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X	X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
30 Journal Management				
02 Entry not journaled				X
06 Journal space not at recoverable boundary				X
38 Template Specification				
03 Materialization length exception	X			

RETRIEVE JOURNAL ENTRIES (RETJENT)

Op Code (Hex)	Operand 1	Operand 2
05CA	Receiver	Option template

Operand 1: Space pointer.

Operand 2: Space pointer.

Description: This instruction retrieves journal entries from the indicated journal spaces and places these journal entries in the receiving area specified by operand 1. The journal spaces indicated in the options template are scanned according to the rules and options specified in the options template and returned in the receiving space identified in operand 1. For a journal entry to be returned in the receiving area, it must match all of the selection criteria specified in the options template.

The format of the receiver data is as follows:

• Materialization length	Char(8)
– Number of bytes provided by user	Bin(4)
– Number of bytes returned	Bin(4)*
• Number of journal entries returned	Bin(4)
• First journal sequence number returned	Bin(4)
• Last journal sequence number returned	Bin(4)
• Entry search status	Char(2)
• Journal entry	Char(*)

Note: The input values of the entries annotated with an asterisk (*) are ignored by this instruction.

The first 4 bytes of the receiver must be aligned on a 16-byte boundary. The first 8 bytes of the materialization output identify the total quantity of bytes provided by the user for the materialization and the total quantity of bytes available to be materialized. If fewer than 22 bytes are available in the space identified by the receiver (operand 1), a template size invalid exception is signaled. The instruction materializes as many bytes as can be contained in the receiver's space. If the space of the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions, other than the template size invalid exception described previously are signaled in the event that the receiver contains insufficient space for the materialization.

The number of journal entries returned field is set to the number of journal entries returned by the instruction in the receiving area.

The first journal sequence number returned field contains the journal sequence number of the first journal entry returned in the receiving area. If the number of journal entries returned is equal to 0, this field contains a value of 0. Zero is not a valid journal sequence number.

The last journal sequence number returned field contains the journal sequence number of the last journal entry returned in the receiving area. If the number of journal entries returned is equal to 0, this field contains a value of 0. Zero is not a valid journal sequence number.

The entry search status indicates whether or not any entries were found that matched the criteria established in the option template. A value of hex 0001 indicates that at least 1 journal entry was found that met the criteria. A value of hex 0002 indicates that no entries were found that met the criteria.

When multiple entries are returned in the receiver area (indicated by a value greater than 1 in the number of journal entries returned field), the length of journal entry and journal entry fields are repeated multiple times with no intervening space in the receiver area.

The machine generates journal entries whenever the indicated change occurs in the system. When the Retrieve Journal Entries instruction returns the specific entries in the receiving area, they have the following formats. Each entry has the same initial fields in their format. These common fields are:

• Journal entry length	Bin(4)
• Journal sequence number	Bin(4)
• Entry type	Char(1)
• Entry subtype	Char(2)
• Status byte	Char(1)
– Journal ID present	Bit 0
– Commit	Bit 1
– Reserved	Bit 2-7
• Journal prefix	Char(*)
– Timestamp	Char(*)
– PCS name	Char(*)
– User profile name	Char(*)
– User program name	Char(*)
• Journal ID	Char(10)
• Commit ID	Char(4)

The entry specific data immediately follows the prefix information and is further described for each entry type.

The entry type corresponds to the object type of the object which is having its change activity journaled.

The entry specific data is determined by the entry subtype for each entry. The entry specific data that is provided for each of the entry subtypes is as follows:

- Entry subtype hex 0010 = Start journaling object

– Object identifier	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Journal attributes	Char(8)

- Entry subtype hex 0011 = Stop journaling object

– Object identifier	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)

- Entry subtype hex 0020 = Object synchronized with journal
  - No specific data

- Entry subtype hex 0022 = Object not synchronized with journal
  - No specific data

- Entry subtype hex 0030 = Start of apply journal changes

– Apply options	Char(4)
– Starting sequence number specified	Bin(4)
– Ending sequence number specified (this entry is not generated if journal entries were only scanned)	Bin(4)

- Entry subtype hex 0031 = Apply journal changes termination

– Apply options	Char(4)
– Starting sequence number specified	Bin(4)
– Ending sequence number specified	Bin(4)
– Number of entries reapplied	Bin(4)
– Completion code (exception ID) (If hex 0000, the apply finished normally)	Char(2)
– First sequence number applied	Bin(4)
– Last sequence number applied	Bin(4)
– First journal space supplied	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier (for first journal space)	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Last journal space supplied	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier (for last journal space)	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Object dependent status field (as defined on the apply journaled changes object list) This entry is not generated if journal entries were only scanned.	Char(16)

- Entry subtype hex 0040 = Object loaded

– Load related data (provided through Request I/O)	Char(*)

- Entry subtype hex 0042 = Object dumped

– Dump related data (provided through Request I/O)	Char(*)

- Entry subtype hex 0050 = Object suspended
  - No specific data

- Entry subtype hex 0060 = Object destroyed
  - No specific data

Entry type hex 09 is associated with entries that pertain to journal ports only. These entries have the entry subtypes and entry specific data as follows:

- Entry subtype hex 0101 = System failure (recorded at IPL
  - No specific data

- Entry subtype hex 0102 = System IPL after normal termination
  - No specific data

- Entry subtype hex 0110 = Receiver attached to journal port

– Number attached (The next 2 fields are repeated for each journal space attached)	Bin(2)
– Object identifier (journal space)	Char(32)
Object type (hex 07)	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier (journal space)	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)

- Entry subtype hex 0120 = Receiver detached from journal port

– Number detached (The next 2 fields are repeated for each journal space detached)	Bin(2)
– Object identifier (journal space)	Char(32)
Object type (hex 07)	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier (journal space)	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)

Entry type hex 0B is associated with entries that pertain to data spaces only. These entries have the entry subtypes and entry specific data as follows:

- Entry subtype hex 0210 = Activate cursor

– Cursor identifier	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Cursor attributes (from the cursor activation template)	Char(2)
– Extended activation functions (from the cursor activation template)	Char(2)
– Entry count	Bin(4)
– Deleted entry count	Bin(4)
– Entry limit	Bin(4)

- Entry subtype hex 0212 = Deactivate cursor

– Cursor identifier	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Entry count	Bin(4)
– Deleted entry count	Bin(4)
– Entry limit	Bin(4)

- Entry subtype hex 0220 = Data space reset
  - No specific data

- Entry subtype hex 0224 = Increment entry limit

– New entry limit	Bin(4)
– Increment size	Bin(4)

- Entry subtype hex 0226 = Insert default entries

– First ordinal number inserted	Bin(4)
– Number of entries inserted	Bin(4)
– Entry count	Bin(4)
– Deleted entry count	Bin(4)
– Entry limit	Bin(4)

- Entry subtype hex 0228 = Insert deleted entries

– First ordinal number inserted	Bin(4)
– Number of entries inserted	Bin(4)
– Entry count	Bin(4)
– Deleted entry count	Bin(4)
– Entry limit	Bin(4)

- Entry subtype hex 0230 = Copy data space entries (source data space and receiver data space are the same)

– Cursor identifier (source)	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier (source)	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Cursor identifier (receiver)	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Context identifier (receiver)	Char(32)
Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)
– Copy options template (from the copy options template on copy data space entries)	Char(48)
– Entry count	Bin(4)
– Deleted entry count	Bin(4)
– Entry limit	Bin(4)

- Entry subtype hex 0240 = Data space forced

– Entry count	Bin(4)
– Deleted entry count	Bin(4)
– Entry limit	Bin(4)

- Entry subtype hex 0250 = Insert data space entry

– Ordinal entry number	Bin(4)
– Data space entry (not provided if after images not selected, length = 0)	Char(*)

- Entry subtype hex 0260 = Update data space entry (before image)
  - Ordinal entry number Bin(4)
  - Data space entry Char(*)

    Notes:

    1. This entry is not provided if before images are not selected.
    2. This entry is provided if before images are selected even if the change is an update of a deleted entry. The data space entry field will not be returned in this case, however.

- Entry subtype hex 0261 = Update data space entry (after image)
  - Ordinal entry number Bin(4)
  - Data space entry Char(*)

    Note: This entry is not provided if after images are selected.

- Entry subtype hex 0264 = Update deleted data space entry
  - Ordinal entry number Bin(4)
  - Data space entry (not provided if after images not selected, length = 0) Char(*)

- Entry subtype hex 0270 = Delete data space entry
  - Ordinal entry number Bin(4)
  - Data space entry (not provided if before images not selected, length = 0) Char(*)

- Entry subtype hex 0271 = Delete deleted data space entry
  - Ordinal entry number Bin(4)

- Entry subtype hex 02E0 = Update data space entry (before image-decommit)
  - Ordinal entry number Bin(4)
  - Data space entry Char(*)

- Entry subtype hex 02E1 = Update data space entry (after image)
  - Ordinal entry number Bin(4)
  - Data space entry Char(*)

- Entry subtype hex 02E4 = Update deleted data space entry (decommit)
  - Ordinal entry number Bin(4)
  - Data space entry Char(*)

- Entry subtype hex 02F0 = Delete data space entry (decommit)
  - Ordinal entry number Bin(4)
  - Data space entry Char(*)

- Entry subtype hex 02F1 = Delete deleted data space entry (decommit)
  - Ordinal entry number Bin(4)

Entry type hex 0F is associated with entries that pertain to commit blocks only. These entries have the entry subtypes and entry specific data as follows:

- Entry subtype hex 0503 = Attach commit block
  - No specific data

- Entry subtype hex 0504 = Detach commit block
  - No specific data

- Entry subtype hex 0510 = Start of commit group
  - No specific data

- Entry subtype hex 0513 = Cursor commit data
  - Internal data area Char(*)

- Entry subtype hex 0520 = Commit
  - Length of commit description Bin(4)
  - Commit description Char(*)

- Entry subtype hex 0530 = Decommit
  - Constant (hex 0F) Char(1)
  - Decommit status Char(4)

    Damaged Bit 0

    0 = Commit block is not damaged

    1 = Commit block is damaged

    Reserved Bits 1-2

    Partially damaged Bit 3

    0 = Commit block is not partially damaged

    1 = Commit block is partially damaged

Reserved Bits 4-15

Decommit Bit 16

0 = The journal has successfully been read backwards until either a start commit or a decommit entry was found. An attempt has been made to decommit all the data base changes but the attempt may not have been successful if the data space is damaged or if the function check flag is on.

1 = The journal has not successfully been read backwards to a start commit or decommit entry and, consequently all the changes have not been decommitted.

Journal read errors Bit 17

0 = No journal read errors

1 = Journal read errors occurred during decommit

Journal write errors Bit 18

0 = No journal write errors

1 = Journal write errors occurred during decommit

Partial damage to data space Bit 19

0 = No partial damage encountered

1 = Partial damage encountered on 1 or more data spaces

Damage to data space Bit 20

0 = No damage encountered

1 = Damage encountered on 1 or more data spaces

Function check Bit 21

0 = No function check encountered

1 = Function check encountered

Data space during IMPL Bit 22

0 = Data space is synchronized with the journal

1 = Data space is not synchronized with the journal. All changes may not be decommitted.

The format of the options template is as follows:

- Reserved (binary 0) Bin(4)

- Search attributes Char(4)
  - Order of search Bit 0
    - 0 = Ascending
    - 1 = Descending
  - Limit to journal IDs indicator Bit 1
    - 0 = Do not limit search to any particular journal IDs
    - 1 = Limit the search to the indicated journal IDs
  - Limit to journal type indicator Bit 2
    - 0 = Do not limit search to any particular journal type codes.
    - 1 = Limit the search to the indicated journal type codes.
  - Limit to journal subtype indicator Bits 3-4
    - 00 = Do not limit search to any particular journal subtype codes.
    - 01 = Limit the search to the indicated journal subtype codes.
    - 10 = Limit the search to all subtype codes equal to or less than the indicated subtype code.
    - 11 = Limit the search to all subtype codes equal to or greater than the indicated subtype code.
  - Limit to user profile indicator Bit 5
    - 0 = Do not limit search to any particular user profile.
    - 1 = Limit the search to the indicated user profile.
  - Limit to process name indicator Bit 6
    - 0 = Do not limit search to any particular process name.
    - 1 = Limit the search to the indicated process name.
  - Limit to program name indicator Bit 7
    - 0 = Do not limit search to any particular program name.
    - 1 = Limit the search to the indicated program name.
  - Limit to commit ID indicator Bit 8
    - 0 = Do not limit search to any particular commit ID.
    - 1 = Limit the search to the indicated commit ID.
  - Reserved (binary 0) Bits 9-31

• Start options	Char(2)
– First entry indicator	Bits 0-2
000 = First	
001 = Last	
010 = Sequence number	
011 = Timestamp	
1xx = Reserved	
– Reserved	Bits 3-15
• End options	Char(2)
– Last entry indicator	Bits 0-2
000 = First	
001 = Last	
010 = Sequence number	
011 = Timestamp	
1xx = Reserved	
– Reserved	Bits 3-15
• Number of journal entries requested	Bin(4)
• Starting journal sequence number	Bin(4)
• Ending journal sequence number	Bin(4)
• Starting timestamp	Char(8)
• Ending timestamp	Char(8)
• Number of journal spaces specified	Bin(2)
• Number of journal IDs specified	Bin(2)
• Number of journal type codes specified	Bin(2)
• Number of journal subtype codes specified	Bin(2)
• Commit ID	Bin(4)
• Reserved (binary 0's)	Char(12)
• Length of user profile name for search	Bin(2)
• User profile name	Char(30)
• Length of process name for search	Bin(2)
• Process name	Char(30)
• Length of program name for search	Bin(2)
• Program name	Char(30)
• List of journal spaces	Space pointer
• List of journal IDs	Space pointer
• Journal type code list	Space pointer
• Journal subtype code list	Space pointer

The options template must be aligned on a multiple of 16 bytes.

The list of journal spaces must also be aligned on a multiple of 16 bytes.

The search options identify the criteria for selecting entries to be returned by the instruction. It indicates the order of the search and the types of entries that will be returned in the receiving area.

If the order of search field contains a binary 0, the entries will be searched in ascending order from the lowest to the highest sequence number provided. If this field contains a binary 1, the journal entries are searched in descending order (from the highest to the lowest sequence number provided).

If the limit to journal IDs indicator field has a value of binary 1, only those entries that were generated for those objects that correspond to the journal IDs provided in the list of journal IDs will be selected. If this field has a value of binary 0, all entries for specific objects will be considered for selection.

If the limit to journal type indicator field has a value of binary 1, only the entries with a journal type code equal to the ones provided in the journal type code list will be considered for selection. If this field contains a value of binary 0, the search is not limited to journal entries of a particular journal type code.

If the limit journal subtype indicator field contains a value of binary 01, only the entries with a subtype code equal to those values contained in the journal subtype code list are considered for selection. If this field contains a value of binary 10, only those entries with a subtype code equal to or less than the single value contained in the journal subtype code list are considered for selection. If this field contains a value of binary 11, only those entries with a subtype code equal to or greater than the value contained in the single value in the journal subtype code list are considered for selection. If this field contains a value of binary 00, all subtype codes will be considered for selection.

If the limit to user profile indicator field has a value of binary 1, only those entries that contain an identifier for the user profile specified in the user profile name field will be considered for selection. If this field has a value of binary 0, entries for any user profile are considered for selection.

If the limit to process name indicator field contains a value of binary 1, only those entries that contain an identifier for the process name indicated in the process name field are considered for selection. If this field has a value of binary 0, entries for all process names are considered for selection.

If the limit to program name indicator field has a value of binary 1, only those entries that contain an identifier for the program name specified in the program name field are considered for selection. If this field has a value of binary 0, entries for any program name are considered for selection.

If the limit to commit ID indicator field has a value of binary 1, only those entries that were generated under commitment control with the commit ID specified in the commit ID field are selected. If the limit to commit ID indicator field has a value of binary 0, all entries are considered for selection. If this field has a value of binary 1 and the commit ID field is binary 0's, all entries are considered for selection even without a commit ID in the journal entry. Only one commit ID is allowed.

The start options indicate which journal entry is to be the first entry considered in the search. If the first entry indicator field contains a value of binary 000, the first entry provided in the journal spaces is considered for selection. If this field contains a value of binary 001, the last entry provided in the journal spaces is considered for selection. If this field contains a value of binary 010, the journal entry with the sequence number indicated in the starting journal sequence number field is the first journal entry considered for selection. If this field contains a value of binary 011, the first journal entry with the timestamp indicated in the starting timestamp field is the first journal entry considered for selection.

The end options indicate which journal entry is to be the last entry considered in the search. If the first entry indicator field contains a value of binary 000, the first entry provided in the journal spaces is the last entry considered for selection. If this field contains a value of binary 001, the last entry provided in the journal spaces is the last entry considered for selection. If this field contains a value of binary 010, the journal entry with the sequence number indicated in the ending journal sequence number field is the last journal entry considered for selection. If this field contains a value of binary 011, the last journal entry with the timestamp indicated in the ending timestamp field is the last journal entry considered for selection.

The number of journal entries requested field value indicates how many entries are returned in the receiving area. If the receiving area will not hold all the entries requested, only those that can be contained entirely within the indicated space of the receiving area are returned.

The starting journal sequence number field must contain a value greater than 0 if the first entry indicator field contains a value of binary 010. If the first entry indicator field contains any other value, the starting journal sequence number field is ignored.

The ending journal sequence number field must contain a value greater than 0 if the last entry indicator field contains a value of binary 010. If the last entry indicator field contains any other value, the ending journal sequence number field is ignored.

The starting timestamp field value is used as a timestamp, if the first entry indicator field contains a value of binary 011. If the search options indicate the retrieval of journal entries is to be done in an ascending manner, the first journal entry encountered with a timestamp equal to or greater than the one provided in the starting timestamp field is the first journal entry considered for selection. If the search options indicate the retrieval of journal entries is done in a descending manner, the first journal entry encountered with a timestamp equal to or less than the one provided in the starting timestamp field is the first journal entry considered for selection. If the first entry indicator field contains any other value, the starting timestamp field is ignored.

The ending timestamp field value is used as a timestamp, if the last entry indicator field contains a value of binary 011. If the search options indicate the retrieval of journal entries is to be done in an ascending manner, the first journal entry encountered with a timestamp greater than the one provided in the ending timestamp field terminates the selection process. If the search options indicate the retrieval of journal entries is to be done in a descending manner, the first journal entry encountered with a timestamp less than the one provided in the ending timestamp field terminates the selection process. If the last entry indicator field contains any other value, the ending timestamp field is ignored.

It should be noted that the journal management facilities cannot guarantee that the timestamp contained in the journal prefix of the journal entries in a journal is ascending.

The number of journal spaces specified field contains the number of journal spaces that are contained in the list of journal spaces area. This field must contain a value greater than 0. The list of journal spaces is composed of a list of system pointers to journal spaces. The journal spaces must be specified in the order that they were attached to a journal port. If the sequence numbers for entries on all journal spaces are not continuous and ascending, a invalid journal space exception is signaled and no entries are returned.

The number of journal IDs specified field contains the number of journal IDs that are contained in the list of journal IDs area. Each entry in the list of journal IDs is 10 bytes long and is assumed to be a valid journal ID. A journal ID of binary zeros is invalid and a template value invalid exception results if such a value occurs. If the limit to journal IDs indicator field has a value of binary 1, the value contained in each entry of the list of journal IDs is compared with the corresponding value in each journal entry and only those that are equal to a value provided will be considered for selection. If the limit to journal IDs indicator field contains a value of binary 1, this field must contain a value greater than 0, or a template value invalid exception is signaled. If the limit to journal IDs indicator field contains a value of binary zero, this field is ignored and the list of journal IDs is also ignored.

The number of journal type codes specified field contains the number of journal type codes that are contained in the list of journal type codes area. Each entry in the list of journal type codes is 1 byte long. If the limit to journal type indicator field has a value of binary 1, the values contained in the list of journal type codes are compared with the corresponding value in each journal entry, and only those that are equal to the values provided will be considered for selection. If the limit to journal type indicator field contains a value of binary 1, this field must contain a value greater than 0, or a template value invalid exception is signaled. An invalid journal type code in the list of journal type codes, will result in a template value invalid exception. If the limit to journal type indicator field has a value of binary 0, the values contained in the list of journal type codes is ignored.

The number of journal subtype codes specified field contains the number of journal subtype codes that are contained in the list of journal subtype codes area. Each entry in the list of journal subtype codes is 2 bytes long. If the limit to journal subtype indicator field has a value of binary 01, the values contained in the list of journal subtype codes field are compared with the corresponding value in each journal entry, and only those that are equal to the values provided will be considered for selection. If the limit to journal subtype indicator field contains a value of binary 01, this field must contain a value greater than 0, or a template value invalid exception is signaled. If the limit to journal subtype indicator field has a value of binary 10, the single (the number of journal subtype codes must be equal to 1) value contained in the list of journal subtype codes is compared with the corresponding value in each journal entry, and only those that are equal to or less than the value provided will be considered for selection. If the limit to journal subtype indicator field has a value of binary 11, the single (the number of journal subtype codes must be equal to 1) value contained in the list of journal subtype codes is compared with the corresponding value in each journal entry and only those that are equal to or greater than the value provided will be considered for selection. If the limit to journal subtype indicator field has a value of binary 00, the value contained in the list of journal subtype codes and the number of journal subtype codes is ignored.

If the limit to user profile indicator field has a value of binary 1, the value contained in the user profile name field is compared with the corresponding value in each journal entry, and only those that are equal to the value provided are considered for selection. Only that portion of the user profile name that is contained within the length specified in the length of user profile name for search field is used for this compare. If the limit to user profile indicator field contains a value of binary 1, this field must contain a value from 1 to 30. If the limit to user profile indicator field has a value of binary 0, the value contained in the length of user profile name for search and user profile name fields are ignored.

If the limit to process name indicator field has a value of binary 1, the value contained in the process name field is compared with the corresponding value in each journal entry and only those that are equal to the value provided will be considered for selection. Only that portion of the process name that is contained within the length specified in the length of process name for search field is used for this compare. If the limit to process name indicator field contains a value of binary 1, this field must contain a value from 1 to 30. If the limit to process name indicator field has a value of binary 0, the value contained in the length of process name for search and process name fields are ignored.

If the limit to program name indicator field has a value of binary 1, the value contained in the program name field is compared with the corresponding value in each journal entry, and only those that are equal to the value provided will be considered for selection. Only that portion of the program name that is contained within the length specified in the length of program name for search field is used for this compare. If the limit to program name indicator field contains a value of binary 1, this field must contain a value from 1 to 30. If the limit to program name indicator field has a value of binary 0, the value contained in the length of program name for search and program name fields are ignored.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
  - Journal spaces indicated in the options template

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
  - Journal spaces indicated in the options template

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage			
04 System object damage	X	X	X
44 Partial system object damage	X	X	X
1A Lock State			
01 Invalid lock state	X	X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	X
02 Object destroyed	X	X	X
03 Object suspended	X	X	X
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2C Program Execution			
06 Instruction cancellation			X
07 Instruction termination			X
30 Journal Management			
04 Invalid journal space		X	
32 Scalar Specification			
01 Scalar type invalid		X	
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	
38 Template Specification			
01 Template value invalid		X	
02 Template size invalid	X		

Chapter 21. Commitment Control Instructions

This chapter describes all the instructions used for commitment control functions. These instructions are arranged in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

COMMIT (COMMIT)

Op Code (Hex)	Operand 1	Operand 2
05D2	Commit block	Commit template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: The changes to system objects, made under the control of the commit block, are made permanent and available for modification by the rest of the system. The operand 2 commit template identifies the commit description associated with the operation. Operand 1 must reference a commit block that is currently attached to the issuing process or an object ineligible for operation exception is signaled.

The commit template has the following format:

• Length of commit description	Bin(4)
• Reserved (binary 0)	Char(12)
• Commit description	Space pointer

The commit template must be aligned on a multiple of 16 bytes.

The length of commit description contains the length of the commit description field. The commit description field is preserved with the commit block until the next time a successful commit is performed against it or the commit block is destroyed or reattached to a process.

This description may be materialized through the Materialize Commit Block Attributes instruction.

Each uncommitted change referenced in the commit block is made permanent in the respective object.

Authorization Required

- Retrieve
  - Context(s) for address resolution

Lock Enforcement

- Materialize
  - Context(s) used for address resolution

Events

0002 Authorization
  0101 Authorization violation

000C Machine resource
  0201 Machine auxiliary storage threshold exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage
  0401 System object damage set
  0801 Partial system object damage set

001A Journal port
  0301 Entry not journaled
  0401 Journal space attached to a journal port became unusable

001C Journal space
  0301 Threshold reached

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
10 Damage			
04 System object damage	X	X	X
44 Partial system object damage	X		X
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
04 Object ineligible for operation	X		
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
30 Journal Management			
02 Entry not journaled			X
32 Scalar Specification			
01 Scalar type invalid	X		
38 Template Specification			
01 Template value invalid		X	

CREATE COMMIT BLOCK (CRTCB)

Op Code (Hex)	Operand 1	Operand 2
05C2	Commit block	Commit block template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction creates a commit block that is used to control changes to system objects before the system objects are committed or decommitted. The commit block template (operand 2) provides the information needed to create the commit block.

The format of the commit block template is as follows:

• Template size	Char(8)
– Number of bytes provided by user	Bin(4)*
– Number of bytes materializable	Bin(4)*
• Object identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Space attributes	Bit 1
0 = Fixed-length	
1 = Variable-length	
– Initial context	Bit 2
0 = No addressability insert	
1 = Insert addressability	
– Reserved (binary 0)	Bits 5-31
• Recovery options	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
• Reserved	Char(7)
• Context	System pointer

Note: The values of the entries annotated with an asterisk (*) are ignored by this instruction.

The commit block template must be aligned on a multiple of 16 bytes.

The commit block is owned by the user profile governing process execution. The owning user profile is implicitly assigned all authority states to the commit block. The storage occupied by the commit block is charged to this owning user profile.

The object identification specifies the symbolic name that identifies the commit block within the machine. A type code of hex 0F is implicitly supplied by the machine. The object identification is used to identify the commit block on materialize instructions as well as to locate the object in a context that addresses the object.

A space may be associated with the created commit block. The space may be fixed or variable. Initial allocation is specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated is dependent on an algorithm defined by a specific implementation. A fixed size space of zero length causes no space to be allocated.

Each byte of the space is initialized to a value specified in the initial value of space entry. When the space is extended in size, this byte value is also used to initialize the new allocation.

If the initial context creation attribute entry indicates that addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created object is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

The recovery options entry is used to specify the functions the machine is to perform in the event the object becomes damaged. This entry is implementation dependent.

The performance class parameter provides information allowing the machine to more effectively manage the object considering the overall performance objective of operations involving the commit block.

Authorization Required

- Retrieve
  - Context(s) referenced for address resolution

- Insert
  - User profile of creating process
  - Context identified by operand 2
  - Access group identified by operand 2

Lock Enforcement

- Materialize
  - Context(s) referenced for address resolution

- Modify
  - Context identified by operand 2
  - User profile of creating process
  - Access group identified by operand 2

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded
    0501 Machine address threshold exceeded

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	 2	Other
02 Access Group			
01 Object ineligible for access group		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage			
04 System object damage		X	X
1A Lock State			
01 Invalid lock state		X	
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded			
20 Machine Support			
02 Machine check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded		X	
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
01 Template value invalid		X	
02 Template size invalid		X	

DECOMMIT (DECOMMIT)

Op Code (Hex)	Operand 1
05D1	Commit block

Operand 1: System pointer.

Description: This instruction removes the changes that are specified in the commit block (operand 1) from the system, and the resources that are involved with the uncommitted changes are released. All objects under commitment control are returned to the status they had at the last established commit boundary or the status they had when they were placed under commitment if no commit boundary was established.

Operand 1 must reference a commit block that is currently attached to the issuing process; otherwise, an object ineligible for operation exception is signaled.

Authorization Required

- Retrieve
  - Context(s) referenced for address resolution

Lock Enforcement

- Materialize
  - Context(s) referenced for address resolution

Events

000C Machine resources
  0201 Machine auxiliary storage exceeded

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage
  0401 System object damage set
  0801 Partial system object damage set

001A Journal port
  0301 Entry not journaled
  0401 Journal space attached to a journal port became usable

001B Commit control
  0401 Errors during decommit

001C Journal space
  0301 Threshold reached

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Subscript range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
10 Damage		
04 System object damage		X
44 Partial system object damage		X
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
04 Object ineligible for operation	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
2E Resource Control Limit		
01 User profile storage limit exceeded		X
30 Journal Management		
02 Entry not journaled		X
32 Scalar Specification		
01 Scalar type invalid	X	
3E Commitment Control		
14 Errors during decommit	X	

DESTROY COMMIT BLOCK (DESCB)

Op Code (Hex)	Operand 1
05CD	Commit block

Operand 1: System pointer.

Description: The commit block specified by operand 1 is destroyed and addressability to the commit block is deleted from the context, if any.

If the commit block is found to be damaged during destroy processing, the commit block is destroyed.

If the commit block is attached to a process, a commit block is attached to process exception is signaled and the commit block is not destroyed.

Authorization Required

- Object control
  - Operand 1

- Retrieve
  - Context(s) referenced for address resolution

Lock Enforcement

- Materialize
  - Context(s) referenced for address resolution

- Modification
  - Context addressing object, if any
  - Access group which this object is member of, if any
  - User profile of object owner

- Object control
  - Operand 1

Events

000C Machine resources
- 0201 Machine auxiliary storage exceeded

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage
- 0401 System object damage set
- 0801 Partial system object damage set

001A Journal port
- 0301 Entry not journaled
- 0401 Journal space attached to a journal port became unusable

001C Journal space
- 0301 Threshold reached

Exceptions

	Operand	
Exception	**1**	**Other**
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Subscript range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameter		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage		
04 System object damage		X
1A Lock State		
01 Invalid lock state	X	
1C Machine-Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
30 Journal Management		
01 Entry not journaled	X	
32 Scalar Specification		
01 Scalar type invalid	X	
3E Commitment Control		
03 Commit block is attached to process	X	

MATERIALIZE COMMIT BLOCK ATTRIBUTES (MATCBATR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
05C7	Receiver template	Commit block	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar.

Description: This instruction materializes the information specified in the materialization options (operand 3) into the receiver template (operand 1). The current values in the creation template for the commit block (operand 2) may be materialized, or the current status of the commit block (operand 2) may be materialized when the commit block is not attached to another process.

The materialization options in operand 3 are as follows:

• Materialization options	Char(1)
– Reserved (binary 0)	Bits 0-4
– Suppress commit description	Bit 5
– Suppress object list	Bit 6
– Materialize option 0 = Creation template 1 = Current status	Bit 7

The attributes materialized will have the following format:

• Materialization length	Char(8)
– Number of bytes provided by user	Bin(4)
– Number of bytes that can be materialized	Bin(4)
• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object creation options	Char(4)
– Existence attributes 0 = Temp 1 = Perm	Bit 0
– Space attributes 0 = Fixed-length 1 = Variable-length	Bit 1
– Initial context 0 = No addressability insert 1 = Insert addressability	Bit 2
– Access group 0 = Do not create as access group member 1 = Create as access group member	Bit 3
– Replace option 0 = Create as new object 1 = Replace currently existing object	Bit 4
– Reserved (binary 0)	Bits 5-31
• Recovery options	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
• Reserved	Char(7)
• Context	System pointer
• Access group	System pointer

The target area must be 16-byte aligned. The first 8 bytes of the materialization output in all the materialization options identify the total number of bytes provided by the user for the materialization and the total number of bytes available to be materialized. If fewer than 8 bytes are available in the byte area identified by the receiver, operand 1, a materialization length exception is signaled. The instruction materializes as many bytes as can be contained in the receiver's byte area. If the byte area of the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions are signaled in the event the receiver contains insufficient space for the materialization, other than the materialization length exception described previously.

The values are defined in the Create Commit Block instruction. These values were defined at the creation of the commit block with the following possible exceptions. The object identification, context, initial context, and size of associated space will contain current values.

If operand 3 specified that the status should be materialized, the receiver will be in the following format:

• Materialization length	Char(8)
– Number of bytes provided by user	Bin(4)
– Number of bytes that can be materialized	Bin(4)
• Commit block status	Char(2)
– Attached indicator 0 = Block not attached to a process 1 = Block attached to a process	Bit 0
– Uncommitted changes indicator 0 = A commit cycle has not been started for this commit block 1 = A commit cycle has been started for this commit block	Bit 1
– Reserved (binary 0)	Bits 3-15
• Number of uncommitted changes	Bin(4)
• Length of commit description	Bin(4)
• Number of objects in associated object list	Bin(4)
• Journal entry sequence number	Bin(4)
• Reserved	Char(6)
• Commit description	Space pointer
• Process control space	System pointer
• Object associated with the commit block (0 to n pointers)	System pointer

If the operand 2 commit block is attached to a process other than the issuing process and both the commit object list and the commit description are not suppressed, an object not eligible for operation exception is signaled.

The materialization length has the same definition as the hex 00 option.

The commit block status reflects the usage of the commit block at the time the materialize was issued.

An attached indicator value of binary 0 indicates that the commit block is not presently attached to a process. A value of binary 1 in this field indicates the block is attached to a process and a pointer to the process control space of that process is returned in the process control space field.

If the uncommitted changes indicator contains a value of binary 1, the referenced commit block has started a commit cycle. A value of binary 0, indicates the commit block has not started a commit cycle. If this indicator has a value of binary 1, then either a commit or a decommit must be executed before the commit block can be detached from the indicated process even if the number of uncommitted changes is equal to 0.

The number of uncommitted entries is a count of the number of changes that have been made to objects associated with the commit block since the last commit. This count is cumulative for all objects associated with the block.

The length of commit description returns the number of bytes in the commit description. If the length of commit description is 0, then either no commit description was provided on the last commit that was executed against this commit block or no successful commit has been executed since it was last attached to a process. The commit description is returned in the operand 1 receiver area and a space pointer to it is provided.

The list of objects associated with the commit block is an array of system pointers; each pointer references a system object whose changes are to be controlled through the commit block, if the commit block is active to the process making the change. The number of objects in associated object list gives the length of this array.

The journal entry sequence number contains the journal entry sequence number of the start commit journal entry for the last commit cycle since the attach of the commit block. If no commit cycle has been started since the commit block was last attached to a process, this field contains a value of zero.

The process control space field contains a system pointer to the process control space for the process to which the commit block is currently attached if the attached indicator contains a value of binary 1. If this indicator contains a value of binary 0, the process control space field contains binary zeros.

If the suppress object list option has a value of binary 1, the number objects in associated object list and the system pointer to the objects associated with the commit block contain binary zeros.

If the suppress commit description option has a value of binary 1, the length of commit description and the system pointer to the commit description contain binary zeros.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Operational
  - Operand 2

- Retrieve
  - Context(s) referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Context(s) referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

0010 Process
    0601 Exception signaled to process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference
    0201 Object location reference

0017 Damage
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands 1	 2	 3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X		
03 Subscript range	X	X	X	
04 String range			X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation		X		
10 Damage				
04 System object damage		X		
1A Lock State				
01 Invalid lock state		X		
1C Machine-Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
04 Object not eligible for operation		X		
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object		X		
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0A Invalid operand length	X		X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	X			

MODIFY COMMIT BLOCK (MODCB)

Op Code (Hex)	Operand 1	Operand 2
05C6	Commit block	Commit block modification template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: This instruction modifies the commit block (operand 1) as is specified in the commit block modification template (operand 2).

The commit block modification template has the following format:

• Modification options	Char(2)
– Attach commit block	Bit 0
– Detach commit block	Bit 1
– Add objects to commitment control	Bit 2
– Remove objects from commitment control	Bit 3
– Remove all objects from commitment control	Bit 4
– Reserved	Bits 5-15
• Number of objects in object list	Bin(2)
• Reserved (binary 0)	Char(12)
• Object list (0 to n pointers)	System pointer

The commit block modification template must be aligned on a multiple of 16 bytes.

If a bit in the modification options is binary 1, then the indicated modification is specified.

If attach commit block is specified in the modification options, the commit block specified by operand 1 is checked for attach status. If the block is attached to another process, an object not eligible for operation exception is signaled. If the block is already attached to the issuing process, an invalid commit block status change exception is signaled. If the commit block is not being journaled, a commit block not journaled exception is signaled. The block is attached to the issuing process if these conditions are met.

If detach commit block is specified in the modification options, the commit block specified in operand 1 is checked for its current status. The commit block must be attached to the issuing process. If the commit block is not attached to the issuing process, an object not eligible for operation exception is signaled. If there are objects under commitment control for this commit block, an objects under commitment control exception is signaled. If any changes are found that are associated with the commit block, a commit block contains uncommitted changes exception is signaled. If both the attach commit block and detach commit block values are binary 1, a template value invalid exception is signaled. If all of these conditions are met, the commit block is detached from the issuing process.

If add objects to commitment control, remove objects from commitment control, or remove all objects from commitment control is requested in the modification options, it must be the only option requested. If not, a template value invalid exception is signaled. If neither add objects nor remove objects is requested, the number of objects in object list and the object list are ignored.

If add objects to commitment control is requested in the modification options, a check of the commit block is made to determine its attached status. If the block is not attached to the issuing process, an object ineligible for operation exception is signaled. If the number of objects in object list value is not greater than zero, a template value invalid exception is signaled. If any of the objects in the object list are already under commitment control, then an object ineligible for commitment control exception is signaled. If these conditions are met, the objects are placed under commitment control of the specified commitment control.

If remove objects from commitment control is requested in the modification options, a check of the commit block is made to determine its attached status. If the block is not attached to the issuing process, an object ineligible for operation exception is signaled. If the number of objects in object list is not greater than zero or is greater than the limit for the number of objects under commitment control, a template value invalid exception is signaled. If any of the objects in the object list are not under commitment control to the specified commit block, an object ineligible for removal from commitment control exception is signaled. If these conditions are met, the objects are removed from commitment control.

If remove all objects from commitment control is requested in the modification options, a check of the commit block is made to determine its attached status. If the block is not attached to the issuing process, an object ineligible for operation exception is signaled. If these conditions are met, all objects currently under commitment control of the specified commit block are removed from commitment control.

Authorization Required

- Operational
  - Operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution

- Modification
  - Operand 1

- Implicit locks
  - The operand 1 commit block is implicitly locked LEAR, it it is being attached to the issuing process.
  - The implicit LEAR lock on the operand 1 commit block is removed if the commit block is being detached from the issuing process.

Events

0002 Authorization
- 0101 Authorization violation

000C Machine resources
- 0201 Machine auxiliary storage exceeded

0010 Process
- 0601 Exception signaled to process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference
- 0201 Object location reference

0017 Damage
- 0401 System object damage set
- 0801 Partial system object damage set

001A Journal port
- 0301 Entry not journaled
- 0401 Journal space attached to a journal port became unusable

001C Journal space
- 0301 Threshold reached

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X	X	
10 Damage			
04 System object damage	X	X	X
44 Partial system object damage	X		
1A Lock State			
01 Invalid lock state	X		
1C Machine-Dependent Exception			
03 Machine storage limit exceeded			X

Exceptions

	Operands		
Exception	**1**	**2**	**Other**
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
04 Object ineligible for operation	X		
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X	X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X	X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
30 Journal Management			
02 Entry not journaled			X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid		X	
03 Scalar value invalid		X	
38 Template Specification			
01 Template value invalid		X	
02 Template size invalid		X	
3E Commitment Control			
01 Invalid commit block status change	X		
04 Commit block controls uncommitted changes	X		
06 Commit control resource limit exceeded	X		X
08 Object under commitment control being incorrectly journaled		X	
11 Process has attached commit block			X
12 Objects under commitment control			X
13 Commit block not journaled	X		
15 Object ineligible for commitment control		X	
16 Object ineligible for removal from commitment control		X	X

Chapter 22. Dump Space Management Instructions

This chapter describes all the instructions used for dump space management. These instructions are arranged in alphabetical order. For an alphabetic summary of all the instructions, see Appendix A. *Instruction Summary*.

CREATE DUMP SPACE (CRTDMPS)

Op Code (Hex)	Operand 1	Operand 2
04D2	Dump space	Dump space template

Operand 1: Dump space.

Operand 2: Dump space template.

Description: A dump space is created with the attributes provided in the dump space creation template specified by operand 2. Addressability to the created dump space is placed in the system pointer that is returned in the addressing object specified by operand 1.

A dump space provides a storage area within the machine for a dump of system objects. In addition to the operations provided by the dump space management instructions, the Request Path Operation instruction provides support for load or dump operations utilizing a dump space.

The dump space template specified by operand 2 must be 16-byte aligned in the space. The dump space template has the following format:

• Template Size Specification	Char(8)*
– Size of template	Bin(4)*
– Number of bytes available for materialization	Bin(4)*
• Object Identification	Char(32)
– Object type	Char(1)*
– Object subtype	Char(1)
– Object name	Char(30)
• Object Creation Options	Char(4)
– Existence attribute	Bit 0
0 = Temporary	
1 = Permanent	
– Space attribute	Bit 1
0 = Fixed length	
1 = Variable length	
– Initial context	Bit 2
0 = Do not insert addressability into context	
1 = Insert addressability into context	
– Reserved (binary 0)	Bit 3-31
• Recovery Options	Char(4)
• Size of Space	Bin(4)
• Initial Value of Space	Char(1)
• Performance Class	Char(4)
• Reserved (binary 0)	Char(7)
• Context	System pointer
• Reserved	Char(16)
• Dump Space Size	Char(4)
• Dump Data Size	Char(4)*
• Dump Data Size Limit	Char(4)
• Reserved (binary 0)	Char(20)

Note: The values associated with template entries annotated with an asterisk (*) are ignored by the instruction.

The created dump space is owned by the user profile governing process execution. The owning user profile is implicitly assigned all authority states for the object. The storage occupied by the created dump space is charged to this owning user profile.

The object identification specifies the symbolic name that identifies the dump space within the machine. A type code of hex 13 is implicitly supplied by the machine. The object identification is used to identify the object on materialize instructions as well as to locate the object in a context that addresses the object.

The existence attribute specifies whether the dump space is to be created as temporary or permanent. A temporary dump space, if not explicitly destroyed by the user, is implicitly destroyed by the machine when machine processing is terminated. A permanent dump space exists in the machine until explicitly destroyed by the user.

The space attribute specifies whether the space associated with the created dump space is of fixed or variable size. The initial allocation is as specified in the size of space entry. The machine allocates a space of at least the size specified. The actual size allocated is dependent on the algorithm used within the specific implementation of the machine. A fixed space of zero length causes no space to be allocated.

Each byte of the space is initialized to a value specified by the initial value of space entry. When the space is extended in size, this byte value is also used to initialize the new allocation.

If the initial context creation attribute entry indicates the addressability is to be inserted into a context, the context entry must contain a system pointer that identifies a context where addressability to the newly created space is to be placed. If addressability is not to be inserted into a context, the context entry is ignored.

The recovery options entry is used to specify the functions the machine is to perform in the event the dump space becomes damaged in some manner. This entry is implementation dependent.

The performance class entry provides information allowing the machine to more effectively manage the dump space considering overall performance objectives of operations involving the dump space.

The dump space size entry specifies the initial allocation size for the dump space. This value specifies the number of 512 byte blocks of the space to be usable for storage of dump data within the dump space. The machine allocates a dump space of at least the size specified. The actual size allocated is dependent on an algorithm defined by the specific implementation of the machine.

The dump data size limit entry specifies the limit for the number of 512-byte blocks of dump data which may be stored into the dump space. A size value of zero specifies that there is no explicit limit on the amount of dump data which can be stored in the dump space. The machine implicitly places a limit on the maximum size of a dump space. This value of this limitation is dependent upon the specific implementation of the machine.

An attempt to insert dump data through the Insert Dump Data instruction from a dump of a size in excess of this limit results in the signaling of the dump data size limit exceeded exception. An attempt to store dump data in excess of this limit through a Request Path Operation instruction dump operation results in return of a feedback record specifying that the dump data size limit has been exceeded. This size value may be larger than the value of the dump space size field. The dump data size limit field can be used to control the size to which a dump space will be extended by the machine either explicitly through a Modify Dump Space instruction or implicitly through an Insert Dump Data instruction or a Request Path Operation instruction dump operation.

Authorization

- Insert
  - User profile of creating process
  - Context identified in operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Context identified in operand 2
  - User profile of creating process

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded
    0501 Machine address threshold exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands				
Exception	**1**	**2**	**3**	**4**	**Other**
06 Addressing					
01 Space addressing violation	X	X			
02 Boundary alignment	X	X			
03 Range	X	X			
06 Optimized addressability invalid	X	X			
08 Argument/Parameter					
01 Parameter reference violation	X	X			
0A Authorization					
01 Unauthorized for operation		X			
0E Context Operation					
01 Duplicate object identification		X			
10 Damage Encountered					
04 System object damage		X			X
44 Partial system object damage					X
1A Lock State					
01 Invalid lock state		X			
1C Machine Dependent Exception					
03 Machine storage limit exceeded					X
04 Object storage limit exceeded		X			
20 Machine Support					
02 Machine check					X
04 Function check					X
22 Object Access					
01 Object not found	X	X			
02 Object destroyed	X	X			
03 Object suspended	X	X			
24 Pointer Specification					
01 Pointer does not exist	X	X			
02 Pointer type invalid	X	X			
03 Pointer addressing invalid object		X			
2A Program Creation					
06 Invalid operand type	X	X			
07 Invalid operand attribute	X	X			
08 Invalid operand value range	X	X			
0C Invalid operand ODT reference	X	X			
0D Reserved bits are not zero	X	X			X
2E Resource Control Limit					
01 User profile storage limit exceeded		X			
32 Scalar Specification					
01 Scalar type invalid	X	X			
38 Template Specification					
01 Template value invalid		X			

DESTROY DUMP SPACE (DESDMPS)

Op Code (Hex)	**Operand 1**
04D1	Dump space to be destroyed

Operand 1: System pointer.

Description: The designated dump space is destroyed and addressability to the dump space is deleted from a context if one is currently addressing the object. The pointer identified by operand 1 is not modified by the instruction and a subsequent reference to the pointer causes an object destroyed exception.

If the dump space is currently in use by a Request Path Operation instruction, load or dump function, or Request I/O instruction load or dump function, the object not eligible for destruction exception is signaled and the dump space is not destroyed.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

- Object control
  - Operand 1

Lock Enforcement

- Modification
  - User profile owning object
  - Context addressing object

- Object control
  - Operand 1

Events

0002 Authorization
  0101 Authorization violation

000C Machine resources
  0201 Machine auxiliary storage exceeded

000D Machine status
  0101 Machine check

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
02 Boundary alignment	X	
03 Range	X	
06 Optimized addressability invalid	X	
08 Argument/Parameters		
01 Parameter reference violation	X	
0A Authorization		
01 Unauthorized for operation	X	
10 Damage Encountered		
04 System object damage	X	X
44 Partial system object damage		X
1A Lock State		
01 Invalid lock state	X	
1C Machine Dependent Exception		
03 Machine storage limit exceeded		X
20 Machine Support		
02 Machine check		X
03 Function check		X
22 Object Access		
01 Object not found	X	
02 Object destroyed	X	
03 Object suspended	X	
06 Object not eligible for destruction	X	
24 Pointer Specification		
01 Pointer does not exist	X	
02 Pointer type invalid	X	
03 Pointer addressing invalid object	X	
2A Program Creation		
06 Invalid operand type	X	
07 Invalid operand attribute	X	
08 Invalid operand value range	X	
0C Invalid operand ODT reference	X	
0D Reserved bits are not zero	X	X
32 Scalar Specification		
01 Scalar type invalid	X	

INSERT DUMP DATA (INSDMPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04D3	Dump Space	Controls	Source

Operand 1: System pointer.

Operand 2: Character(16) scalar.

Operand 3: Space pointer.

Description: The instruction inserts the blocks of dump data specified by operand 2 and operand 3 into the dump space specified for operand 1.

The dump data inserted into the operand 1 dump space is accessed from the byte string addressed by the operand 3 space pointer for the number of blocks specified in the controls operand block count field. The dump data in the source is appended to the dump data, if any, currently stored in the operand 1 dump space.

The blocks of data contained in the source are assumed to have been previously retrieved from a dump space through use of the Retrieve Dump Data instruction. Only the length attribute of the source data is defined. Each block of data in the source is 528 bytes long. A source block contains a 512-byte block of dump data to be inserted into the operand 1 dump space and 16 bytes of machine control information related to the block. The internal format of the source data is not defined other than for the length attribute just described.

The 512 byte blocks of dump data to be inserted into the receiver are each appended in sequence to the dump data contained in the operand 1 dump space. The 16 bytes of machine control information contained in each 528-byte source block provides for verifications to insure that the insertion of the 512 bytes of dump data is valid for the receiver dump space. Detection of an error by these verifications results in the signaling of the invalid dump data insertion exception. The verifications are performed on a block-by-block basis so that the exception may be detected at any point during the insertion of the source data into the operand 1 dump space.

The controls operand must be a character scalar which specifies control information for the dump data to be inserted. It must be at least 16 bytes long and have the following format:

• Controls	Char(16)
– Reserved (binary 0)	Char(4)
– Block count	Char(4)
– Reserved (binary 0)	Char(8)

The block count field specifies the number of blocks of dump data to be inserted. A value of zero is invalid and results in the signaling of the template invalid exception.

Insertion of dump data into the dump space at a point beyond the current dump space size results in an implicit extension of the dump space to a size adequate for the data to be inserted. The storage extension is charged to the user profile which owns the dump space.

Insertion of dump data into the dump space for a dump of a size greater than that allowed by the dump data size limit attribute of the dump space results in the signaling of the dump data size limit exceeded exception. This exception is detected on the initial insertion of dump data for the dump and prior to insertion of any of the dump data from the source into the operand 1 dump space.

The dump space specified by operand 1 must not be in use by a Request Path Operation load or dump function or a Request I/O load or dump function when this instruction is executed or the object not available to process exception is signaled.

Substring operand references which allow for a null substring reference (a length value of zero) may not be specified for this instruction.

The insertion of dump data into the receiver dump space is performed in a manner that insures the value of the dump data size attribute will correctly reflect the amount of valid dump data contained in the dump space even in the event of a system failure during the insertion of dump data. In this event, the Materialize Dump Space instruction can be used to determine the dump data size attribute value upon subsequent availability of machine processing.

It is anticipated that the normal usage of this instruction will be to insert all of the dump data for a dump which was retrieved from a source dump space. Due to the large size of the dump data which can be contained in a dump space, the insertion of dump data will probably be performed by placing portions of the dump data into a buffer and then inserting the buffer of data into the target dump space. Partial insertion of a dump into a dump space allows for subsequent loading of only the objects completely contained within the inserted dump data. Because object boundaries within the dump data are not defined, the point at which a partial dump would allow for the loading of a particular object can not be determined by the user of the machine. The point at which a partial dump ends may occur at any point within the dump data for an object causing the data for the object to be only partially contained in the dump space. Attempting to load an object partially contained in a dump space may result in it not being loaded or may result in it being loaded and marked as damaged.

For a load operation to correctly load the objects contained in a dump space, the dump data retrieved from a dump space must not be modified and must be inserted into the exact same offset in the target dump space as that from which it was retrieved. Violation of these requirements may or may not be detected on the load operation. However, subsequent usage of an object which could not be correctly loaded may produce erroneous results, result in termination of the process accessing the object or even possibly result in termination of machine processing entirely.

Authorization Required

- Insert
  - Dump space
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Dump space

Events

0002 Authorization
- 0101 Authorization violation

000C Machine resources
- 0201 Machine auxiliary storage exceeded
- 0501 Machine address threshold exceeded

000D Machine status
- 0101 Machine check

0010 Process
- 0701 Maximum processor time exceeded
- 0801 Process storage limit exceeded

0016 Machine observation
- 0101 Instruction reference

0017 Damage set
- 0401 System object damage set
- 0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation	X			
10 Damage Encountered				
04 System object damage	X			X
44 Partial system object damage	X			X
1A Lock State				
01 Invalid lock state	X			
1C Machine Dependent Exception				
03 Machine storage limit exceeded				X
04 Object storage limit exceeded	X			
20 Machine Support				
02 Machine check			X	
03 Function check			X	
22 Object Access				
01 Object not found	X	X	X	
03 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
05 Object not available to process	X			
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object	X			
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
2E Resource Control Limit				
01 User profile storage limit exceeded				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		
40 Dump Space Management				
01 Dump data size limit exceeded	X			
02 Invalid dump data insertion			X	

MATERIALIZE DUMP SPACE (MATDMPS)

Op Code (Hex)	Operand 1	Operand 2
04DA	Receiver	Dump space

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The current attributes of the dump space specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less then eight causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The template identified by operand 1 must be 16-byte aligned in the space. The format of the materialization is as follows:

• Materialization Size Specification	Char(8)
– Number of bytes provided for materialization	Bin(4)
– Number of bytes available for materialization (always 128 for this instruction)	Bin(4)
• Object Identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
• Object Creation Options	Char(4)
– Existence attributes 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed length 1 = Variable length	Bit 1
– Context 0 = Addressability not in context 1 = Addressability in context	Bit 2
– Reserved (binary 0)	Bit 3-31
• Recovery Options	Char(4)
• Size of Space	Bin(4)
• Initial Value of Space	Char(1)
• Performance Class	Char(4)
• Reserved	Char(7)
• Context	System pointer
• Reserved	Char(16)
• Dump Space Size	Char(4)
• Dump Data Size	Char(4)
• Dump Data Size Limit	Char(4)
• Reserved	Char(20)

The dump space size entry is set with the current size value for the number of 512-byte blocks of space allocated for storage of dump data within the dump space.

The dump data size entry is set with the current size value for the number of 512-byte blocks of dump data contained in the dump space. This value specifies the number of blocks from the start of the dump space through the block of dump data which has been placed into the dump space at the largest dump space offset value. A value of zero indicates that the dump space currently contains no dump data.

The dump space size and dump data size values are implicitly updated by the Insert Dump Data instruction and Request Path Operation instruction dump function and can be explicitly reset by the Modify Dump Space instruction. Additionally, the dump data size value is used by the Retrieve Dump Data instruction, and Request Path Operation instruction load function as the amount of dump data available in the dump space.

The dump data size limit entry is set with the current size limit for the number of 512-byte blocks of dump data which may be stored in the dump space. A value of zero indicates that no explicit limitation is placed on the amount of dump data which may be stored in the dump space. The machine implicitly places a limit on the maximum size of a dump space. This value of this limitation is dependent upon the specific implementation of the machine.

The dump data size limit is used by the Insert Dump Data instruction and Request Path Operation instruction dump function as the limit for the amount of dump data which can be stored in a dump space. It can be modified by the Modify Dump Space instruction.

Authorization Required

- Operational
  - Operand 2

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment violation	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	
10 Damage Encountered			
04 System object damage		X	X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state		X	
1C Machine Dependent Exception			
03 Machine storage limit exceeded			X
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object		X	
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length	X		
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
32 Scalar Specification			
01 Scalar type invalid	X	X	
38 Template Specification			
01 Template value invalid	X		

MODIFY DUMP SPACE (MODDMPS)

Op Code (Hex)	Operand 1	Operand 2
04D6	Dump space	Modification template

Operand 1: System pointer.

Operand 2: Character(32) scalar.

Description: This instruction modifies the attributes of the dump space specified by operand 1 to have the attribute values specified in operand 2.

The operand 2 modification template must be a character scalar which specifies the modifications to be performed. It must be at least 32 bytes long and have the following format:

• Modification	Char(4)
– Reset dump data size 0 = No 1 = Yes	Bit 0
– Reset dump space size 0 = No 1 = Yes	Bit 1
– Modify dump data size limit 0 = No 1 = Yes	Bit 2
– Reserved (binary 0)	Bit 3-31
• Dump Space Size	Char(4)*
• Dump Data Size	Char(4)*
• Dump Data Size Limit	Char(4)
• Reserved (binary 0)	Char(16)

Note: The values associated with template entries annotated with an asterisk (*) are ignored by the instruction.

The modification indicators select the modifications to be performed on the dump space.

The reset dump data size modification indicator controls whether or not the dump data size attribute for the dump space is to be reset. When yes is specified, the dump data size value is reset to zero. When no is specified, the dump data size value is not reset.

The reset dump space size modification indicator controls whether or not the dump space size attribute is to be reset back to the dump data size attribute value. When yes is specified, the current allocation of the dump space is decreased as closely as possible toward the current value of the dump data size attribute. The actual size of the decreased allocation is dependent upon the algorithm used within the specific implementation of the machine. When no is specified, the dump space size value is not reset.

The modify dump data size limit modification indicator controls whether or not the dump data size limit for the dump space is to be modified. When yes is specified, the dump data size limit attribute for the dump space is set to the value specified in the controls operand dump data size limit field. This value specifies the size limit for the number of 512-byte blocks of dump data which may be stored in a dump space. A value of zero indicates that no explicit limitation is placed on the amount of dump data which may be stored in the dump space. The machine implicitly places a limit on the maximum size of a dump space. This value of this limitation is dependent upon the specific implementation of the machine. A value less than the current value of the dump data space size attribute value for the dump space is invalid and results in the signaling of the invalid dump space modification exception. When no is specified, the dump data size limit attribute for the dump space is not modified and the dump data size limit field is ignored.

A reset of the dump data size attribute will be performed first when requested in conjunction with either a reset of the dump space size or a modify of the dump data size limit causing the current value of the attribute to be zero when those functions are performed.

Refer to the description of the dump space size, dump data size, and dump data size limit attributes under the Materialize Dump Space instruction for additional information on their meaning.

For any modification request other than an increase in the value of the dump data size limit attribute, the dump space specified by operand 1 must not be in use by a Request Path Operation load or dump function or a Request I/O load or dump function when this instruction is executed or the object not available to process exception is signaled.

Partial object damage for a dump space can be removed through a reset of the dump space size when the dump data size is zero. The dump data size can either already be zero or can be requested through an accompanying reset of the dump data size attribute.

Substring operand references which allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Object Management
  - Operand 1

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Modify
  - Operand 1

- Materialization
  - Contexts referenced for address resolution

Events

0002 Authorization
  0101 Authorization violation

000C Machine resources
  0201 Machine auxiliary storage exceeded
  0501 Machine address threshold exceeded

000D Machine status
  0101 Machine check

0010 Process
  0701 Maximum processor time exceeded
  0801 Process storage limit exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object damage set
  0801 Partial system object damage set

Exceptions

Exception	Operands		Other
	1	2	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment violation	X	X	
03 Range	X	X	
06 Optimized addressability invalid	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation	X		
10 Damage Encountered			
04 System object damage	X		X
44 Partial system object damage			X
1A Lock State			
01 Invalid lock state	X		
1C Machine Dependent Exception			
03 Machine storage limit exceeded			X
04 Object storage limit exceeded	X		
20 Machine Support			
02 Machine check			X
03 Function check			X
22 Object Access			
01 Object not found	X	X	
02 Object destroyed	X	X	
03 Object suspended	X	X	
04 Object suspended	X		
24 Pointer Specification			
01 Pointer does not exist	X	X	
02 Pointer type invalid	X	X	
03 Pointer addressing invalid object	X		
2A Program Creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value range	X	X	
0A Invalid operand length		X	
0C Invalid operand ODT reference	X	X	
0D Reserved bits are not zero	X	X	X
2E Resource Control Limit			
01 User profile storage limit exceeded			X
32 Scalar Specification			
01 Scalar type invalid	X	X	
02 Scalar attributes invalid		X	
38 Template Specification			
01 Template value invalid		X	
40 Dump Space Management			
03 Invalid dump space modification		X	

RETRIEVE DUMP DATA (RETDMPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04D7	Receiver	Controls	Dump space

Operand 1: Space pointer.

Operand 2: Character(16) variable scalar.

Operand 3: System pointer.

Description: The instruction retrieves the number of 512-byte blocks of dump data selected by operand 2 from the dump space specified for operand 3 and places it along with associated control information in the space object specified by operand 1.

The dump data placed into the receiver for the retrieve operation is set into the byte string addressed by the operand 1 space pointer for the number of blocks returned in the controls operand block count retrieved field.

Only the length attribute of the retrieved dump data is defined. Each block of data placed in the receiver is 528 bytes long. The receiver block contains the requested 512-byte block of dump data from the operand 3 dump space and 16 bytes of machine control information related to the block. The machine control information provides for verifications on the data upon its subsequent insertion into a target dump space. The internal format of the retrieved dump data is not defined other than that subsequent insertion of it into a dump space can be done to prime a dump space prior to performing a Request Path Operation instruction load function on the dump space.

The number of blocks of dump data retrieved is the lessor of that requested and that available at the specified block location in the dump space. The number of blocks available is calculated by subtracting the number of the first block requested from the current value of the dump data size attribute for the dump space and adding one to the difference. If blocks of dump data are available, the number is greater than zero, the instruction results in the retrieval of dump data. The blocks retrieved field is set to the lessor of the calculated blocks available and the blocks requested values. If the request is invalid, the number is less than or equal to zero, the instruction results in the signaling of the invalid dump data retrieval exception.

The controls operand must be a character scalar which specifies selection information for the dump data to be retrieved. It must be at least 16 bytes long and have the following format:

• Controls	Char(16)
– First block requested	Char(4)
– Block count requested	Char(4)
– Block count retrieved	Char(4)
– Reserved	Char(4)

The first block requested field specifies the number of the first block of dump data which is to be retrieved from the operand 3 dump space. A value of one identifies the first block of dump data contained in the dump space. A value of zero is invalid and results in the signaling of the template value invalid exception.

The block count requested field specifies the number of blocks of dump data to be retrieved. A value of zero is invalid and results in the signaling of the template value invalid exception.

The block count retrieved field is set, as described above, by the instruction. This field is continually updated during the execution of the instruction to keep count of the number of blocks of data which may have been retrieved. Due to this, it may also be modified from its original value when the instruction results in the signaling of an exception.

The dump space specified by operand 3 must not be in use by a Request Path Operation load or dump function or a Request I/O load or dump function when this instruction is executed or the object no available to process exception is signaled.

Substring operand references which allow for a null substring reference (a length value of zero) may not be specified for this instruction.

It is anticipated that the normal usage of this instruction will be to retrieve all of the dump data contained in the source dump space. Due to the large size of the dump data which can be contained in a dump space, the retrieval of dump data will probably be performed by sequentially retrieving portions of the dump data into a buffer and then sending the buffer of data on to another system or some other storage media. Subsequent insertion of this dump data into a target dump space would then be done prior to loading the objects contained in the dump back into existence on the machine. Partial retrieval of a dump from a dump space allows for subsequent loading of only the objects completely contained within the retrieved dump data. Because object boundaries within the dump data are not defined, the point at which a partial dump would allow for the loading of a particular object can not be determined by the user of the machine. The point at which a partial dump ends may occur at any point within the dump data for an object causing the data for the object to be only partially contained in the dump space. Attempting to load an object partially contained in a dump space may result in it not being loaded or may result in it being loaded and marked as damaged.

For a load operation to correctly load the objects contained in a dump, the dump data retrieved from a dump space must not be modified and must be inserted into the exact same relative location in the target dump space as that from which it was retrieved. Violation of these requirements may or may not be detected on the load operation. However, subsequent usage of an object which could not be correctly loaded may produce erroneous results, result in termination of the process accessing the object or even possible result in termination of machine processing entirely.

Authorization Required

- Retrieve
  - Dump space
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Events

0002 Authorization
    0101 Authorization violation

000C Machine resources
    0201 Machine auxiliary storage exceeded

000D Machine status
    0101 Machine check

0010 Process
    0701 Maximum processor time exceeded
    0801 Process storage limit exceeded

0016 Machine observation
    0101 Instruction reference

0017 Damage set
    0401 System object damage set
    0801 Partial system object damage set

Exceptions

	Operands			
Exception	**1**	**2**	**3**	**Other**
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment violation	X	X	X	
03 Range	X	X	X	
06 Optimized addressability invalid	X	X	X	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0A Authorization				
01 Unauthorized for operation			X	
10 Damage Encountered				
04 System object damage			X	X
44 Partial system object damage			X	X
1A Lock State				
01 Invalid lock state			X	
1C Machine Dependent Exception				
03 Machine storage limit exceeded				X
20 Machine Support				
02 Machine check				X
03 Function check				X
22 Object Access				
01 Object not found	X	X	X	
02 Object destroyed	X	X	X	
03 Object suspended	X	X	X	
05 Object not available to process			X	
24 Pointer Specification				
01 Pointer does not exist	X	X	X	
02 Pointer type invalid	X	X	X	
03 Pointer addressing invalid object			X	
2A Program Creation				
06 Invalid operand type	X	X	X	
07 Invalid operand attribute	X	X	X	
08 Invalid operand value range	X	X	X	
0C Invalid operand ODT reference	X	X	X	
0D Reserved bits are not zero	X	X	X	X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
38 Template Specification				
01 Template value invalid		X		
40 Dump Space Management				
03 Invalid dump space modification		X		

Chapter 23. Exception Specifications

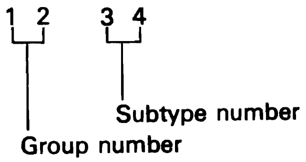
Exception generation is the only facility for synchronously communicating error conditions that are a direct result of System/38 instruction processing. Machine exceptions identify error conditions that require processing before the next sequential System/38 instruction is executed. Instructions that cause a particular exception may not function identically before execution is stopped; however, each instruction produces consistent results. These results ensure machine integrity and reliability. The results are inherent in a particular exception definition or in the detailed instruction definition.

The user can monitor any number of exceptions. There are three basic techniques for the user to handle an exception. One technique is to provide detailed handling specified by a program defined exception description object. The second technique is to provide a default exception handler for the process. This exception handler is invoked whenever an invocation fails to handle an exception. The third technique is to accept the machine default of process termination by not providing an appropriate exception handling mechanism. See *Exception Management* in the *Functional Concepts Manual*, for a general description of exception management.

MACHINE INTERFACE EXCEPTION DATA

Exception data is communicated across the machine interface through a Retrieve Exception Data instruction. Certain information is available for all exceptions when an appropriate exception description has been defined by the user. That information includes the following:

- Exception identification–This is a 2-byte hexadecimal field formed by concatenating to the high-order 1-byte exception group number a low-order 1-byte exception subtype number. The format of the exception identification is as follows:



- Compare value length
- Compare value (machines signaled have a compare value of hex 00000000 with a length of 4)
- Exception-specific data
- Signaling program invocation address
- Signaled program invocation address
- Signaling program instruction address
- Signaled program instruction address
- Machine-dependent data identifying the component that generated the exception

The exception-specific data provides additional pointers and data that may be required for an individual exception.

EXCEPTION LIST

The following is a list of all exceptions in alphabetic and numeric order by group. The subtypes within each group are in numeric order.

02 Access Group

- 01 Object ineligible for access group

04 Access State

- 01 Access state specification invalid

06 Addressing

- 01 Space addressing violation
- 02 Boundary alignment
- 03 Range
- 04 External data object not found
- 05 Invalid space reference
- 06 Optimized addressability invalid

08 Argument/Parameter

- 01 Parameter reference violation
- 02 Argument list length violation
- 03 Argument list length modification violation

0A Authorization

- 01 Unauthorized for operation
- 02 Privileged instruction
- 03 Attempt to grant/retract authority state to an object that is not authorized
- 04 Special authorization required
- 05 Create/modify user profile beyond level of authorization

0C Computation

- 01 Conversion
- 02 Decimal data
- 03 Decimal point alignment
- 04 Edit digit count
- 05 Edit mask syntax
- 06 Floating-point overflow
- 07 Floating-point underflow
- 08 Length conformance
- 09 Floating-point invalid operand
- 0A Size
- 0B Zero divide
- 0C Invalid floating-point conversion
- 0D Floating-point inexact result
- 0E Floating-point zero divide
- 0F Master key not defined
- 10 Weak key not valid
- 11 Key parity invalid

0E Context Operation

- 01 Duplicate object identification
- 02 Object ineligible for context

10 Damage Encountered

- 02 Machine context damage state
- 04 System object damage state
- 44 Partial system object damage state

12 Data Base Management

01	Conversion mapping error
02	Key mapping error
03	Cursor not set
04	Data space entry limit exceeded
05	Data space entry already locked
06	Data space entry not found
07	Data space index invalid
08	Incomplete key description
09	Duplicate key value in existing data space entry
0A	End of path
0B	Duplicate key value detected while building unique data space index
0D	No entries locked
0F	Duplicate key value in uncommitted data space entry
13	Invalid mapping template
14	Invalid selection template
15	Data space not addressed by index
16	Data space not addressed by cursor
17	Key changed since set cursor
19	Invalid rule option
1A	Data space entry size exceeded
1B	Logical space entry size limit exceeded
1C	Key size limit exceeded
1D	Logical key size limit exceeded
1E	Selection routine buffer size limit exceeded
1F	User exit routine criteria not satisfied
21	Unable to maintain a unique key data space index
22	Data space index with user exit selection routine build termination
23	Data space index user exit selection routine failure
25	Invalid data base operation
26	Data space index with invalid floating-point field build termination
27	Data space index key with invalid floating-point field
30	Specified data space entry rejected
32	Join value changed
33	Data space index with non-user exit selection routine build termination
34	Non-user exit selection routine failure
36	No mapping code specified
37	Operation not valid with join cursor
38	Derived field operation error
39	Derived field operation error during build index

14 Event Management

01	Duplicate event monitor
02	Event monitor not present
03	Machine event requires specification of a compare value
04	Wait on event attempted while masked
05	Disable timer event monitor invalid
06	Signal timer event monitor invalid

16 Exception Management

01	Exception description status invalid
02	Exception state of process invalid
03	Invalid invocation address

18 Independent Index

01	Duplicate key argument in index

1A Lock State

01	Invalid lock state
02	Lock request not grantable
03	Invalid unlock request
04	Invalid object lock transfer request
05	Invalid space location unlock

1C Machine-Dependent Exception

01	Machine-dependent request invalid
02	Program limitation exceeded
03	Machine storage limit exceeded
04	Object storage limit exceeded
06	Lock limit exceeded
07	Modify main storage pool controls invalid
08	Requested function not valid

1E Machine Observation

01	Program not observable

20 Machine Support

01	Diagnose
02	Machine check
03	Function check

22		Object Access
	01	Object not found
	02	Object destroyed
	03	Object suspended
	04	Object not eligible for operation
	05	Object not available to process
	06	Object not eligible for destruction
24		Pointer Specification
	01	Pointer does not exist
	02	Pointer type invalid
	03	Pointer addressing invalid object
	04	Pointer not resolved
26		Process Management
	02	Queue full
28		Process State
	01	Process ineligible for operation
	02	Process control space not associated with a process
	0A	Process attribute modification invalid
2A		Program Creation
	01	Program header invalid
	02	ODT syntax error
	03	ODT relational error
	04	Operation code invalid
	05	Invalid op code extender field
	06	Invalid operand type
	07	Invalid operand attribute
	08	Invalid operand value range
	09	Invalid branch target operand
	0A	Invalid operand length
	0B	Invalid number of operands
	0C	Invalid operand ODT reference
	0D	Reserved bits are not zero
2C		Program Execution
	01	Return instruction invalid
	02	Return point invalid
	03	Stack control invalid
	04	Branch target invalid
	05	Activation in use by invocation
	06	Instruction cancellation
	07	Instruction termination
2E		Resource Control Limit
	01	User profile storage limit exceeded
30		Journal
	01	Apply journal changes failure
	02	Entry not journaled
	03	Maximum objects through a journal port limit exceeded
	04	Invalid journal space
	05	Maximum journal spaces attached
	06	Journal space not at a recoverable boundary
	07	Journal ID not unique
	08	Object already being journaled
	09	Transaction limit list exceeded
32		Scalar Specification
	01	Scalar type invalid
	02	Scalar attributes invalid
	03	Scalar value invalid
34		Source/Sink Management
	01	Source/sink configuration invalid
	02	Source/sink physical address invalid
	03	Source/sink object state invalid
	04	Source/sink resource not available
36		Space Management
	01	Space extension/truncation
	02	Invalid space modification
38		Template Specification
	01	Template value invalid
	02	Template size invalid
	03	Materialization length exception
3A		Wait Time-Out
	01	Dequeue
	02	Lock
	03	Wait on event
	04	Space location lock wait
3C		Service
	01	Invalid service session state
	02	Unable to start service session

3E Commitment Control

01 Invalid commit block status change
03 Commit block is attached to process
04 Commit block controls uncommitted changes
06 Commitment control resource limit exceeded
08 Object under commitment control being incorrectly journaled
10 Operation not valid under commitment control
11 Process has attached commit block
12 Objects under commitment control
13 Commit block not journaled
14 Errors during decommit
15 Object ineligible for commitment control
16 Object ineligible for removal from commitment control

40 Dump Space Management

01 Dump data size limit exceeded•
02 Invalid dump data insertion
03 Invalid dump space modification
04 Invalid dump data retrieval

02 Access Group

0201 Object Ineligible for Access Group

An attempt was made to insert an object into an access group. The operation could not be performed for one of the following reasons:

- The object is temporary, or the object is permanent and the access group is temporary.

- The object is restricted by the machine from membership in an access group.

Information Passed:

• Access group	System pointer
• Object to be inserted (binary 0 for objects not yet created)	System pointer

Instructions Causing Exception:

- Any create instruction that specifies an access group in the create template

- Signal Exception

04 Access State

0401 Access State Specification Invalid

An access state not supported by the machine was specified for an object.

Information Passed:

• The invalid access state	Char(1)

Instructions Causing Exception:

- Set Access State

- Signal Exception

06 Addressing

0601 Space Addressing Violation

An attempt has been made to operate outside the current extent of the space contained in a system object.

Information Passed:

• Object referenced	System pointer
• Offset specified	Bin(4)

Instructions Causing Exception:

- Any instruction using a pointer or scalar as an operand
- Any instruction using a scalar as an index, a length suboperand, or a space pointer as a base suboperand
- Signal Exception

0602 Boundary Alignment

A program object has been referenced, and it does not have the proper alignment relative to the beginning of a space. Pointers must always be 16-byte aligned. Program objects that are not pointers must have at least the alignment specified by the ODT entry.

Information Passed:

• Addressability to pointer or template	Space pointer

Instructions Causing Exception:

- Any instruction having a pointer operand or a template operand that requires a specific boundary alignment

0603 Range

A subscript value in a compound operand array reference is outside the range defined for the array. A subscript value of less than 1 or greater than the number of elements defined by the array causes this exception.

A reference to a string has a position and/or length that exceeds the bounds of the string. A compound operand that defines a character string that does not completely fall within the bounds of the base character string was referenced. A substring with position (P) e1 and length (L) e1 does not meet the following constraint (n is the length of the base string):

$$P + L - 1 \leq n$$

Instructions Causing Exception:

- All instructions that use scalar or pointer operands
- Signal Exception

0604 External Data Object Not Found

An unsuccessful attempt was made to resolve a data pointer. The external data object specified by the initial value of the data pointer was not found in the process activations. If a program name was specified in the symbolic address, then only that program's activation is considered for resolution. If no program was specified, then all of the programs with activations in the process are considered for data pointer resolution.

Information Passed:

External data object name	Char(32)

Instructions Causing Exception:

- Any instruction that references an external data object through a data pointer.
- Any instruction where a data pointer is used as the scalar value for an index of a length suboperand. This includes scalar and pointer operands that may be subscripted.
- Signal Exception
- Compare Pointer Addressability
- Compare Pointer for Space Addressability
- Convert Character to Numeric
- Convert External Form to Numeric
- Convert Numeric to Character
- Copy Bytes Left Adjusted
- Copy Bytes Left Adjusted With Pad
- Copy Bytes Right Adjusted
- Copy Bytes Right Adjusted With Pad
- Copy Numeric Value
- Edit
- Materialize Pointer
- Resolve Data Pointer
- Set Data Pointer Addressability
- Set Data Pointer Attributes
- Set Space Pointer From Pointer
- Set System Pointer From Pointer

0605 Invalid Space Reference

An attempt was made to address a space contained in an object that has no space.

Instruction Causing Exception:

- Set Space Pointer from Pointer

0606 Optimized Addressability Invalid

An instruction attempted to use the internally optimized value of a space pointer that was invalid due to the failure of a prior instruction in trying to access the pointer's value.

The machine may optimize the retrieval of a pointer's value by using the value retrieved on one instruction for use by multiple instructions that have need to reference the pointer's value. This avoids the overhead of continually retrieving the pointer's value from storage for every instruction that would have need to use it. If, in attempting to retrieve the pointer's value, an exception is detected, the machine marks the internally optimized value as invalid. This is done to provide for detecting the invalid addressability upon subsequent execution of instructions that depend on the internally optimized value. These instructions have no provision for retrieving the pointer's value from storage. These instructions will not redetect the original exception, but instead detect the optimized addressability invalid exception for this condition. This condition can occur when an exception is detected during an attempt to retrieve a pointer's value and the exception is ignored which causes execution of the program to continue without successfully retrieving the pointer's value.

This exception may not be detected on certain cases of internal machine optimizations that may be performed on references to space pointer machine objects. A reference to the space data addressed by the pointer is necessary to ensure consistent detection of this exception. Although the exception may not be detected for initial operations, it will be detected on any subsequent operation that references the space data addressed by the space pointer machine objects.

The optimization of retrieving a pointer's value can be prevented by specifying the abnormal attribute for the pointer.

Instructions Causing Exception:

- Any instruction using a pointer or scalar as an operand

- Signal Exception

08 Argument/Parameter

0801 Parameter Reference Violation

An attempt was made to reference an internal or an external parameter for which no corresponding argument was passed.

Instructions Causing Exception:

- Any instruction that references a parameter operand

- Signal Exception

0802 Argument List Length Violation

An argument list does not properly correspond to the length required by the parameter list.

Information Passed:

• Minimum number of arguments allowed	Char(2)
• Maximum number of arguments allowed	Char(2)
• Actual number of arguments allowed	Char(2)

Instructions Causing Exception:

- Call External

- Transfer Control

- Initiate Process

- Signal Exception

0803 Argument List Length Modification Violation

An attempt was made to change the length of a variable-length argument list to a value less than 0 or greater than the maximum size of the argument list.

Instructions Causing Exception:

- Set Argument List Length

- Signal Exception

0A Authorization

0A01 Unauthorized for Operation

A reference to a permanent system object is invalid because the user profiles that provide authorization for this process do not have sufficient authorization for the object.

Information Passed:

• Object preventing execution	System pointer

Instructions Causing Exception:

- Any instruction with operands or operand lists that refer to an existing permanent system object
- Signal Exception

0A02 Privileged Instruction

The user profiles that provide authorization for this process do not authorize the use of this instruction by the process.

Instructions Causing Exception:

- Create Controller Description
- Create Logical Unit Description
- Create Network Description
- Create User Profile
- Diagnose
- Initiate Process
- Modify Resource Management Control
- Modify User Profile
- Terminate Machine Processing
- Signal Exception

0A03 Attempt to Grant/Retract Authority State to an Object That Is Not Authorized

An attempt has been made to grant or retract authority states to a specified object. The user profiles that provide authorization for this instruction are not authorized to grant or retract authorization.

Information Passed:

- System pointer to the object

Instructions Causing Exception:

- Grant Authority
- Grant-Like Authority
- Retract Authority
- Signal Exception

0A04 Special Authorization Required

An attempt has been made to execute an instruction requiring special authorization. The user profiles that provide authorization for the process do not have the proper authorization.

Instructions Causing Exception:

- Materialize Process
- Modify Process
- Suspend Process
- Resume Process
- Terminate Process
- Modify Machine Attributes
- Request I/O For Load or Dump Requests
- Set Access State
- Suspend Object
- Signal Exception

0A05 Create/Modify User Profile Beyond Level of Authorization

A Create or Modify User Profile instruction has attempted to set a privileged instruction or special authorization state in the user profile that is being created or modified. The user profiles that provide authorization to the process that is executing the create or modify instruction are not authorized.

Instructions Causing Exception:

- Create User Profile
- Modify User Profile
- Signal Exception

0C Computation

0C01 Conversion

A scalar value cannot be converted to the necessary type in this instruction.

Instructions Causing Exception:

- Convert Character to Hex
- Convert External Form to Numeric
- Convert SNA to Character
- Convert MRJE to Character
- Convert BSC to Character
- Signal Exception

0C02 Decimal Data

The sign or digit codes of a decimal operand, either packed or zoned, contain an invalid value. For packed and zoned format, either the sign is outside the valid range of A through F or a digit field is outside the range 0 through 9.

Instructions Causing Exception:

- Add Numeric
- Compare Numeric Value
- Convert Character to Numeric
- Convert Decimal Form to Floating-Point
- Convert Numeric to Character
- Copy Numeric Value
- Divide
- Divide With Remainder
- Edit
- Extract Magnitude
- Multiply
- Negate
- Remainder
- Scale
- Subtract Numeric
- Sum
- Signal Exception

0C03 Decimal Point Alignment

The value of a numeric operand cannot be aligned in a 31-digit decimal field. Decimal point alignment was attempted by padding with 0's on the right. Nonzero digits would have to be truncated on the left to fit the aligned value into a 31-digit decimal field.

Instructions Causing Exception:

- Add Numeric
- Compare Numeric Value
- Subtract Numeric
- Signal Exception

0C04 Edit Digit Count

The number of digit position characters in the mask operand of an Edit instruction is not equal to the number of digits in the source operand value.

Instructions Causing Exception:

- Edit
- Signal Exception

0C05 Edit Mask Syntax

The characters of the mask operand do not follow the valid syntax rules for an Edit instruction.

Instructions Causing Exception:

- Edit
- Signal Exception

0C06 Floating-Point Overflow

The result of a floating-point operation is finite and not an invalid value, but its exponent is too large for the target floating-point format. The signed exponent has exceeded 127 in short format or 1023 in long format.

Information Passed:

• Floating-point value attributes	Char(1)
– Normal bias	Bit 0
– Modified bias	Bit 1
– Rounded to short floating-point precision	Bit 2
– NaN	Bit 3
– Reserved (binary 0)	Bits 4-7
• Reserved (binary 0)	Char(7)
• Overflowed floating-point value	Floating-(8 point
• Reserved (binary 0)	Char(16)

Instructions Causing Exception:

- Add Numeric
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Numeric to Character
- Convert Decimal Form to Floating-Point
- Copy Numeric Value
- Divide
- Extract Magnitude
- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C07 Floating-Point Underflow

The result of a floating-point operation is not zero but has too small an exponent for the destination's format without being denormalized. The signed exponent is less than -126 in short format or less than -1022 in long format.

Information Passed:

• Floating-point value attributes	Char(1)
– Normal bias	Bit 0
– Modified bias	Bit 1
– Rounded to short floating-point precision	Bit 2
– NaN	Bit 3
– Reserved (binary 0)	Bits 4-7
• Reserved (binary 0)	Char(7)
• Underflowed floating-point value	Floating-(8 point
• Reserved (binary 0)	Char(16)

Instructions Causing Exception:

- Add Numeric
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Numeric to Character
- Convert Decimal Form to Floating-Point
- Copy Numeric Value
- Divide
- Extract Magnitude
- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C08 Length Conformance

The operand lengths or resultant value length or both do not conform to the rules of the instruction:

CVTHC — Twice the length of the source operand must be less than or equal to the length of the receiver operand.

CVTCH — The length of the operand must be less than or equal to twice the length of the receiver operand.

CVTMC — The length of a record in the receiver was not enough to contain the converted form of a record from the source.

EDIT — The length of the resultant edited value must be equal to the length of the receiver operand.

SCAN — The length of the compare operand must not be greater than the length of the base string.

SEARCH — The length of the find operand plus the value of the location operand must be less than or equal to the length of an element of the array operand.

XLATE — The source and receiver operands must be the same length.

Instructions Causing Exception:

- Convert Character to Hex
- Convert Hex to Character
- Convert MRJE to Character
- Edit
- Extended Character Scan
- Scan
- Search
- Signal Exception
- Translate

0C09 Floating-Point Invalid Operand

A floating-point invalid operand condition is caused by one of the following conditions:

- A source operand is an unmasked NaN.
- Addition of infinities of different signs and subtraction of infinities of the same sign.
- Multiplication of zero times infinity.
- Division of zero by zero or infinity by infinity.
- Computing the sine, cosine, or tangent function for infinity.
- Computing the arc tangent, exponential, logarithm, square root, or power function for infinity when in projective infinity mode.
- Floating-point values compared unordered and no branch or indicator options are specified for the unordered, negation of unordered, equal, or negation of equal conditions on compare numeric value.
- An unordered resultant condition occurred on a computational instruction because the result was NaN, and branch or indicator conditions are specified but unordered, negation of unordered, zero, or negation of zero conditions are not specified.

Information Passed:

- Exception type Char(1)
  Hex 00 = Invalid arithmetic
  operation or operand
  is unmasked NaN.
  Hex 01 = Invalid branch or
  indicator conditions.
  Hex 02 through hex FF are reserved.
- Reserved (binary 0) Char(31)

Instructions Causing Exception:

- Add Numeric
- Compare Numeric Value
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Floating-Point to Decimal Form
- Convert Numeric to Character
- Copy Numeric Value
- Divide
- Extract Magnitude
- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C0A Size

An operand was too small to contain a result. This condition is detected only when a fixed-point result is too large to be assigned to a fixed-point receiver. The receiver operand is set with the result of the operation truncated to the receiver size.

Instructions Causing Exception:

- Add Numeric
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert External Form
- Convert Numeric to Character
- Copy Numeric Value
- Divide
- Divide With Remainder
- Extract Magnitude
- Multiply
- Negate
- Remainder
- Scale
- Subtract Numeric
- Sum
- Signal Exception
- Trim Length

0C0B Zero Divide

An attempt was made to divide by 0 on a fixed-point divide operation.

Instructions Causing Exception:

- Divide
- Divide With Remainder
- Remainder
- Signal Exception

0C0C Invalid Floating-Point Conversion

This exception is detected on a conversion from binary floating-point to other than a binary floating-point format because overflow, infinity, or NaN is detected before conversion is complete.

Information Passed:

• Floating-point value attributes	Char(1)
– Normal bias	Bit 0
– Modified bias	Bit 1
– Reserved (binary 0)	Bit 2
– NaN	Bit 3
– Infinity	Bit 4
– Reserved (binary 0)	Bits 5-7
• Reserved (binary 0)	Char(7)
• Invalid floating-point value	Floating-(8 point
• Reserved (binary 0)	Char(16)

Instructions Causing Exception:

- Add Numeric
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Floating-Point to Decimal Form
- Convert Numeric to Character
- Copy Numeric Value
- Divide
- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C0D Floating-Point Inexact Result

This exception is signaled when the rounded result of an operation is not exact because of one of the following conditions:

- The rounded result of an operation is not exact because a value is modified (rounded) to fit in a receiver operand and nonzero fraction digits are lost.
- The occurrence of a floating-point overflow condition when that condition is masked and the result is no longer exact because it is set to infinity.

Information Passed:

• Reserved (binary 0)	Char(32)

Instructions Causing Exception:

- Add Numeric
- Compare Numeric Value
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Decimal Form to Floating-Point
- Convert Floating-Point to Decimal Form
- Copy Numeric Value
- Divide
- Extract Magnitude
- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C0F Master Key Not Defined

The cipher operation requested use of the master key but the master key has not been defined by the Modify Machine Attributes instruction.

Instructions Causing Exception:

- Cipher
- Cipher Key
- Signal Exception

0C10 Weak Key Not Valid

The key supplied in the template is a weak key and cannot be accepted by the cipher operation.

Instructions Causing Exception:

- Cipher
- Cipher Key
- Modify Machine Attributes
- Signal Exception

0C11 Key Parity Invalid

The key supplied in the template does not have odd parity in each byte and is, therefore, unacceptable for the cipher operation.

Information Passed:

• Offset (byte) to the key field	Bin(2)
• Reserved	Char(6)

Instructions Causing Exception:

- Cipher
- Modify Machine Attributes
- Signal Exception

0C0E Floating-Point Zero Divide

This exception is signaled for a floating-point division operation if the divisor is zero and the dividend is a finite nonzero number.

Instructions Causing Exception:

- Compute Math Function Using Two Input Values
- Divide
- Signal Exception

0E Context Operation

0E01 Duplicate Object Identification

An attempt was made to place addressability in a context to an object having the same name, type, and subtype as an existing entry in the context.

Information Passed:

- System pointer to the existing object

• Object identification	Char(32)
– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)

Instructions Causing Exception:

- All create instructions
- Modify Addressability
- Rename Object
- Signal Exception

0E02 Object Ineligible For Context

An attempt was made to delete addressability to an object of a type that may be addressed only by the machine context, or an attempt was made to place addressability to an object in a temporary or permanent context that may be addressed only by the machine context.

Information Passed:

- System pointer to object

• Object identification	Char(32)
– Object type	Char(1)
– Object subtype code	Char(1)
– Object name	Char(30)

Instructions Causing Exception:

- Modify Addressability
- Signal Exception

10 Damage

1002 Machine Context Damage State

The machine context cannot be referenced because it is in the damaged state. The machine context rebuild option of the Reclaim instruction can be used to correct the problem or an IPL can correct the problem.

Information Passed:

- Reserved (binary 0) Char(16)
- VLOG dump ID Char(8)
- Error class Bin(2)
- The error class codes for the type of damage detected are as follows:

Hex 0000 = Previously marked damaged

Hex 0001 = Detected abnormal condition

Hex 0002 = Locally invalid device sector

Hex 0003 = Device failure

- Auxiliary storage device failure Bin(2)

  This field is defined for error classes hex 0002 and hex 0003. It is the OU number of the failing device or 0 for a main storage failure.

- Reserved (binary 0) Char(100)

Instructions Causing: Exception:

- Materialize Context
- Resolve System Pointer
- Any instruction that resolves a system object that is located by the machine context
- Signal Exception

1004 System Object Damage State

A system object cannot be accessed because it is in the damaged state.

Information Passed:

- System pointer to the damaged object System pointer
- VLOG dump ID Char(8)
- Error class Bin(2)
- The error class codes for the type of damage detected are as follows:

Hex 0000 = Previously marked damaged

Hex 0001 = Detected abnormal condition

Hex 0002 = Locally invalid device sector

Hex 0003 = Device failure

- Auxiliary storage device indicator Bin(2)

  This field is defined for error classes hex 0002 and hex 0003. It is the OU number of the failing device or 0 for a main storage failure.

- Reserved (binary 0) Char(100)

Instructions Causing Exception:

- Any instruction that references a system object
- Signal Exception

1044 Partial System Object Damage

Partial damage to a system object has been detected.

Information Passed:

• System pointer to the damaged object	System pointer
• VLOG dump ID	Char(8)
• Error Class	Bin(2)

- The error class codes for the type of damage detected are as follows:

Hex 0000	=	Previously marked damaged
Hex 0001	=	Detected abnormal condition
Hex 0002	=	Locally invalid device sector
Hex 0003	=	Device failure

• Auxiliary storage device indicator	Bin(2)

This field is defined for error classes hex 0002 and hex 0003. It is the OU number of the failing device or 0 for a main storage failure.

• Reserved (binary 0)	Char(100)

Instructions Causing Exception:

- Any instruction that references a system object
- Signal Exception

12 Data Base Management

1201 Conversion Mapping Error

During conversions of a numeric field from one numeric data representation to another numeric data representation, the source value was too large to fit in the destination field, the digit (nonzone) portion of a packed or zoned source field contained an invalid numeric encoding, or the sign encoding was invalid.

Information Passed:

The following data is provided:

Cursor	System pointer
Data space number	Bin(2)
Ordinal entry number (0 if signaled during an Insert Data Space Entry or an Insert Sequential Data Space Entries instruction)	Bin(4)
Number of fields in error	Bin(2)
Field data (repeated for each field that is in error)	
– Field number	Bin(2)
– Error type	Char(2)

The ordinal entry number will contain a value of binary zero when the exception occurs when inserting new data space entries or performing group-by and join operations.

The field number is the relative location of the field as specified when creating the cursor. A field number of 1 is the first field in the data field location.

The error type values are as follows:

- Hex 0002–Decimal Data: (1) Sign encoding is invalid for packed or zoned format, or (2) digit encoding is invalid for packed or zoned format.

- Hex 0006–Floating-Point Overflow: During conversion, a floating-point value exceeded the maximum value that can be represented.

- Hex 0007–Floating-Point Underflow: During conversion, a floating-point value became less than the minimum value that can be represented.

- Hex 0009–Floating-Point Invalid Operand: A floating-point NaN was used as an operand to convert from long floating-point format to short floating-point format, from short floating-point format to long floating-point format, or from floating-point to binary.

- Hex 000A–Size: The destination field is too small to hold all significant digits of the source field.

- Hex 000C–Invalid Floating-Point Conversion: A floating-point NaN or floating-point infinity was used as an operand to convert floating-point to packed decimal.

- Hex 000D–Floating-Point Inexact Result: In a conversion operation, a floating-point value had at least one bit of precision rounded away.

These errors are equivalent to the data base hexadecimal exceptions numbered 0C02, 0C06, 0C07, 0C09, 0C0A, 0C0C, 0C0D, and occur for similar reasons.

Instructions Causing Exception:

- Copy Data Space Entries

- Insert Data Space Entry

- Insert Sequential Data Space Entries

- Retrieve Data Space Entry

- Retrieve Sequential Data Space Entries

- Update Data Space Entry

- Signal Exception

1202 Key Mapping Error

During conversions of a numeric field from one numeric data representation to another numeric data representation, the source value was too large to fit in the destination field, the digit (nonzone) portion of a packed or zoned source field contained an invalid numeric encoding, or the sign encoding was invalid.

Information Passed:

The following data is provided:

Cursor	System pointer
Data space number	Bin(2)
Binary 0	Char(4)
Number of fields in error	Bin(2)
Field data (repeated for each field that is in error)	
– Field number	Bin(2)
– Error type	Char(2)

The field number is the relative location of the field as specified when creating the cursor. A field number of 1 is the first field in the data field location.

The error type values are as follows:

- Hex 0002–Decimal Data: (1) Sign encoding is invalid for packed or zoned format, or (2) digit encoding is invalid for packed or zoned format.

- Hex 0006–Floating-Point Overflow: During conversion, a floating-point value exceeded the maximum value that can be represented.

- Hex 0007–Floating-Point Underflow: During conversion, a floating-point value became less than the minimum value that can be represented.

- Hex 0009–Floating-Point Invalid Operand: A floating-point NaN was used as an operand to convert from long floating-point format to short floating-point format, from short floating-point format to long floating-point format, or from floating-point to binary.

- Hex 000A–Size: The destination field is too small to hold all significant digits of the source field.

- Hex 000B–A substring set cursor operation was specified. The byte string terminated in the middle of a binary field.

- Hex 000C–Invalid Floating-Point Conversion: A floating-point NaN or floating-point infinity was used as an operand to convert floating-point to packed decimal.

- Hex 000D–Floating-Point Inexact Result: In a conversion operation, a floating-point value had at least one bit of precision rounded away.

- Hex 00F0–A substring set cursor operation was specified. The byte string terminated in the middle of a floating-point field.

These errors are equivalent to the data base hexadecimal exceptions numbered 0C02, 0C06, 0C07, 0C09, 0C0A, 0C0C, 0C0D, and occur for similar reasons.

Instructions Causing Exception:

- Copy Data Space Entries (mapping from template)

- Materialize Cursor Attributes (mapping key out to buffer)

- Retrieve Sequential Data Space Entries (mapping key to buffer)

- Set Cursor (mapping key in/out)

- Signal Exception

1203 Cursor Not Set

An attempt was made to perform a data base operation using a cursor that is not set to address a data space entry.

Information Passed:

• System pointer to cursor	
• Data space number	Bin(2)

The data space number will be zero for a non-join cursor.

Instructions Causing Exception:

- Retrieve Data Space Entry

- Set Cursor

- Signal Exception

1204 Data Space Entry Limit Exceeded

The operation caused the user-provided maximum number of entries limitation for the data space to be exceeded.

Information Passed:

• Cursor (binary 0 for instruction not involving a cursor)	System pointer
• Data space	System pointer

Instructions Causing Exception:

- Copy Data Space Entries
- Data Base Maintenance (insert default entries and insert deleted entries option)
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Signal Exception

1205 Data Space Entry Already Locked

An attempt has been made to lock a data space entry using the Set Cursor instruction when the data space entry is already locked to a cursor (this cursor or another cursor) or to a commit block that is not attached to this process. A system pointer to the process control space of the process that activated the cursor or attached the commit block that holds the lock is returned.

Information Passed:

• Cursor	System pointer
• Data space	Bin(2)
• Ordinal entry number	Bin(4)
• Return code (bit significant) – Hex 00 = Locked to another process – Hex 01 = Locked to current process	Char(1)
• Reserved (binary 0)	Char(9)
• Process control space	System pointer

Instructions Causing Exception:

- Set Cursor
- Signal Exception

1206 Data Space Entry Not Found

An attempt has been made to refer to a data space entry that could not be found because the entry has been deleted or its key has been omitted from the data space index.

Information Passed:

• Cursor	System pointer
• Data space number	Bin(2)

Instructions Causing Exception:

- Retrieve Data Space Entry
- Set Cursor
- Signal Exception

1207 Data Space Index Invalid

The index specified for a data base operation is not usable.

Information Passed:

• Cursor (binary 0 for instructions not involving cursor)	System pointer
• Data space index	System pointer

Instructions Causing Exception:

- Activate Cursor
- Copy Data Space Entries
- Retrieve Data Space Entry
- Retrieve Sequential Data Space Entries
- Set Cursor
- Signal Exception

1208 Incomplete Key Description

The cursor cannot be set by key for this data space index because the output mapping template used to create this cursor failed to provide a description of each field that comprises the key.

Information Passed:

• Cursor	System pointer
• Data space number of the key format selected	Bin(2)

Instructions Causing Exception:

- Copy Data Space Entries
- Create Cursor
- Set Cursor
- Signal Exception

1209 Duplicate Key Value in Existing Data Space Entry

An attempt has been made to insert or update a data space entry in a data space over which a unique keyed index has been built, and the data space entry has a key value identical to an existing data space entry addressed by the index.

Information Passed:

• Cursor (binary 0 for operations not involving a cursor)	System pointer
• Data space index	System pointer
• The data space number of the entry associated with the key already in the data space index	Bin(2)
• The ordinal number of the entry associated with the key already in the data space index	Bin(4)
• The data space number of the entry that was being added or changed and caused the exception	Bin(2)
• The ordinal number of the entry that was being changed and caused the exception (0 if an insert was being attempted)	Bin(4)

Instructions Causing Exception:

- Apply Journaled Changes
- Copy Data Space Entries
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Update Data Space Entry
- Signal Exception

120A End of Path

The end of an access path was reached when an attempt was made to position a cursor.

Information Passed:

• Cursor	System pointer

Instructions Causing Exception:

- Retrieve Sequential Data Space Entries
- Set Cursor
- Signal Exception

120B Duplicate Key Value Detected

While creating or rebuilding a data space index with the unique key attribute, entries were found to generate the same key value. The build detected up to a maximum of 20 duplicate key values before terminating.

Information Passed:

• Data space index	System pointer
• Number of duplicates detected	Bin(2)
• (Repeated for each duplicate)	
– Data space number of first entry	Bin(2)
– Ordinal number of first entry	Bin(4)
– Data space number of second entry	Bin(2)
– Ordinal number of second entry	Bin(4)

Instructions Causing Exception:

- Create Data Space Index
- Data Base Maintenance (rebuild option)
- Signal Exception

120D No Entries Locked

No data space entries were locked to this cursor.

Information Passed:

• Cursor	System pointer

Instructions Causing Exception:

- Delete Data Space Entry
- Update Data Space Entry
- Signal Exception

120F Duplicate Key Value in Uncommitted Data Space Entry

An attempt has been made to insert or update a data space entry in a data space over which a unique keyed index has been created, and the data space entry has a key value identical to a data space entry key value that has been deleted or changed under commitment control but is still reserved by the index. The insert or update cannot be done until the uncommitted changes are committed.

Information Passed:

• Cursor (binary 0 for operations not involving a cursor)	System pointer
• Data space index	System pointer
• The data space number of the entry associated with the key reserved in the data space index	Bin(2)
• The ordinal number of the entry associated with the key reserved in the data space index	Bin(4)
• The data space number of the entry that was being added or changed and caused the exception	Bin(2)
• The ordinal number of the entry that was being changed and caused the exception (zero if an insert was being attempted)	Bin(4)

Instructions Causing Exception:

- Copy Data Space Entries
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Update Data Space Entry
- Signal Exception

1213 Invalid Mapping Template

An error was detected in a mapping template. The data space number indicates the template in the mapping template list that contains the error. This field will equal 1 for a group-by mapping template, and will reference the appropriate template for the per data space mapping templates for Create Cursor and Create Data Space Index instructions. This field will contain a zero when an intermediate mapping table is missing.

The template field number indicates which field number is in error for the scalar part of the mapping template (number of bytes in mapping template and mapping type). The contents of the reserved field immediately following the mapping type field is ignored (only present when intermediate mapping is specified).

The template field number also indicates the field in error for input and output mapping tables. Field number equal 0 indicates the template field number of bytes in the mapping template, field number equal 1 indicates field designating input mapping type, field number equal 2 indicates field designating output mapping type, field number equal 3 indicates field designating number of fields in the input/output mapping tables, and so forth. Each specification in the input/output tables is considered one field for counting purposes.

When an intermediate mapping table is in error, the template field number will equal 0. The offset to field in error field will designate the offset to the field from the start of the intermediate mapping table. Offset equal 0 designates the field number of intermediate mapping specifications, and so forth.

Possible errors are an invalid value, a value that exceeds allowed range, a length that is invalid for the specified type or a type that is inconsistent with the type specified for the field in the data space, key description, or an intermediate buffer description.

The invalid mapping template exception will not be signaled when a data pointer fails verification. The normal exception associated with verifying data pointers will be signaled instead.

Information Passed:

- Data space number Bin(2)
- Template field number Bin(2)
  (valid only for input/output mapping tables and the scalar part of the mapping template; number of bytes in template, input mapping type, output mapping type)
- Template type Char(1)
  Hex 00 = Per data space mapping template (input/output mapping table and scalar part of template)
  Hex 01 = Per data space mapping template (output intermediate mapping table)
  Hex 02 = Group-by intermediate mapping table
  Hex 03 = Derived key intermediate mapping table
- Reserved Char(1)

• Offset to field in error (valid only for intermediate mapping table errors)	Bin(4)
• Error type (valid only for intermediate mapping table errors)	Char(2)
– Operand 1 error	Bit(0)
– Operand 2 error	Bit(1)
– Result field error	Bit(2)
– Operation field error	Bit(3)
– Missing intermediate mapping table	Bit(4)
– Reserved	Bit(5)
– Array position of data pointer is invalid	Bit(6)
– Array position of translate table is invalid	Bit(7)
– Field number invalid	Bit(8)
– Field type invalid	Bit(9)
– Field length invalid	Bit(10)
– Operand location/type invalid	Bit(11)
– Start character invalid	Bit(12)
– End character invalid	Bit(13)
– Rounding mode invalid	Bit(14)
– Reserved field invalid	Bit(15)

Instructions Causing Exception:

- Create Cursor

- Signal Exception

1214 Invalid Selection Template

An error was detected in a selection template. The data space number indicates which template in the selection template list contains the error. This field will equal 1 for a group-by selection template, and will reference the appropriate template for the per data space mapping template for Create Cursor and Create Data Space Index.

The offset to field in error indicates which field is in error in the selection template. The offset equal 0 designates the field length of selection template is in error, offset equal 4 designates the field number of selection descriptors is in error, and so forth.

The invalid selection template exception will not be signaled when a data pointer fails verification. The normal exception associated with verifying data pointers will be signaled instead.

Information Passed:

• Data space number (position list)	Bin(2)
• Offset to field in error	Bin(4)
• Selection template type Hex 00 = Per data space selection Hex 01 = Group-by selection	Char(1)
• Reserved	Char(1)
• Selection descriptor errors	Char(2)
– Descriptor type error	Bit(0)
– Operand/operation error	Bit(1)
– Maximum number specifications exceeded	Bit(2)
– Literal content in error	Bit(3)
– Reserved	Bit(4-7)
– Operand location invalid	Bit(8)
– Field number invalid	Bit(9)
– Starting offset invalid	Bit(10)
– Ending offset invalid	Bit(11)
– Array position of data pointer invalid	Bit(12)
– Offset to pattern descriptor invalid	Bit(13)
– Number pattern descriptor invalid	Bit(14)
– Reserved field invalid	Bit(15)
• Pattern descriptor error types	Char(2)
– Descriptor type invalid	Bit(0)
– Descriptor field invalid	Bit(1)
– Reserved	Bit(2-7)
– Field location invalid	Bit(8)
– Field number invalid	Bit(9)
– Starting offset invalid	Bit(10)
– Ending offset invalid	Bit(11)
– Array position of data pointer invalid	Bit(12)
– Span type invalid	Bit(13)
– Span width invalid	Bit(14)
– Reserved field invalid	Bit(15)

Instructions Causing Exception:

- Create Cursor

- Signal Exception

1215 Data Space Not Addressed by Index

An entry in the data space list does not address the same data space that is addressed by the corresponding entry in the data space list defined for the data space index.

Information Passed:

• Entry in the data space list of the Create Cursor instruction template	Space pointer

Instructions Causing Exception:

- Create Cursor

- Signal Exception

1216 Data Space Not Addressed by Cursor

An entry in the data space list does not address the same data space that is addressed by the corresponding list that is defined for the cursor.

Information Passed:

• Cursor	System pointer
• Entry in the data space list of the Activate Cursor instruction template	Space pointer

Instructions Causing Exception:

- Activate Cursor

- Signal Exception

1217 Key Value Changed Since Set Cursor

The data space index key for the entry currently addressed by the cursor has changed since the cursor was set. The former value of the key was instrumental in finding the entry and is no longer valid; therefore, the entry is no longer the expected entry.

Information Passed:

• Cursor	System pointer
• Data space number	Bin(2)

Instructions Causing Exception:

- Retrieve Data Space Entry
- Set Cursor
- Signal Exception

1219 Invalid Rule Option

The cursor has addressability to a data space index and the current cursor setting allows only rule options of relative or ordinal.

Information Passed:

• Cursor	System pointer

Instructions Causing Exception:

- Retrieve Sequential Data Space Entries
- Set Cursor
- Signal Exception

121A Data Space Entry Size Exceeded

The sum of the field lengths in the entry definition template exceeds 32 766 bytes which is the maximum size allowed for a data space entry.

Instructions Causing Exception:

- Create Data Space
- Signal Exception

121B Logical Data Space Entry Size Limit Exceeded

The user's view of the data space entry (defined by the mapping code) exceeds 32 766 bytes, which is the maximum size allowed.

Information Passed:

• Template number (position list)	Bin(2)
• Template type – Hex 00 = Input mapping template – Hex 01 = Output mapping template – Hex 02 = Intermediate mapping template – Hex 03 = Group-by output mapping template – Hex 04 = Group-by intermediate mapping template	Char(1)

Instructions Causing Exception:

- Create Cursor
- Signal Exception

121C Key Size Limit Exceeded

The sum of the key field lengths plus the specified fork characters exceeds 120 bytes, which is the maximum size allowed for a data space index key.

Information Passed:

- Data space number Bin(2)

Instructions Causing Exception:

- Create Data Space Index
- Signal Exception

121D Logical Key Size Limit Exceeded

The user's view of the data space index key exceeds 32 766 bytes, which is the maximum size allowed.

Information Passed:

- Data space number Bin(2)

Instructions Causing Exception:

- Create Cursor
- Signal Exception

121E Selection Routine Buffer Size Limit Exceeded

The selection routine's view of the data space entry as specified in the selection specification exceeds 32 767 bytes, which is the maximum size allowed.

Information Passed:

- Data space number Bin(2)

Instructions Causing Exception:

- Create Data Space Index
- Signal Exception

121F User Exit Routine Criteria Not Satisfied

The specified user exit routine failed to meet the criteria for a data space user exit routine.

Information Passed:

- User exit routine System pointer

Instructions Causing Exception:

- Create Data Space Index
- Signal Exception

1221 Unable to Maintain a Unique Key Data Space Index

An attempt has been made to insert or update a data space entry in a data space over which a unique keyed index exists that has been implicitly invalidated.

Information Passed:

• Cursor (binary 0 for operations not involving a cursor)	System pointer
• Data space	System pointer
• Data space index (invalidated)	System pointer

Instructions Causing Exception:

- Apply Journaled Changes
- Copy Data Space Entries
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Update Data Space Entry
- Signal Exception

1222 Data Space Index with User Exit Selection Routine Build Termination

While creating or rebuilding a data space index that contains a user exit selection routine, data space entries that resulted in an error in the selection routine were encountered. The build, before termination, found up to 20 instances of these types of errors. The instruction is terminated.

Information Passed:

• Data space index (binary 0's if signaled during creation)	System pointer
• Number of errors detected (repeated for each selection routine error)	Bin(2)
• Error descriptor (repeated for each selection routine error)	Char(8)
– Data space number	Bin(2)
– Ordinal entry number	Bin(4)
– Reason code Hex 01 = Selection mapping error Hex 02 = Selection routine failure Hex 03 = Error in invoking selection routine	Char(1)
Reserved (binary 0)	Char(1)

Instructions Causing Exception:

- Activate Cursor (over delayed maintenance data space index)
- Create Data Space Index
- Data Base Maintenance (rebuild data space index option)
- Signal Exception

1223 Data Space Index User Exit Selection Routine Failure

An attempt has been made to insert or update a data space entry in a data space over which a data space index with a selection routine exists, and an error was encountered in the selection routine.

Information Passed:

• Cursor (binary 0 for operations not involving a cursor)	System pointer
• Data space	System pointer
• Data space index	System pointer
• Data space number (in the data space list of the data space index)	Bin(2)
• Ordinal entry number (0 if entry was being inserted)	Bin(4)
• Reason code Hex 01 = Selection mapping error Hex 02 = Selection routine failure Hex 03 = Error invoking selection routine	Char(1)
• Reserved	Char(1)
• Cursor data space number	Bin(2)

Instructions Causing Exception:

- Apply Journaled Changes
- Copy Data Space Entries
- Data Base Maintenance (insert default entries)
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Update Data Space Entry
- Signal Exception

1225 Invalid Data Base Operation

A data base operation was attempted through a cursor whose activation options indicated that the operation was not to be allowed.

Information Passed:

• Cursor	System pointer
• Extended activation functions (as defined in the cursor activation template)	Char(2)
• Operation attempted Hex 80 = Retrieval of data space entry Hex 40 = Update of data space entry Hex 20 = Delete of data space entry Hex 10 = Insert of data space entry	Char(1)

Instructions Causing Exception:

- Copy Data Space Entries
- Delete Data Space Entry
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Retrieve Data Space Entry
- Retrieve Sequential Data Space Entries
- Update Data Space Entry
- Signal Exception

1226 Data Space Index with Invalid Floating-Point Field Build Termination

While creating or rebuilding a data space index that contains floating-point keys, an invalid floating-point value was encountered. Up to 20 instances of these types of errors may be found before the instruction is terminated.

Information Passed:

• Data space index (binary 0 is signaled during creation)	System pointer
• Number of errors detected	Bin(2)
• Error description (repeated for each selection routine error)	
– Data space number	Bin(2)
– Ordinal entry number	Bin(4)
– Reason code Hex 01 = Floating-point NaN detected	Char(1)
– Reserved (binary 0)	Char(1)

Instructions Causing Exception:

- Create Data Space Index
- Data Base Maintenance
- Signal Exception

1227 Data Space Index Key with Invalid Floating-Point Field

An attempt was made to insert or update a data space entry in a data space under a data space index that contains floating-point key fields, or a key is being used to search a data space index that contains floating-point key fields and a floating-point key field contains an invalid value.

Information Passed:

• Cursor (binary 0 for operations not involving a cursor)	System pointer
• Data space	System pointer
• Data space index	System pointer
• Data space number (in the (data space list of the data space index)	Bin(2)
• Ordinal entry number (zero if entry was being inserted)	Bin(4)
• Reason code Hex 01 = Floating-point NaN detected	Char(1)
• Reserved	Char(1)
• Cursor data space number	Bin(2)

Instructions Causing Exception:

- Apply Journaled Changes
- Data Base Maintenance
- Copy Data Space Entries
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Set Cursor
- Update Data Space Entry
- Signal Exception

1230 Specified Data Space Entry Rejected

An attempt has been made to position a cursor to a specific data space entry but the retrieval selection criteria has rejected the entry.

Information Passed:

• Cursor	System pointer
• Data space number	Bin(2)
• Ordinal number	Bin(4)

Instructions Causing Exception:

- Retrieve Data Space Entry
- Set Cursor
- Signal Exception

1232 Join Value Changed

A join value in a data space entry field used in the current join position in the cursor has changed since the cursor was positioned. The former value of the field was instrumental in performing the join operation and is no longer valid; therefore, the entry is no longer the expected entry.

Information Passed:

• Cursor	System pointer
• Data space number in cursor associated with changed field	Bin(2)

Instructions Causing Exception:

- Retrieve Data Space Entry
- Set Cursor
- Signal Exception

1233 Data Space Index with Non-User Exit Selection Routine Build Termination

While creating or rebuilding a data space index that contains a non-user exit selection routine, data space entries were encountered which resulted in an error in the selection routine. The build, before terminating, found up to 20 instances of these types of errors. The instruction is terminated.

Information Passed:

• Data space index (binary zeros if signaled during creation)	System pointer
• Number of errors detected	Bin(2)
• Reserved	Char(4)
• Error description (repeated for each selection routine error)	Char(22)
– Data space number	Bin(2)
– Ordinal entry number	Bin(4)
– Reserved (binary 0)	Char(6)
– Error type	Char(2)
– Operand 1 field data	Char(4)
— Field number	Bin(2)
— Field location Hex 00 = Data space entry Hex 01 = Cursor intermediate buffer Hex 03 = Intermediate key buffer Hex 04 = Key field	Char(1)
— Reserved	Char(1)
– Operand 2 field data (same as operand 1 field data)	Char(4)

The field number designates the relative location of the field as specified when creating the cursor, index, or data space. Field number equal 1 is the first field in the field location.

The error type values are as follows:

Hex 0002–decimal data: (1) Sign encoded is invalid for packed or zoned format, (2) Digit encoding is invalid for packed or zoned format.

Hex 0009–floating-point invalid operand: A floating-point NaN was used as an operand in a comparison.

Instructions Causing Exception:

- Activate Cursor (over delayed maintenance data space index)
- Create Data Space Index
- Data Base Maintenance (rebuild data space index)
- Modify Data Space Index Attributes
- Signal Exception

1234 Non-User Exit Selection Routine Failure

An attempt has been made to insert, retrieve, or update a data space entry in a data space, and an error was encountered in a non-user exit selection routine.

Information Passed:

• Cursor (binary 0 for operations not involving a cursor)	System pointer
• Data space	System pointer
• Data space index (binary 0 if selection error not involving the index)	System pointer
• Join cursor ordinal positions	Char(128)
• Error description	Char(22)
– Index data space number (binary 0 if selection error not on the index)	Bin(2)
– Cursor data space number (binary 0 for operations not involving a cursor)	Bin(2)
– Ordinal entry number (0, if entry was being inserted or group-by selection)	Bin(4)
– Reserved	Char(4)
– Error type	Char(2)
– Operand 1 field data (0 if literal)	Char(4)
Field number	Bin(2)
Field location Hex 00 = Data space entry Hex 01 = Cursor intermediate buffer Hex 02 = Group-by intermediate buffer Hex 03 = Intermediate key buffer Hex 04 = Key field	Char(1)
Reserved	Char(1)
– Operand 2 field data (same as operand 1 field data)	Char(4)

The field number designates the relative location of the field as specified when creating the cursor, index, or data space. Field number equal 1 is the first field in the field location. A field number equal 0 designates there is no exception data for this operand.

The error type values are identical with those received for exception hex 1233.

The ordinal entry number will contain a binary zero value if the exception occurs while processing default values during a join operation.

The join cursor ordinal positions area is meaningful only on operations with a join cursor and is zero otherwise. It will be zero for group-by selection. Each ordinal number occupies 4 bytes. All current ordinal numbers associated with a join cursor are returned.

Instructions Causing Exceptions

- Apply Journaled Changes
- Data Base Maintenance (insert default entries)
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Retrieve Data Space Entry
- Retrieve Sequential Data Space Entry
- Set Cursor
- Update Data Space Entry
- Signal Exception

1236 No Mapping Code Specified

Cursor cannot be used to perform inserts, retrieves, or updates with the specified data space due to no data space entry input mapping code or no output mapping code specified in a Create Cursor instruction.

Information Passed:

• Cursor	System pointer
• Data space number	Bin(2)

Instructions Causing Exception:

- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Retrieve Data Space Entry
- Retrieve Sequential Data Space Entry
- Signal Exception
- Update Data Space Entry

1237 Operation Not Valid with Join Cursor

An attempt has been made to insert, update, or delete a data space entry through a join cursor.

Information Passed:

• Cursor	System pointer

Instructions Causing Exception:

- Copy-Data Space Entries
- Delete Data Space Entry
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Signal Exception
- Update Data Space Entry

1238 Derived Field Operation Error

During derived field operations, one of a variety of derived field operational errors occurred. The field number designates the relative location of the field as specified when creating the cursor or data space index. Field number equal 1 is the first field in the field location. The field location is described by the field location exception data.

The error type is identical with those for exception number hex 1201, conversion mapping error with the addition of:

- Hex 000B = An attempt has been made to divide by zero on a fixed-point divide operation.
- Hex 000E = An attemp has been made to divide by zero on a floating-point divide operation.

Information Passed:

• Cursor (0 for operations not involving a cursor)	System pointer
• Index (derived key operations)	System pointer
• Join cursor ordinal positions	Char(128)
• Index data space number (binary 0 for operations not involving the index)	Bin(2)
• Cursor data space number (binary 0 for operations not involving the cursor)	Bin(2)
• Ordinal entry number	Bin(4)
• Buffer location type Hex 01 = Cursor intermediate buffer Hex 02 = Group-by intermediate buffer Hex 03 = Intermediate key buffer	Char(1)
• Reserved	Char(3)
• Number of offending fields	Bin(2)
• Field data (repeated)	Char(6)
– Field number	Bin(2)
– Error type	Char(2)
– Reserved	Char(2)

The ordinal entry number will contain a binary zero value if the exception occurs during inserts of new entries into a data space or the error occurs while processing default values during a join operation.

The ordinal entry number and data space number will contain a value of binary zero when the error occurs during group-by derived field operations.

Instructions Causing Exception:

- Apply Journal Changes
- Copy Data Space Entries
- Data Base Maintenance
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Materialize Cursor Attributes
- Modify Data Space Index Attributes
- Retrieve Data Space Entry
- Retrieve Sequential Data Space Entry
- Set Cursor
- Signal Exception
- Update Data Space Entry

1239 Derived Field Operation Error During Build Index

While creating or rebuilding a data space index, data space entries were encountered which resulted in derived field operational errors. The build, before terminating, found up to 20 instances of these types of errors. The instruction is terminated.

Even though multiple errors may have occurred on a data space entry, only the first occurrence per entry is reported.

Information Passed:

• Data space index (binary zeros if during index creation)	System pointer
• Number of error descriptions	Bin(2)
• Reserved	Char(4)
• Error description (repeated)	Char(14)
– Data space number	Bin(2)
– Ordinal entry number	Bin(4)
– Field number	Bin(2)
– Eror type	Char(2)
– Field location	Char(1)
Hex 03 = Intermediate key buffer	
Unassigned values reserved	
– Reserved	Char(3)

The exception field definitions are the same as for the hex 1238 exception (derived field operation error).

Instructions Causing Exception:

- Create Data Space Index
- Data Base Maintenance
- Activate Cursor (over delayed maintenance index)
- Insert Sequential Data Space Entries
- Signal Exception

14 Event Management

1401 Duplicate Event Monitor

This exception is signaled when identical event monitors (the existing event monitor and the requested event monitor) do not specify event handlers or when the event monitors specify different event handlers.

Information Passed:

• Addressability to the monitor event template	Space pointer

Instructions Causing Exception:

- Monitor Event
- Signal Exception

1402 Event Monitor Not Present

An event monitor with matching event ID, compare value length, and compare value was not found in the executing process.

Information Passed:

• Option indicators	Char(2)
– Compare value content	Bit 0
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 1-15
• Reserved (binary 0)	Char(8)
• Event identification	Char(4)
– Event class	Char(2)
– Event type	Char(1)
– Event subtype	Char(1)
• Compare value length	Bin(2)
• Compare value	Char(32)

Instructions Causing Exception:

- Cancel Event Monitor
- Disable Event Monitor
- Enable Event Monitor
- Test Event
- Wait On Event
- Signal Exception

1403 Machine Event Requires Specification of a Compare Value

The referenced machine event requires use of a compare value.

Instructions Causing Exception:

- Monitor Event
- Signal Exception

1404 Wait On Event Attempted While Masked

The process was masked when the Wait On Event instruction was issued.

Instructions Causing Exception:

- Wait On Event
- Signal Exception

1405 Disable Timer Event Monitor Invalid

An attempt was made to disable an event monitor that is monitoring a timer event.

Instructions Causing Exception:

- Disable Monitor Event
- Signal Exception

1406 Signal Timer Event Monitor Invalid

An attempt was made to signal an event monitor that is monitoring a timer event.

Instructions Causing Exception:

- Signal Event
- Signal Exception

16 Exception Management

1601 Exception Description Status Invalid

The tested exception description was not in the deferred state.

Instructions Causing Exception:

- Test Exception
- Signal Exception

1602 Exception State of Process Invalid

An attempt was made to retrieve exception data or resignal an exception when the process is not in an exception handling state; that is, the process is not in an external program, internal entry point, or branch point exception handler. The re-signal option is valid only for an external exception handler.

Instructions Causing Exception:

- Signal Exception
- Retrieve Exception Data

1603 Invalid Invocation Address

The invocation address specified in the space pointer on a Return From Exception instruction or Signal Exception instruction did not represent an existing program invocation.

Information Passed:

- Space pointer

Instructions Causing Exception:

- Return From Exception
- Sense Exception Description
- Signal Exception

18 Independent Index

1801 Duplicate Key Argument in Index

An attempt was made to insert a key argument that already exists in the index.

Information Passed:

• Independent index	System pointer

Instructions Causing Exception:

- Insert Independent Index Entry
- Signal Exception

1A Lock State

1A01 Invalid Lock State

The lock enforcement rule or rules were violated when an attempt was made to access an object.

Information Passed:

- System pointer to the object

Instructions Causing Exception:

- All instructions that enforce the lock rules
- Signal Exception

1A02 Lock Request Not Grantable

The lock request cannot be granted immediately and neither the synchronous nor asynchronous wait option was specified.

Information Passed:

• Pointer to lock request template	Space pointer
• Failing request number (relative entry position)	Bin(2)

Instructions Causing Exception:

- Lock Object
- Signal Exception

1A03 Invalid Unlock Request

An attempt was made to unlock a lock state not held by the current requesting process.

Information Passed:

• Pointer to unlock request template	Space pointer
• Number of requests not unlocked	Bin(2)
• Request number (relative entry position for each lock not unlocked)	Bin(2)

Instructions Causing Exception:

- Unlock Object
- Signal Exception

1A04 Invalid Object Lock Transfer Request

An attempt was made to transfer locks that were not held by the transferring process, or the transfer lock request was not granted because the lock granting rules would have been violated.

Information Passed:

• Pointer to lock transfer request template	Space pointer
• Number of requests not transferred	Bin(2)
• Request number (relative entry position for each lock not transferred)	Bin(2)

Instructions Causing Exception:

- Transfer Object Lock
- Signal Exception

1A05 Invalid Space Location Unlocked

An attempt was made to unlock a space location lock not held by the current requesting process.

Information Passed:

• Space location process attempted to unlock	Space pointer
• Unlock request	Char(1)

Instructions Causing Exception:

- Unlock Space Location
- Signal Exception

1C Machine-Dependent Exception

1C01 Machine-Dependent Request Invalid

A function requested by an instruction may not be performed because of the current status of the machine or process.

This exception is caused because of one of the following conditions:

- An attempt is made to use an instruction trace while the program event monitor is in use by the service function.
- A contiguous region of 32 K bytes of auxiliary storage cannot be obtained for an access group.

Instruction Causing Exception:

- Machine-dependent

1C02 Program Limitation Exceeded

The program template contained objects or instructions that caused at least one part of the encapsulated program to exceed its machine specification limit.

Information Passed:

• Instruction number (0 is returned in this field if the error code does not apply to a specific instruction)	Bin(2)
• Error code	Char(2)

The error codes and their meanings for the Create Program instruction are as follows:

Error Code	Meaning
0001	The data needed to initialize static areas exceeds 65 535 bytes. This includes storage for IDLs (6 bytes for each entry in an IDL), the values that are the initial values for the static areas, and the logic needed to copy these initial values and to initialize pointers.
0002	The logic needed to initialize automatic areas exceeds 65 535 bytes. This includes the logic needed to copy initial values into automatic storage and to initialize pointers in automatic storage.
0003	Certain internal constants, which are encapsulated into the program and used with specific machine interface instructions, exceed 4096 bytes.

Error Code **Meaning**

0004 The encapsulated form of an instruction requires that the machine address more data items than are supported on one instruction. The particular instruction in error is identified by number in the exception data for this exception. Internal addressability is required for the following types of operands:

- Compound operands, such as those that specify subscripting, substringing, or explicit basing.
- Operands that exist in other than the first 4 K bytes of static or automatic storage.
- Operands that are parameters or based.
- Constant operands for which the encapsulated form exists in other than the first 4 K bytes of the internal program constant area.
- Operands for which the encapsulated form exists in other than the first 4 K bytes of the internal machine work space needed to support the machine interface invocation.

In certain cases, this limitation may apply to the instruction when the perform optimization option has been specified but would not apply when the no optimization option was specified. This is due to differences in the optimization algorithms being employed in each case. To determi whether or not the limitation also applies when no optimization is specified, the program must be recreated with that optio in effect.

0005 The constants that are built into the encapsulated object from the program template exceed 64 K bytes minus 288 bytes. Constants defined in the program template and initial values for automatic storage are included in this area.

Error Code **Meaning**

0006 The work space needed by the machine to support the machine interface invocation for this program exceeds 65 535 bytes (see note).

0007 An instruction required more than the maximum amount of storage allowed for it in the encapsulated program. The particular instruction in error is identified by number in the exception data for this exception:

- Call External, Transfer Control, and Call Internal instructions cannot occupy more than 4800 bytes of storage. For these instructions, passing a large number of arguments or passing arguments with many levels of basing can cause the storage limit to be exceeded.
- All other instructions are limited to a maximum of 1000 bytes. For these instructions, an extensive amount of indirect basing in operand addressability can cause the storage limit to be exceeded.

0008 Encapsulation of the machine interface instruction results in a requirement for more than 1016 K bytes.

0009 The number of items that the machine needs to address exceeds 4040. One addressable item is needed for each of the following:

- Each parameter
- The external parameter list
- Each nonarray pointer
- Each 4096 bytes of static program objects

- Each 4096 bytes of automatic program objects

- Each 4096 bytes of work space needed by the machine to support the machine interface invocation (see note)

The error code and its meaning for the Transfer Control instruction is as follows:

Error Code	Meaning
0006	The work space needed by the machine to support the machine interface invocation for the program that is given control exceeds 65 535 bytes (see note).

Note: The total amount of storage allocated to an invocation of a program excluding storage allocated from the process automatic storage area is 65 200 bytes.

The following objects cause storage to be allocated in the invocation of a program:

Object	**Size (Bytes for Each)**
Operand list	
• Argument	2 + 6 * number of elements
• Parameter (internal)	6 * number of elements
• Parameter (external)	2

Exception descriptions

- Fixed per entry (26 bytes)

- Variable-length per entry (1 + length of compare value if even; plus 1 if odd)

The storage allocated from the PASA is determined by the space that was allocated by the user for PASA use.

Instructions Causing Exception:

- Create Program

- Signal Exception

- Transfer Control

1C03 Machine Storage Limit Exceeded

The storage capacity of the machine was exceeded.

Instructions Causing Exception:

- This exception can be signaled during any machine operation that causes auxiliary storage to be allocated. The operations can include creation or extension of system objects and extension of auxiliary storage for machine overhead supporting the established processes in the machine.

- Signal Exception

1C04 Object Storage Limit Exceeded

The maximum size for an object was exceeded.

The following maximum size limitations are defined for the various system objects. Listed with each object is a specification of the size as well as a definition of the characteristics on which the object size depends. Size is independent of any associated space that is considered elsewhere.

Object	Maximum Size	Size Dependency
Access group	4 MB - 32 K	Access group directory and all objects contained in the access group
Context	16 MB	Context entries including object identification and address
Controller description	4 K	Controller description definition as well as relationships to LUDs and NDs
Cursor	64 K	Cursor definition including entry mappings
Data space	16 x 16 MB	Data space definition, information for each data space index, and data space entries
Data space index	16 MB + 64 K	Data space index and key information for every data space in the index, and the size of the user exit routine for select/omit
Index	16 MB	Number and size of index entries
Logical unit description	4 K	Logical unit description definition and relationship to other source/sink objects
Network description	4 K	Network description definition and relationship to other source/sink objects
Program	20 MB	Program definition including object definitions, instruction stream, initial values, and size of the program template
Queue	16 MB	Queue definition plus entries enqueued to queue
Space	16 MB	Space definition and its associated space
User profile	16 MB	User profile definition and entries for owned and authorized objects

The maximum size of the space in a system object depends on the size and packaging of the system object.

The maximum space size ensures that a space less than or equal to this size may always be allocated with the object. Fixed-length spaces can always have this size. Variable-length spaces can always be extended to at least this size. This value is independent of the object's size.

The following is a list of the guaranteed maximum sizes of associated space for various system objects:

Object	Guaranteed Maximum Space
Access group	Initial allocation
Context	16 MB - 32 B
Controller description	16 MB - 4 K
Cursor	16 MB - 64 K
Data space	16 MB - 32 B
Data space index	16 MB - 64 K - 32 B
Index	16 MB - 32 B
Logical unit description	16 MB - 4 K
Network description	16 MB - 4 K
Program	16 MB - 32 K
Queue	16 MB - 64 K
Space	16 MB - 160 B
User profile	16 MB - 32 B

Information Passed:

• Return object pointer (binary 0 if the object is being created)	System pointer

Instructions Causing Exception:

- All Create instructions
- All instructions that cause additional storage to be allocated for an object
- Signal Exception

1C06 Machine Lock Limit Exceeded

The maximum number of concurrently held locks was exceeded.

No more than 57,344 locks, implicit locks, and internal locks required for machine operation may exist at any one time.

Instructions Causing Exception:

- Activate Cursor
- Apply Journaled Changes
- Create Cursor
- Create Data Space Index
- Data Base Maintenance
- Dequeue
- Initiate Process
- Lock Object
- Modify Process Attributes
- Set Cursor
- Signal Exception
- Any instruction that acquires an implicit lock or an internal machine lock

1C07 Modify Main Storage Pool Controls Invalid

An attempt was made to modify a main storage pool to a size smaller than the machine required minimum size.

Information Passed:

• Number of storage pools in error	Bin(2)
• Error description (repeated for each error)	Char(10)
– Storage pool ID	Bin(2)
– Reserved (binary 0)	Bin(8)

Instructions Causing Exception:

- Modify Resource Management Controls
- Signal Exception

1C08 Requested Function Not Valid

The requested function is not valid or is not avaiable on the machine.

Information Passed:

• Failure explanation Hex 00 = Function requested which requires use of S/38 support of data encryption algorithm RPQ (S40270) and the RPQ is not on the system.	Char(1)
• Reserved	Char(10)

Instructions Causing Exception:

- Cipher
- Cipher Key
- Signal Exception

1E Machine Observation

1E01 Program Not Observable

The program observation functions were destroyed for the program referenced by the executing instruction.

Information Passed:

- Program System pointer

Instruction Causing Exception:

- Materialize Instruction Attributes
- Materialize Invocation
- Signal Exception

20 Machine Support

2001 Diagnose

An error or discrepancy was found when a Diagnose instruction was processed.

Information Passed:

- Space element to the subelement in the operand 2 object that was being processed
- Data Bin(4)
  - Subidentifier unique to the requested function Bin(2)
  - Indicator of the pointer in operand 2 that was being processed Bin(2)

Instructions Causing Exception:

- Diagnose
- Signal Exception

2002 Machine Check

A machine malfunction affecting system-wide operation has been detected during execution of an instruction in this process.

Information Passed:

- Timestamp that gives the current value of the machine time-of-day clock. Char(8)
- Error code indicating nature of machine check. (This value is machine-dependent and is only defined in the machine service documentation.) Char(2)
- Reserved (binary 0) Char(6)
- VLOG dump ID Char(8)
- Error class Bin(2)

  The error class codes for the type of damage detected are as follows:

  Hex 0000 = Unspecified abnormal condition
  Hex 0002 = Logically invalid device sector
  Hex 0003 = Device failure

- Auxiliary storage device indicator Bin(2)

  This field is defined for error classes hex 0002 and hex 0003. It is the OU number of the failing device or 0 for a main storage failure.

- Reserved (binary 0) Char(100)

Instructions Causing Exception:

- Any instruction
- Signal Exception

2003 Function Check

The executing instruction has failed unexpectedly during execution within the process.

Information Passed:

- Timestamp giving the current value of the machine time-of-day clock. Char(8)
- Error code indicating the nature of the function check. (This value is machine-dependent.) Char(2)
- Reserved (binary 0) Char(6)
- VLOG dump ID Char(8)
- Error class Bin(2)

  The error class codes for the type of damage detected are as follows:

  Hex 0000 = Unspecified abnormal condition
  Hex 0002 = Logically invalid device sector
  Hex 0003 = Device failure

- Auxiliary storage device indicator Bin(2)

  This field is defined for error classes hex 0002 and hex 0003. It is the OU number of the failing device or 0 for a main storage failure.

- Reserved (binary 0) Char(100)

Instructions Causing Exception:

- Any instruction
- Signal Exception

22 Object Access

2201 Object Not Found

An attempt to resolve addressability into a system pointer was not successful for one of the following reasons:

- The named object was not located in the context specified in the symbolic address or in any context referenced in the name resolution list.
- An object with a corresponding name was found but the user profile(s) governing execution of the instruction did not have the authority required for resolution.

Information Passed:

- Object identification Char(32)
  - Object type Char(1)
  - Object subtype Char(1)
  - Object name Char(30)
- Required authorization Char(2)

Instructions Causing Exception:

- Any instruction that references an object through a system pointer
- Signal Exception

2202 Object Destroyed

An attempt was made to reference an object that no longer exists.

Instructions Causing Exception:

- Any instruction that references an object through a system pointer, a space pointer, or a data pointer

- Any instruction that references a scalar or a pointer operand when the object and the space containing the scalar or pointer have been destroyed

- Signal Exception

2203 Object Suspended

An attempt was made to reference an object that is in suspended state and, with its contents truncated, is not suitable for processing.

Information Passed:

• Object	System pointer

Instructions Causing Exception:

- Instructions that reference space, queue, index, data space, or data space index objects, except for the following instructions:
  - Resolve System Pointer (to the target object)
  - Grant Authority
  - Retract Authority
  - Transfer Ownership
  - Modify Addressability
  - Lock Object
  - Unlock Object
  - Transfer Object Lock
  - Request I/O (for load/dump)
  - Materialize Object Lock
  - All Destroy instructions
  - Create Data Space Index (allows suspended data space only)
  - Materialize System Object
  - Materialize Pointer

- Signal Exception

2204 Object Not Eligible for Operation

An attempt to reference an object was unsuccessful because the object was not eligible for the operation requested for one of the following reasons:

- An object that cannot be duplicated was specified on a Create Duplicate Object instruction.

- A data space or data space index was activated when a Suspend instruction or a load/dump operation was attempted.

- An index that can contain pointers was referenced by a Suspend Object instruction or was referenced for a load/dump operation.

- An attempt was made to activate a cursor that is already activated to this process or is activated to another process.

- A temporary object was referenced on a Transfer Ownership instruction.

- An attempt was made to replace a program through a load operation.

- An attempt was made to materialize cursor statistics when the cursor was not active for this process.

- The receiving data space for a Copy Data Space Entries instruction is active under more than one cursor.

- An attempt was made to modify (through the Modify Data Space Index Attributes instruction) a data space index when the current attributes of the data space index where incompatible with the attempted modification.

- An attempt was made to retrieve (through the Retrieve Data Space Entry instruction) a data space entry when the cursor position is the result of group-by processing.

- An attempt was made to set a cursor (by a Set Cursor or a Retrieve Sequential Data Space Entries instruction) in an event handler while the cursor is waiting for a lock for another Set Cursor instruction. This can happen when an event is handled during a data base entry lock wait.

- The source or receiver cursor has had a set cursor operation or another operation performed on it that left the cursor set to a data space entry after the cursor was activated and before a Copy Data Space Entries instruction was issued.

- An attempt was made to apply journal changes to an object for which journaling is not supported.

- An attempt was made to apply journal changes to a data space that had a cursor active over it.

Information Passed:

- System pointer to the object

Instructions Causing Exception:

- Activate Cursor

- Apply Journaled Changes

- Commit

- Copy Data Space Entries

- Create Duplicate Object

- Data Base Maintenance (all options)

- Decommit

- Delete Object From Access Group

- De-activate Cursor

- Ensure Data Space Entries

- Journalize Object

- Materialize Commit Block Attributes

- Materialize Cursor Attribute

- Modify Commit Block

- Modify Data Space Index Attributes

- Modify Journal Port

- Request I/O

- Retrieve Data Space Entry

- Retrieve Sequential Data Space Entries

- Set Cursor

- Signal Exception

2205 Object Not Available to Process

An attempt to reference an object was unsuccessful because it was restricted, temporarily or permanently, to another process for one of the following reasons:

- An active cursor was restricted to the process that activated it.
- Application of implicit locks failed.
- An attempt was made to reference a dump space through an Insert Dump Data, a Modify Dump Space, a Retrieve Dump Data, a Request I/O, or a Request Path Operation instruction when the dump space was in use by a Request Path Operation or Request I/O instruction load or dump function.

Information Passed:

- System pointer to the object

Instructions Causing Exception:

- Activate Cursor
- Apply Journaled Changes
- Create Data Space Index
- Data Base Maintenance
- De-activate Cursor
- Delete Data Space Entry
- Ensure Data Space Entries
- Initiate Process
- Insert Data Space Entry
- Insert Dump Data
- Modify Data Space Index Attributes
- Modify Dump Space
- Release Data Space Entries
- Request I/O
- Request Path Operation
- Retrieve Data Space Entry
- Retrieve Dump Data
- Set Cursor
- Update Data Space Entry
- Signal Exception

2206 Object Not Eligible for Destruction

An attempt to destroy an object cannot be processed because one of the following conditions exists within that object:

- A Destroy User Profile instruction refers to a user profile that still owns objects or has a process currently initiated for it.
- A Destroy Data Space instruction refers to a data space that is being used through a cursor.
- A Destroy Data Space Index instruction refers to a data space index that is being used through a cursor.
- A Destroy Access Group instruction refers to an access group that contains one or more objects.
- A Destroy Dump Space instruction refers to a dump space which is in use by a Request Path Operation or Request I/O instruction load or dump function.

Information Passed:

- System pointer to the object

Instructions Causing Exception:

- Destroy Access Group
- Destroy Data Space
- Destroy Data Space Index
- Destroy Dump Space
- Destroy Journal Port
- Destroy Journal Space
- Destroy Process Control Space
- Destroy User Profile
- Signal Exception

24 Pointer Specification

2401 Pointer Does Not Exist

A pointer reference was made to a storage location in a space that does not contain a pointer data object, or a reference was made to a space pointer machine object that was not set to address a space.

Instructions Causing Exception:

- Any instruction that has pointer operands
- Any instruction that references a base operand (scalar or pointer) when the base pointer is not a space pointer
- Any instruction that allows a scalar defined by a data pointer to be an operand
- Any instruction that requires a pointer as part of the input template
- Signal Exception

2402 Pointer Type Invalid

An instruction has referenced a pointer object that contains an incorrect pointer type for the operation requested.

Instructions Causing Exception:

- Any instruction that has pointer operands
- Any instruction that contains a base operand (scalar or pointer) when the base pointer is not a space pointer
- Any instruction that allows a scalar defined by a data pointer to be an operand
- Any instruction that requires a pointer as part of the input template
- Signal Exception

2403 Pointer Addressing Invalid Object

An instruction has referenced a system pointer that addresses an incorrect type of system object for this operation.

Information Passed:

- The invalid system pointer

Instructions Causing Exception:

- Any instruction that references a system pointer, either as an operand or within a template operand, and that requires a specific object type as a part of its operation
- Signal Exception

2404 Pointer Not Resolved

The operation did not find a resolved system pointer. For example, NRL (name resolution list) entries must be resolved system pointers that address contents.

Information Passed:

- The invalid pointer

Instructions Causing Exception:

- Resolve System Pointer
- Any instruction that causes a system pointer to be implicitly resolved when the NRL is used in the resolution. All entries in the NRL must be resolved.
- Resolved Data Pointer
- Any instruction that causes a data pointer to be implicitly resolved. All activation entries in the process must contain a resolved pointer to the associated program.
- Signal Exception

26 Process Management

2602 Queue Full

An attempt was made to enqueue a message to a queue that is full and is not extendable.

Information Passed:

- System pointer to the queue for which the enqueue was attempted

Instructions Causing Exception:

- Enqueue
- Request I/O
- Signal Exception

28 Process State

2801 Process Ineligible for Operation

An attempt was made by a subordinate process to terminate a superordinate process.

Information Passed:

- Process control space system pointer to the process to be terminated.

Instructions Causing Exception:

- Terminate Process
- Signal Exception

2802 Process Control Space Not Associated with a Process

The process control space system pointer referenced a process control space that was not currently associated with an existing process.

Information Passed:

• Process control space	System pointer

Instructions Causing Exception:

- Materialize Invocation Stack
- Materialize Process Attributes
- Materialize Process Locks
- Materialize Selected Locks
- Modify Process Attributes
- Resume Process
- Suspend Process
- Terminate Process
- Transfer Object Lock
- Signal Event
- Signal Exception

280A Process Attribute Modification Invalid

The modification control indicators for a process did not allow the process to modify this attribute.

Information Passed:

• System pointer to the process control space	
• Modification control indicators (bit significant)	Char(8)
• Modify attribute (bit significant)	Char(1)

Instructions Causing Exception:

- Modify Process Attributes
- Signal Exception

2A Program Creation

2A01 Program Header Invalid

The data in the program header was invalid.

Instructions Causing Exception:

- Create Program
- Signal Exception

2A02 ODT Syntax Error

The syntax (bit setting) of an ODT (object definition table) entry was invalid.

Information Passed:

• ODT entry number	Char(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A03 ODT Relational Error

An ODT (object definition table) entry reference to another ODT entry was invalid.

Information Passed:

• ODT entry number	Char(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A04 Operation Code Invalid

One of the following conditions occurred.

- The operation code did not exist.
- The optional form was not allowed.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A05 Invalid Op Code Extender Field

The branch/indicator options were invalid.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A06 Invalid Operand Type

One of the following conditions was detected:

- An operand was not the required type (signed immediate, immediate, constant data object, scalar data object, pointer data object, null, branch point, or instruction definition list).
- An operand was described as an immediate or constant data object. However, the instruction specifies that the operand be modified to something other than an immediate or constant data object, or the instruction does not allow an immediate or constant data object operand.
- The operand type specified is not a valid operand type.
- The type of one operand does not satisfy a required relationship with the type of another operand.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A07 Invalid Operand Attribute

One of the following conditions was detected:

- An operand did not have the attributes required by the instruction (character, packed decimal, zoned decimal, binary, floating-point, scalar, array, assumed, overlay, restricted, open, based, explicitly based).
- The attributes of one operand did not match the required attributes of another operand.
- At least one operand in the argument list for a Transfer Control instruction was specified as automatic.
- The receiver for an instruction specified with the optional round form has the floating-point attribute.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A08 Invalid Operand Value Range

One of the following conditions was detected:

- An operand was a constant or immediate data object and was used as an index into an array or indicated a position in a character string, but it was outside the range of the array or character string.

- An operand was a constant or immediate data object and did not conform to the value required by the instruction.

- The operand immediate value is outside of the accepted range. The valid range for an unsigned immediate value is equal to or greater than 0 and less than or equal to 8191. The valid range for a signed immediate value is from negative 4096 through a positive 4095.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program

- Signal Exception

2A09 Invalid Branch Target Operand

One of the following conditions was detected:

- An operand was not an instruction pointer, branch point, instruction number, or relative instruction number.

- An operand was an instruction number or relative instruction number but was outside the range of the program.

- A branch target operand identified an instruction that was not indicated as a branch target.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program

- Signal Exception

2A0A Invalid Operand Length

One of the following conditions was detected:

- The length attribute of an operand was not greater than or equal to the length required by the instruction.
- The length attribute of an operand was invalid based on its relationship to the length attribute of another operand in the same instruction.
- The length attribute of a decimal operand exceeds 15 digits when specified in conjunction with a floating-point operand.
- A decimal operand has an invalid integer or fractional digit length in relationship to that required by the instruction.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A0B Invalid Number of Operands

The number of arguments in a Call Internal instruction was not equal to the number of parameters in the called entry point.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A0C Invalid Operand ODT Reference

The ODT reference was not within the range of the ODV.

Information Passed:

• Instruction number of the instruction being analyzed	Bin(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2A0D Reserved Bits Are Not Zero

The reserved bits in an opcode or operand are nonzero.

Information Passed:

• Instruction number of the instruction being processed	Bin(2)

Instructions Causing Exception:

- Create Program
- Signal Exception

2C Program Execution

2C01 Return Instruction Invalid

This exception was improper usage of the Return, Transfer Control, or Return From Exception instruction for one of the following reasons:

- A Return From Exception instruction was executed in an invocation that was not defined as an exception handler.
- A Return External or Transfer Control instruction was issued from a first-invocation-level exception handler.
- A Transfer Control instruction was issued from a first-invocation-level event handler.

Instructions Causing Exception:

- Return External
- Return From Exception
- Transfer Control
- Signal Exception

2C02 Return Point Invalid

An attempt was made to use a Return External instruction with a return point that was invalid for one of the following reasons:

- The return point value was outside the range of the return list specified on the preceding Call External instruction.
- A nonzero return point was supplied, but no return list was supplied on the preceding Call External instruction.
- A nonzero return point was supplied when a Return External instruction was issued in the first invocation in the process.
- A nonzero return point was supplied when the Return External instruction was issued by an invocation acting as an event handler.

Instructions Causing Exception:

- Return External
- Signal Exception

2C03 Stack Control Invalid

Information Passed:

• Cause indicator		Bin(2)
– Hex 0003 =	The chain being modified bit in the PSSA base entry was on when it was necessary for the machine to use the chain of PSSA activations or it was necessary for the machine to modify the chain of PSSA activations.	

Instructions Causing Exception:

- Activate Program
- Call External
- De-activate Program
- Modify Automatic Storage Allocation
- Resolve Data Pointer
- Transfer Control
- Signal Exception

2C04 Branch Target Invalid

An attempt was made to branch to an instruction defined through an instruction pointer, but the instruction pointer was set by a program other than the one that issued the branch.

Information Passed:

- Instruction pointer causing the exception

Instructions Causing Exception:

- All instructions that have a branch form
- Signal Exception

2C05 Activation in Use by Invocation

An attempt was made to de-activate a program that has an existing invocation which is not the invocation issuing the instruction.

Information Passed:

• Program	System pointer

Instructions Causing Exception:

- De-activate Program
- Signal Exception

2C06 Instruction Cancellation

This exception is raised when an instruction is canceled due to a process termination. Because the process is about to be terminated, only an internal handler is able to field this exception before the process is terminated. No guarantee is given as to how far the internal handler is able to proceed before the process is terminated.

Instructions Causing Exception:

- Activate Cursor
- Apply Journaled Changes
- Copy Data Space Entries
- Create Data Space Index
- Create Program
- Data Base Maintenance
- Insert Sequential Data Space Entries
- Modify Data Space Index Attributes
- Request I/O
- Retrieve Journal Space Entries
- Retrieve Sequential Data Space Entries
- Set Cursor
- Signal Exception

2C07 Instruction Termination

This exception is signaled when an instruction is terminated due to execution of a Terminate Instruction instruction specifying a process currently executing an instruction which is subject to termination. Certain instructions which require a relatively long period of time for their execution are subject to being terminated in this manner. Refer to the definition of Terminate Instruction for the list of applicable instructions. The signaling of this exception does not indicate a pending process termination. Normal exception handler processing can be performed for this exception.

Information Passed:

- Reserved (binary 0) Char(16)

Instructions Causing Exception:

- Activate Cursor
- Apply Journaled Changed
- Copy Data Space Entries
- Create Data Space Index
- Create Program
- Data Base Maintenance
- Insert Sequential Data Space Entries
- Modify Data Space Index Attributes
- Request I/O
- Retrieve Journal Entries
- Retrieve Sequential Data Space Entries
- Set Cursor
- Signal Exception

2E Resource Control Limit

2E01 User Profile Storage Limit Exceeded

The user profile specified insufficient auxiliary storage to create or extend a permanent object.

Information Passed:

- System pointer to the user profile

Instructions Causing Exception:

- All create instructions creating a permanent object
- All instructions extending a permanent object
- Signal Exception
- Transfer Ownership

30 Journal Management

3001 Apply Journal Changes Failure

An exception condition was encountered while the system was trying to apply a change contained on the indicated journal space to the indicated object. The journal sequence number of the change that could not be made and the type of condition encountered are indicated in the exception data.

Information Passed:

• Journal space	System pointer
• Object being changed	System pointer
• Journal sequence number	Bin(4)
• Exception condition	Char(2)
– Object type	Char(1)
– Condition code	Char(1)
Hex 01=Data space entry for update did not exist.	
Hex 02=Data space entry for update of deleted entry was not deleted in the data space.	
Hex 03=Data space entry for insert was not next available data space entry in the data space.	
Hex 04=Data space entry for delete did not exist in the data space.	
Hex 05=Data space entry for delete of a deleted entry was not already deleted in the data space.	
Hex 08=A before image or entry was required to perform the apply but was not contained on the journal.	
Hex 09=The journal entry describes a valid change to an object whose changes are being reapplied, but Apply Journal Changes instruction does not support reapplying through the change described by the indicated entry.	
Hex 0A=Data space entry limit was exceeded.	

Instructions Causing Exception:

- Apply Journal Changes
- Signal Exception

3002 Entry Not Journaled

An operation could not be performed because a journal entry could not be placed on the journal space of the indicated journal port. The condition described in the exception data identifies the reason for the failure.

Information Passed:

• Journal port	System pointer
• Journal space	System pointer or null

- Return value
  - Hex 01=No journal space was attached to the specified journal port.
  - Hex 02=Unable to obtain sufficient space on the journal space for the entry.
  - Hex 03=Journal port is damaged.
  - Hex 04=All journal spaces attached to the journal port are damaged.
  - Hex 05=The journal sequence number has reached its maximum allowed value.
  - Hex 06=Journal failure.

Instructions Causing Exception:

- Any instruction that causes journal entries to be created and placed on the journal receiver
- Signal Exception

3003 Maximum Objects Through a Journal Port Limit Exceeded

The identified journal port is already journaling changes to the maximum number of objects.

Information Passed:

• Journal port	System pointer
• Object attempted to journal	System pointer

Instructions Causing Exception:

- Journal Object
- Signal Exception

3004 Invalid Journal Space

The identified journal space was provided as input to one of the indicated instructions and was not in a valid position in the journal space list. The exceptional condition is identified in the exception data.

Information Passed:

• Journaled space	System pointer
• Error condition indicator – Hex 01=Journal sequence numbers not ascending – Hex 02=Journal sequence numbers missing – Hex 03=Journal space contains no journal entries – Hex 04=Journal entry prefix length does not match	Char(1)

Instructions Causing Exception:

- Apply Journaled Changes
- Retrieve Journal Entries
- Signal Exception

3005 Invalid Selection/Transaction List Entry

A selection/transaction list entry passed to the Apply Journaled Changes instruction contains invalid values.

Information Passed:

• Template that addresses the selection/transaction list with the entry in error	Space pointer
• Selection/transaction list	Space pointer
• Entry in the selection/transaction list	Bin(4)

Instructions Causing Exception:

- Apply Journaled Changes
- Signal Exception

3006 Journal Space Not at a Recoverable Boundary

An operation was attempted on a journal space or journal port that required all object changes corresponding to journal entries contained on the indicated journal space to be at a recoverable boundary.

Information Passed:

• Journal port	System pointer
• Number of journal spaces	Bin(2)
• Reserved (binary 0)	Char(14)
• Journal space (1 to n)	System pointer

Instructions Causing Exception:

- Destroy Journal Space
- Modify Journal Port
- Suspend Object
- Request I/O (request for a load operation)
- Signal Exception

3007 Journal ID Not Unique

An operation was attempted that required unique journal IDs and a duplicate journal ID was encountered.

Information Passed:

• Journal port (binary 0 if apply journaled entries failure)	System pointer
• Object being journaled	System pointer
• Duplicate object	System pointer
• Journal ID of duplicates	Char(10)

Instructions Causing Exception:

- Apply Journaled Changes
- Journal Object
- Signal Exception

3008 Object Already Being Journaled

An object was attempted to be journaled that was already being journaled. The exception data indicates through which journal port the object is currently being journaled.

Information Passed:

• Object being journaled	System pointer
• Journal port	System pointer
• Journal ID	Char(10)

Instructions Causing Exception:

- Journal Object
- Signal Exception

3009 Transaction List Limit Reached

The maximum number of transactions (defined in the apply options template) has been reached.

Information Passed:

• Number of entries in the transaction list	Bin(4)

Instructions Causing Exception:

- Apply Journaled Changes
- Signal Exception

32 Scalar Specification

3201 Scalar Type Invalid

A scalar operand did not have the following data types required by the instruction:

- Character
- Packed decimal
- Zoned decimal
- Binary
- Floating-point

Instructions Causing Exception:

- Any instruction using a late bound (data pointer) scalar operand
- Signal Exception

3202 Scalar Attributes Invalid

A scalar operand did not have the following attributes required by the instruction:

- Length
- Precision
- Boundary

Instructions Causing Exception:

- Any instruction using a late-bound (data pointer) scalar operand
- Any instruction that verifies the length of a character scalar in a space object operand
- Signal Exception

3203 Scalar Value Invalid

A character scalar operand does not contain a correct value as required by the instruction.

Information Passed:

• Length of data passed	Bin(2)
• Bit offset to invalid field (relative to 0)	Bin(2)
• Operand number	Bin(2)
• Invalid data	Char(*)

Instructions Causing Exception:

- Any instruction using a scalar operand
- Signal Exception

34 Source/Sink Management

3401 Source/Sink Configuration Invalid

A source/sink object associated with a source/sink create, modify or request I/O instruction was not properly configured to allow the requested operation.

Information Passed:

• System pointer to the object that prevented execution from completing, if appropriate, otherwise; binary zero.	
• Exception data–A defect code that provides a further definition of the cause of the exception as follows (bit significant):	Char(2)

Defect Code (Hex)	Instruction	Meaning
1101	CRTND	Backward object supplied is of the wrong source/sink object subtype.
1102	CRTND	Backward object supplied is already connected to another forward object.
1103	CRTND	Duplicate backward pointers supplied.
1104	CRTND	Backward object does not have attributes that match this ND: • CD cannot be a switched CD. • Role indicator must indicate an opposite role to that of this ND.
1105	CRTND	Invalid object supplied in eligibility list.
1106	CRTND	Backward objects have duplicate physical addresses.
1107	CRTND	Attempting to create an ND that will exceed the maximum allowable number of NDs for this OU number.
1201	CRTCD	Backward or forward object supplied is of the wrong source/sink object subtype.
1202	CRTCD	Backward object supplied is already connected to another forward object.
1203	CRTCD	Duplicate backward pointers supplied.
1204	CRTCD	Backward or forward object does not have attributes that match this CD. Forward object (ND) checks: • ND cannot be a switched ND. • Role indicator must indicate an opposite role to that of this CD. • If the ND is a primary point to point configuration that already has one CD attached.
1205	CRTCD	Backward objects have duplicate physical addresses.
1206	CRTCD	Invalid ND candidate (not switched, wrong ND type or wrong line discipline).
1207	CRTCD	Backward objects supplied have duplicate LU names.
1208	CRTCD	Backward object has an address that is partly out of range for this controller.
1209	CRTCD	Backward object has specific characteristics that are out of range for this controller.
1301	CRTLUD	Forward object supplied is of the wrong source/sink object subtype.
1304	CRTLUD	Invalid forward object attributes.
1307	CRTLUD	Forward object supplied already has an object with duplicate LU names.
1308	CRTLUD	Forward object supplied is of a type not compatible with the address range of this device.
1309	CRTLUD	Forward object supplied is of a type not compatible with the specific characteristics of this device.

Defect Code (Hex)	Instruction	Meaning
2101	MODND	Status change attempted with no CDs or LUDs attached to this ND (nonswitched).
2103	MODND	Status change attempted to enable dial in with no valid eligibility objects.
2104	MODND	Invalid object supplied in eligibility list.
2105	MODND	ASCII translation–Invalid address characters.
2201	MODCD	Status change attempted with no LUDs attached.
2202	MODCD	Status change attempted with no valid forward pointer (nonswitched CD).
2203	MODCD	Dial attempted with no valid ND candidate list entries.
2205	MODCD	Invalid ND candidate (not switched, wrong ND type or wrong line discipline).
2206	MODCD	ASCII translation–Invalid address character.
2301	MODLUD	Status change attempted with no valid forward pointer.
2302	MODLUD	ASCII translation–Invalid address character.
2307	MODLUD	Change to device specific area would result in duplicate LU names.
3401	Request I/O	Request I/O or request path operation response queue or the LUD object does not have proper attributes for the Request I/O instruction.
3402	Request I/O	Object specified in the SSR does not support synchronous Request I/O instructions.
3403	Request I/O	LUD specified has invalid value in device specific area.
5307	MODMATR	Modify system name would result in duplicate LU names.

Instructions Causing Exception:

- Create Controller Description
- Create Logical Unit Description
- Create Network Description
- Materialize Controller Description
- Materialize Network Description
- Modify Controller Description
- Modify Logical Unit Description
- Modify Machine Attributes
- Modify Network Description
- Request I/O
- Signal Exception

3402 Source/Sink Physical Address Invalid

An attempt was made to create a source/sink object with the same physical address and exchange identification as an already existing object of the same type, or the physical address has a component part that does not match the physical address of the related forward or backward object specified.

The duplicate address exception data (hex 01) is not signaled for the creation of CD objects for BSC controllers or for the creation of CD objects for the host system when these CD objects have the switched line or the switched backup attribute.

Information Passed:

- System pointer to the object preventing execution of this instruction

• Exception data (bit significant)	Char(1)
– Hex 01 = Duplicate address	
– Hex 02 = Related object address mismatch	

Instructions Causing Exception:

- Create Controller Description
- Create Logical Unit Description
- Create Network Description
- Signal Exception

3403 Source/Sink Object State Invalid

The source/sink object associated with a source/sink create, destroy, modify, or request I/O instruction was not in the proper state or proper mode to allow execution of the instruction to complete successfully.

Information Passed:

• Object preventing execution of the instruction	System pointer
• Exception data (bit significant)	Char(16)
– Affected element within the source/sink object (bit significant)	Char(2)
– Source/Sink object status field for the object that prevented execution of the instruction (bit significant)	Char(8)
– Reserved	Char(6)
• Primary object for the instruction (on a create, this entry is binary 0)	System pointer
• Template for the instruction (binary 0 if not applicable)	Space pointer

The following chart shows the elements that can be indicated in the exception data.

	Instruction									
	Create			Destroy			Modify			Request
Element	**ND**	**CD**	**LUD**	**ND**	**CD**	**LUD**	**ND**	**CD**	**LUD**	**I/O**
ND Status		X	X	X	X	X	X	X	X	
CD Status			X		X	X		X	X	
LUD Status						X			X	X
Other ND Elements							X			
Other CD Elements								X		
Other LUD Elements									X	X

Instructions Causing Exception:

- Create Controller Description
- Create Logical Unit Description
- Create Network Description
- Destroy Controller Description
- Destroy Logical Unit Description
- Destroy Network Description
- Modify Controller Description
- Modify Logical Unit Description
- Modify Network Description
- Request I/O
- Signal Exception

3404 Source/Sink Resource Not Available

An attempt was made to create a source/sink object, but physical hardware or system support for this hardware does not exist; or an attempt was made to modify a source/sink object, but hardware sequences cannot be completed successfully.

Information Passed:

- A system pointer that identifies the object that caused the instruction termination, if appropriate, otherwise; binary zero

• Exception data (bit significant)	Char(4)
– Generic error code	Char(2)
– Device-specific error code	Char(2)

The exception data consists of two 2-byte return codes that define the cause of this exception. The first 2-byte field provides a generic error code that is common to all source/sink objects of that code, and the second 2 bytes provide further device-specific error code for the device in question. The following list defines the generic error codes that can be presented by this exception. The generic error code values are formatted as hex jknn, where:

j = 1	Indicates a create instruction
j = 2	Indicates a modify instruction
k = 1	Indicates an ND object
k = 2	Indicates a CD object
k = 3	Indicates an LUD object
nn	Indicates the generic error code that provides further definition of the cause of the exception as follows:

Code (Hex)	Instruction	Meaning
1101	Create ND	ND hardware not installed
1201	Create CD	CD hardware not installed
1301	Create LUD	LUD hardware not installed

(The above error codes indicate that the object creation being attempted, although potentially valid on some system, does not agree with the hardware or support attributes currently configured on this system.)

Code (Hex)	Instruction	Meaning
2101	Modify ND	Set diagnostic mode failure
2102	Modify ND	Vary on failure
2103	Modify ND	Manual answer failure
2105	Modify ND	Enable failure
2108	Modify ND	Continue failure
2109	Modify ND	Cancel failure
2202	Modify CD	Vary on failure
2204	Modify CD	Dial out failure
2206	Modify CD	Power on failure
2207	Modify CD	Power off failure
2208	Modify CD	Continue failure
2209	Modify CD	Cancel failure
2300	Modify LUD	Other-than-status element failure
2302	Modify LUD	Vary on failure
2303	Modify LUD	Activate failure
2306	Modify LUD	Power on failure
2307	Modify LUD	Power off failure
2308	Modify LUD	Continue failure
2309	Modify LUD	Cancel failure
2311	Modify LUD	Resume failure
2312	Modify LUD	Suspend failure
2313	Modify LUD	Quiesce failure
3401	Request I/O	Request I/O failure

Instructions Causing Exception:

- Create Controller Description
- Create Logical Unit Exception
- Create Network Description
- Modify Controller Description
- Modify Logical Unit Description
- Modify Network Description
- Signal Exception

36 Space Management

3601 Space Extension/Truncation

A Modify Space Attributes instruction made one of the following invalid attempts to modify the size of the space:

- Truncate the space to a negative size.
- Extend or truncate a fixed size space.
- Extend a space beyond the space allowed in the referenced object.

Information Passed:

- System pointer to the space

Instructions Causing Exception:

- Activate Program
- Call External
- Modify Automatic Storage Allocation
- Modify Space Attributes
- Signal Exception
- Transfer Control
- Any instruction that invokes an external exception handler or an external event handler or an invocation exit

3602 Invalid Space Modification

A Modify Space Attributes instruction made an attempt to modify the attributes of a space but the requested modification is invalid.

Information Passed:

- System pointer to the object
- Error code Char(2)

  Error codes and their meanings are as follows:

  Code **Meaning**

  0001 An attempt was made to modify the performance class attribute of the system object containing the space and the space was not a fixed length of size zero.

  0002 An attempt was made to modify a system object to or from the state of having a fixed length space of size zero and the operation is invalid for that type of system object.

Instructions Causing Exception:

- Modify Space Attributes
- Signal Exception

38 Template Specification

3801 Template Value Invalid

A template did not contain a correct value required by the instruction.

Information Passed:

• Addressability to the template	Space pointer
• Offset to invalid field (leftmost byte) in bytes (A value of 0 is the first byte in the template. An invalid field is considered to be the lowest-level character or numeric template entry that contains the information that is in error.)	Bin(2)
• Bit offset in invalid field field or 0 (A 0 value indicates the leftmost bit in the invalid field.)	Bin(2)
• The number of bytes in the invalid field	Bin(2)
• Instruction operand number (The first operand in an instruction is 1.)	Bin(2)

Instructions Causing Exception:

- Any instruction that has a space pointer as a source operand
- Convert BSC to Character
- Convert Character to BSC
- Convert Character to MRJE
- Convert MRJE to Character
- Signal Exception
- Scan with Control

3802 Template Size Invalid

A source template was not large enough for this instruction.

Information Passed:

• Addressability to the template	Space pointer

Instructions Causing Exception:

- Any instruction that has a space pointer that addresses a source template operand
- Signal Exception

3803 Materialization Length Exception

Less than 8 bytes was specified to be available in the receiver operand of a materialize instruction.

Instructions Causing Exception:

- Any materialize instruction
- Any retrieve instruction
- Signal Exception

3A Wait Time-Out

3A01 Dequeue

A specified time period elapsed, and a Dequeue instruction was not satisfied.

Information Passed:

• The queue waited for	System pointer
• Time-out value	Char (8)

Instructions Causing Exception:

- Dequeue
- Signal Exception

3A02 Lock

A specified time period elapsed, and a Lock Object instruction was not satisfied.

Information Passed:

• System pointer to the object waited for	
• Time-out value	Char(8)

Instructions Causing Exception:

- Lock Object
- Signal Exception

3A03 Event

A specified time period elapsed, and a Wait On Event instruction was not satisfied.

Information Passed:

• Number of event monitors	Bin(2)
• Time-out value	Char(8)
• Template from operand 2 of the Wait On Event instruction and repeated for each number of event monitors (0's when number of event monitors is 0)	Char(48)

Instructions Causing Exception:

- Wait On Event
- Signal Exception

3A04 Space Location Lock Wait

A specified time period has elapsed and a Lock Space Location instruction has not been satisfied.

Information Passed:

• Space location	Space pointer
• Time-out value	Char(8)

Instructions Causing Exception:

- Lock Space Location
- Signal Exception

3C Service

3C01 Invalid Service Session State

The process is not in the proper service session for the request service command because of one of the following conditions:

- No service session exits for the process, and the command is other than start service session.

- The process is in service session, and the command is to start service session.

- The process is in service session, but a previous stop service session command was issued.

Instructions Causing Exception:

- Request I/O (service)

- Signal Exception

3C02 Unable to Start Service Session

The machine was unable to start a valid service session.

Instructions Causing Exception:

- Request I/O

- Signal Exception

3E Commitment Control

3E01 Invalid Commit Block Status Change

An attempt was made to modify (to an invalid status) the status of a commit block attached to the issuing process. The exception data defines the attempted change to the commit block status.

Information Passed:

• Commit block	System pointer
• Attempted status change (as defined in the modifications options in the modification template for the Modify Commit Block instruction)	Char(2)

Instructions Causing Exception:

- Modify Commit Block

- Signal Exception

3E03 Commit Block Is Attached to Process

The identified commit block is attached to a process making the operation requested impossible. The process that has the commit block attached is identified in the exception data.

Information Passed:

• Commit block	System pointer
• Process control space	System pointer

Instructions Causing Exception:

- Destroy Commit Block

- Signal Exception

3E04 Commit Blocks Control Uncommitted Changes

The identified commit block controls uncommitted changes, and an attempt was made to detach the commit block from the issuing process.

Information Passed:

• Commit block	System pointer

Instructions Causing Exception:

- Modify Commit Block
- Signal Exception

3E06 Commitment Control Resource Limit Exceeded

One of the resource limits for the commitment control functions has been reached.

Information Passed:

• Commit block	System pointer
• Condition code Hex 01=Lock limit exceeded Hex 02=Object list size exceeded Hex 03=Limit of attached commit blocks in the system exceeded	Char(1)

Instructions Causing Exception:

- Insert Data Space Entry
- Modify Commit Block
- Set Cursor
- Signal Exception

3E08 Object Under Commitment Control Being Journaled Incorrectly

All objects under commitment control must have their changes journaled through the same journal port as the commit block. An attempt was made to place an object under commitment control that did not meet this condition.

Information Passed:

• Object	System pointer
• Journal port that must be used	System pointer
• Journal port currently being used (binary 0 if not currently being journaled)	System pointer

Instructions Causing Exception:

- Modify Commit Block
- Signal Exception

3E10 Operation Not Valid Under Commitment Control

An operation was attempted on an object or through an object that was currently under commitment control. The operation is not supported under commitment control.

Information Passed:

• Object under commitment control	System pointer

Instructions Causing Exception:

- Copy Data Space Entries
- De-activate Cursor
- Destroy Cursor
- Insert Sequential Data Space Entries
- Retrieve Sequential Data Space Entries
- Signal Exception

3E11 Process Has Attached Commit Block

An attempt was made to attach a second commit block to a process.

Information Passed:

• Commit block (attached)	System pointer
• Commit block (attempted to attach)	System pointer

Instructions Causing Exception:

- Modify Commit Block
- Signal Exception

3E12 Objects Under Commitment Control

An attempt was made to detach a commit block from a process that has objects under commitment control.

Information Passed:

• Commit block	System pointer

Instructions Causing Exception:

- Modify Commit Block
- Signal Exception

3E13 Commit Block Not Journaled

An attempt was made to attach a commit block to a process, and the commit block was not being journaled.

Information Passed:

• Commit block	System pointer

Instructions Causing Exception:

- Modify Commit Block
- Signal Exception

3E14 Errors During Decommit

Errors were detected during an execution of a Decommit instruction. The exception data indicates the type of errors detected.

Information Passed:

• Commit block	System pointer
• Reserved (binary 0)	Char(1)
• Decommit status	Char(4)
– Damaged 0 = Commit block is not damaged 1 = Commit block is damaged	Bit 0
– Reserved (binary 0)	Bits 1-2
– Partially damaged 0 = Not partially damaged 1 = Partially damaged	Bit 3
– Reserved (binary 0)	Bits 4-15
– Decommit 0 = All changes were decommitted 1 = Not all changes were decommitted	Bit 16
– Journal read errors 0 = No journal read errors 1 = Journal read errors occurred during decommit	Bit 17
– Journal write errors 0 = No journal write errors 1 = Journal write errors occurred during decommit	Bit 18
– Partial damage to data space 0 = No damage encountered 1 = Damage encountered on one or more data spaces	Bit 19
– Damage to data space 0 = No damage encountered 1 = Damage encountered on one or more data spaces	Bit 20
– Function check 0 = No function check encountered 1 = Function check encountered	Bit 21
– Reserved (binary 0)	Bits 22-23
– Constant = 100	Bits 24-26
– Reserved (binary 0)	Bits 27-31
• Reserved (binary 0)	Char(7)
• Journal entry sequence number of start commit journal entry	Bin(4)

Instructions Causing Exception:

- Decommit
- Signal Exception

3E15 Object Ineligible for Commitment Control

The specified object is not eligible to be placed under commitment control.

Information Passed:

• Object	System pointer
• Reason code Hex 01=Object is a type that is not supported under commitment control Hex 02=Object is a cursor that was not activated under the issuing process Hex 03=Object is already under commitment control to this commit block or to another commit block Hex 04=Object is a cursor that holds data space entry locks Hex 05=Object is a join cursor	Char(1)

Instructions Causing Exception:

- Modify Commit Block
- Signal Exception

3E16 Object Ineligible for Removal from Commitment Control

The specified object cannot be removed from commitment control.

Information Passed:

• Object	System pointer
• Reason code Hex 01=Object is a type that is not supported under commitment control Hex 02=Object is not under commitment control of this commit block Hex 03=Object is a cursor holding data space entry locks	Char(1)

Instructions Causing Exception:

- Modify Commit Block
- Signal Exception

40 Dump Space Management

4001 Dump Data Space Size Limit Exceeded

An insert of dump data for a dump of a size greater than that allowed by the dump data size limit attribute of the target dump space is attempted. The exception is detected on the initial insertion of dump data for the dump.

The operation is suppressed in that none of the dump data is inserted into the dump space. The dump data size limit attribute can be modified through use of the Modify Dump Space instruction to increase the value of the limit if desired.

Information Passed:

- System pointer to the dump space

Instructions Causing Exception:

- Insert Dump Data
- Signal Exception

4002 Invalid Dump Data Insertion

An Insert Dump Data instruction attempted to insert dump data into a dump space but the insertion is invalid.

Information Passed:

• Space pointer to the dump space	System pointer
• Space pointer to the invalid block	System pointer
• Error code	Char(2)

Error codes and their meanings are as follows:

Code	**Meaning**
0001	The block of dump data addressed by the above space pointer was retrieved from the source dump space at an offset different from the offset at which it is being inserted.
0002	The block of dump data addressed by the above space pointer has been altered in value since it was retrieved from the source dump space or is not valid relative to the current dump data contained within the dump space.
0003	The block of dump data addressed by the above space pointer was produced on a newer version of the machine in a manner which is not compatible with the version of the machine performing the insert operation.

Instructions Causing Exception:

- Insert Dump Data
- Signal Exception

4003 Invalid Dump Space Modification:

An attempt has been made to modify the attributes of a dump space but the modification is invalid.

Information Passed:

• System pointer to the dump space	
• Error code	Char(2)

Error codes and their meanings as as follows:

Code	**Meaning**
0001	Modification of the dump data size limit attribute to a value less than the current value of the dump data size attribute is specified.

Instructions Causing Exception:

- Modify Dump Space
- Signal Exception

4004 Invalid Dump Data Retrieval

A retrieve Dump Data instruction attempted to retrieve dump data from a dump space but the retrieval was invalid.

Information Passed:

- System pointer to the dump space

- Error code Char(2)

  Error codes and their meanings as as follows:

Code	**Meaning**
0001	The first block of dump data specified for retrieval is beyond the last block of dump data currently stored in the dump space.

Instructions Causing Exception:

- Modify Dump Space

- Signal Exception

Chapter 24. Event Specifications

Events are managed by using the event management instructions. See Chapter 15. *Event Management Instructions*. Each event is identified by specifying the event class, type, and subtype.

To monitor all the event types under an event class, a hex 00 is entered in the event type element field. To monitor all the event subtypes under an event type, a hex 00 is entered in the event subtype element field.

EVENT DEFINITION ELEMENTS

Event definitions contain the following elements:

- Event identification
  - Class
  - Type
  - Subtype
- Optional compare value
- Event-related data
  - Standard
  - Specific

Event Identification

Events are identified by class, type, and subtype as follows:

Event Class

Events are divided into classes such as queue events, process events, and machine status events. Valid entries for this event identification element are hex 0001-7FFF.

Event Type

The event type within a class further describes the event. Valid entries for this event identification element are hex 00-FF. Type hex 00 is never signaled by the machine. It is restricted to supporting the technique of generic monitoring of the event type.

Event Subtype

This entry further describes the event type. Valid entries for this event identification element are hex 00-FF. Subtype hex 00 is never signaled by the machine. It is restricted to supporting the technique of generic monitoring of the event subtype.

Compare Value Qualifier

Certain classes of machine events allow a compare value to be specified. The compare value can contain a system pointer, but the system pointer must be located in the first 16 bytes of the compare value. The system pointer can optionally be followed by a scalar; for example, a counter value limit. The compare value can be supplied to further qualify the event monitors.

For timer events, the compare value specifies the time of day or the realtime interval that, when reached, causes the event monitor to be signaled.

Event-Related Data

Associated with machine events is information made available to the event monitor that is monitoring the event when a signaled condition is met. Both standard and specific event-related data are supplied with all signals. This information can be materialized through the use of the Retrieve Event Data instruction.

Standard Event-Related Data

The following format describes the standard event-related data available for retrieval when an event monitor has been signaled. The format of the data is:

• Template size specification	Char(8)
– Number of bytes provided for retrieval	Bin(4)
– Number of bytes in event-related data	Bin(4)
• Reserved (binary 0)	Char(2)
• Event ID	Char(4)
– Class	Char(2)
– Type	Char(1)
– Subtype	Char(1)
• Compare value length	Bin(2)
• Compare value	Char(32)
• Indicators	Char(2)
– Origin of signal	Bit 0
0 = Signaled by the machine	
1 = Signaled by the Signal Event instruction	
– Compare value content	Bit 1
0 = System pointer not present	
1 = System pointer present	
– Reserved (binary 0)	Bits 2-15
• Event-specified data length	Bin(2)

This value is 0 for short form event monitors, and the following attributes are not supplied.

• Signals pending count	Bin(4)
• Time of event signal	Char(8)

This time is presented as a 64-bit unsigned binary value in which bit 41 equals 1024 microseconds.

• Process (causing signal is denoted by process control space system pointer)	System pointer

This attribute is set to binary 0 if the event signal is not related to a process action, such as a timer event.

A pointer of all 0's is returned if the signaling process or process controls no longer exists when the data is retrieved.

• Event-specific data	Char(*)

Specific Event-Related Data

Machine events contain specific event-related data, which is in addition to the standard event-related data that accompanies the event signal.

This specific data is logically appended to the standard event-related data when an event handler retrieves the data.

The specific event-related data format is defined for each machine event under *Event Definitions*, later in this chapter.

EVENT DEFINITIONS

This section gives the definitions of the events that can be monitored. They are arranged in numeric order by event class. The types and subtypes within each event class are in numeric order. Subheadings under each event class give the combined type and subtype number and name followed by the compare value and event related data.

0002 Authorization

0101 Object Authorization Violation

Compare Value: None allowed

Event-Related Data: System pointer to the object

0201 Privileged Instruction Violation

Compare Value: None allowed

Event-Related Data: None

0301 Special Authorization Violation

Compare Value: None allowed

Event-Related Data: None

0004 Controller Description

0401 Controller Description Successful Contact

Compare Value: Allowed
- System pointer to the controller description

Event-Related Data:
- System pointer to the controller description
- System pointer to the network description (supplied only for CD type 10; otherwise, binary 0)

– Data length (hex 0000)	Bin(2)
– Variable data	None

0402 Controller Description Unsuccessful Contact

Compare Value: Allowed
- System pointer to the controller description

Event-Related Data:
- System pointer to the controller description
- System pointer to the network description (supplied only for CD type 10; otherwise, binary 0)

– Data length (2 to 66 bytes)	Bin(2)
– Variable data (2 to 66 bytes)	Char(*)
Status	Char(2)
XID data or SSCP ID data from the contacted station (up to 64 bytes)	Char(*)

0403 Loss of Contact

Compare Value: Allowed
- System pointer to the controller description

Event-Related Data:

– System pointer to the controller description	
– System pointer to the network description only for controller description type 10, otherwise 0	
– Data length (hex 000E or decimal 14	
Status code	Char(2)
Reserved	Char(12)

0501 Controller Description Failure (station inoperative)

0502 Controller Description Failure (protocol violation detected)

0503 Controller Description Failure (SSCP to physical unit session inactive)

0504 Controller Description Failure (BSC MTAM protocol considerations)

Compare Value (for all subtypes): Allowed
- System pointer to the controller description

Event-Related Data (for all subtypes):

– System pointer to the controller description	
– Data length (hex 001E)	Bin(2)
– Variable data	Char(30)
Error code (see *IBM System/38 Functional Reference Manual—Volume 2*, GA21-9800 for local source/sink devices, communications, and locally attached work stations)	Char(2)
Timestamp (if matching error log entry)	Char(8)
OU number	Char(2)
Optional data (see *IBM System/38 Functional Reference Manual—Volume 2*, GA21-9800 for local source/sink devices, communications, and locally attached work stations)	Char(2)
Optional system pointer	Char(16)

0601 Controller Description Manual Intervention

0602 Controller Description Unbound Intervention

0603 Controller Description Switched Intervention

0604 Controller Description Primary Intervention

Compare Value (for all subtypes): Allowed
- System pointer to the controller description

Event-Related Data (for all subtypes):

– System pointer to the controller description	
– System pointer to the network description (binary 0 if not switched line)	
– Data length (hex 000E)	Bin(2)
– Variable data	Char(14)
Status (manual dial operation hex 0001) (see *IBM System/38 Functional Reference Manual—Volume 2*, GA21-9800 for local source/sink devices, communications, and locally attached work stations)	Char(2)
Timestamp	Char(8)
OU number	Char(2)
Optional data	Char(2)

0007 Data Space

0301 Data Space Compression Threshold Exceeded

Compare Value: Allowed
- System pointer to the data space

Event-Related Data:
- System pointer to the data space

0008 Data Space Index

0301 Data Space Index Invalidated (signaled when data space index was unexpectedly invalidated)

Compare Value: Allowed
- System pointer to the data space index

Event-Related Data:
- System pointer to the data space index

000A Lock

0101 Object Locked (after asynchronous wait—signaled to receiving process)

Compare Value: None allowed

Event-Related Data:
- Space pointer to original lock request template

0201 Object Destroyed (during asynchronous wait—signaled to requesting process)

Compare Value: None allowed

Event-Related Data: None

0301 Object Lock Transferred (signaled to receiving process)

Compare Value: None allowed

Event-Related Data:
- A copy of the lock transfer template.
- For lock or unlock, the lock transfer template contains one entry for each lock transferred. The template is binary 0 except for the number of entries, the offset to the selected bytes, the system pointers, the lock state selection bit, and the entry active bit. The system pointers provided contain no authority.

0401 Asynchronous Lock Wait Time-Out (signaled to requesting process)

Compare Value: None allowed

Event-Related Data: None

000B Logical Unit Description

0401 Unformatted Supervisory Service Request

0402 Formatted Supervisory Service Request

Compare Value (for all subtypes): Allowed
- System pointer to the logical unit description

Event-Related Data (for all subtypes):

– System pointer to the logical unit description	
– Data length	Bin(2)
– Variable data (RU data as received–up to 80 bytes allowed)	Char(*)

0501 Logical Unit Description Unsolicited Incoming Messages Expedited

0502 Logical Unit Description Unsolicited Incoming Messages Nonexpedited (with or without data)

0503 Logical Unit Description Unsolicited Incoming Messages SSCP to LU Unsolicited Data

0504 Logical Unit Description Unsolicited Incoming Messages BSC Line Bid Received or BSC Emulation Select Received

0505 Logical Unit Description Unsolicited Incoming Messages Expedited (secondary)

0506 Logical Unit Description Unsolicited Incoming Messages Nonexpedited (secondary)

0507 Logical Unit Description Unsolicited Incoming Messages BSC MTAM Invalid Data Received

0508 Logical Unit Description Unsolicited Incoming Messages BSC MTAM Data Stream Restarted

0509 Logical Unit Description Unsolicited Incoming Messages BSC MTAM Incoming Data Discarded

0510 Logical Unit Description Unsolicited Incoming Messages BSC MTAM Device Conflict

0520 Logical Unit Description APPC Conversation Incoming Data Expedited Flow

0521 Logical Unit Description APPC Conversation Incoming Data Nonexpedited Flow

0522 Logical Unit Description Attached Request Not Activated

0523 Logical Unit Description Attached Request Maximum Exceeded

Compare Value (for all subtypes): Allowed

– System pointer to the logical unit description	
– Conversation ID (subtypes 20 and 21 only)	Char(16)
– Mode name (subtype 22 only)	Char(8)
– Unit of work ID (subtype 23 only)	Char(32)

Event-Related Data (for all subtypes):

– System pointer to the logical unit description	
– Data length (hex 0000)	Bin(2)
– Variable data	None

0601 Logical Unit Description Contact Successful

Compare Value:

– System pointer to the logical unit description

Event-Related Data:

– System pointer to the logical unit description	
– Reserved binary 0)	Char(16)
– Data length (hex 0000)	Bin(2)
– Variable data	None

0602 Logical Unit Description Contact Unsuccessful

Compare Value: Allowed

– System pointer to the logical unit description

Event-Related Data:

– System pointer to the logical unit description	
– Reserved (binary 0)	Char(16)
– Data length (hex 0012)	Bin(2)
– Variable data	Char(18)
– Status	Char(2)
– Additional data	Char(16)

The following status values are defined:

Reason	**Status**	**Additional Data**
Invalid response to ACTLU	0001	First 4 bytes of ACTLU response
Unable to communicate with device	0002	None
LUD not varied on (LUD and CD are for a primary station)	0005	None

0701 Operator Intervention Required

0702 Not Ready to Ready Transition

Compare Value: Allowed
- System pointer to the logical unit description

Event-Related Data:

– System pointer to the logical unit description	
– Reserved (binary 0)	Char(16)
– Data length (hex 000E)	Bin(2)
– Variable data	Char(14)
Status (see *IBM System/38 Functional Reference Manual—Volume 2*, GA21-9800 for local source/sink devices, communications, and locally attached work stations)	Char(2)
Timestamp (if matching error log entry)	Char(8)
OU number	Char(2)
Optional data	Char(2)

0801 Device Failure (inoperative)

0802 Device Failure (not available)

0803 Device Failure (SSCP to LU session inactive)

0804 Device Failure (LU to LU termination)

Compare Value (for all subtypes): Allowed
- System pointer to the logical unit description

Event-Related Data:

– System pointer to the logical unit description	
– Data length (hex 000E)	Bin(2)
– Variable data	Char(14)
Error code (see *IBM System/38 Functional Reference Manual—Volume 2*, GA21-9800 for local source/sink devices, communications and locally attached work stations)	Char(2)
Timestamp of matching error log entry	Char(8)
OU number	Char(2)
Optional data (see *IBM System/38 Functional Reference Manual—Volume 2*, GA21-9800 for local source/sink devices, communications, and locally attached work stations)	Char(2)

0901 Session Related Event (request I/O completed signaled to requesting process only when the flag is set in the SSR)

0902 Idle Session Event (signaled machine wide)

0903 No Session Event (signaled machine wide)

0904 Incoming Data and No Outstanding Request I/O instruction (signaled only to the process that activated the session)

0905 Request Path Operation Completed (signaled to the requesting process only)

Compare Value (for all subtypes): Allowed
- System pointer to the logical unit description

Event-Related Data:

– System pointer to the logical unit description	
– Data length (hex 0000 to 0102)	Bin(2)
– Variable data	Char(2 + N)
Key length	Char(2)
Key'(10 to 256 bytes)	Char(N)
The variable data includes a 2-byte length of key and an N-byte key from the SSR.	

0A01 Request I/O or Request Path Operation Response Queue Destroyed (signaled to requesting process only)

Compare Value: Allowed
- System pointer to the logical unit description

Event-Related Data:

– System pointer to the logical unit description	
– Space pointer to the SSR or path operation template (source/sink request)	
– Data length (hex 0000)	Bin(2)
– Variable data	None

0B02 Pass-Thru Terminated

Compare Value: Allowed
- System pointer to the requestor logical unit description

Event-Related Data:

– System pointer to the requestor logical unit description	
– Data length	Bin(2)
– Termination code	Char(4)
– VLOG ID	Char(8)
– System name	Char(8)
– Failing device name	Char(10)

000C Machine Resource

0201 Machine Auxiliary Storage Threshold Exceeded

Compare Value: None allowed

Event-Related Data:

– Machine auxiliary storage threshold (set to 0 when event is signaled)	Bin(8)
– Current amount of auxiliary storage used by the machine	Bin(8)
– Current amount of auxiliary storage available in the machine	Bin(8)

0301 Machine Ineligible State Threshold

Compare Value: None allowed
Event-Related Data:

– Number of processes in the ineligible state	Bin(2)
– Machine ineligible threshold value	Bin(2)

0401 MPL (multiprogramming level) Class Ineligible State Threshold

Compare Value: None allowed

Event-Related Data:

– MPL class ID	Bin(2)
– Number of processes in ineligible state in the MPL class	Bin(2)
– MPL class ineligible threshold value	Bin(2)

0501 Machine Address Threshold Exceeded

Compare Value: None allowed

Event-Related Data:

– Reserved (binary 0)	Char(7)
– The address generator that was depleted	Char(1)
Reserved (binary 0)	Bits 0-5
Temporary generator depleted	Bit 6
Permanent generator depleted	Bit 7
– Total permanent address possible	Char(8)
– Total temporary addresses possible	Char(8)
– Permanent addresses remaining	Char(8)
– Temporary addresses remaining	Char(8)
– Permanent address generation threshold	Char(8)
– Temporary address generation threshold	Char(8)

0601 Load/Dump Session Buffer Released

Compare Value: None allowed

Event-Related Data:

– ID of the storage pool that the buffer was released from	Bin(2)

000D Machine Status

0101 Machine Check

Compare Value: None allowed

Event-Related Data:

– Machine-related data	Char(16)

Machine-related data contains information about the type of machine check and the status of the machine. The following chart shows the byte significance of the machine-related data field.

Byte	Bit	Name	Indicates
1	0	Machine check status	The severity of the error 0 = Permanent machine check. An unrecoverable machine check occurred. 1 = Recovered machine check. A machine malfunction occurred, and the machine recovered. Information is available for a record of recovered machine checks. No special recovery is required as a result of a recovered machine check because these checks do not affect the process active at the time of the machine check.
	1	Machine check occurrence	Where the error occurred 0 = Occurred in a process. 1 = Occurred in a machine component unrelated to a process.

Byte	Bit	Name	Indicates
1 (continued)			
	2-7	Reserved (binary 0)	
2	0-7	Reserved (binary 0)	
3	0-7	Machine check type	This type of machine check for diagnostic purposes only.
4	0-7	Machine check log status	Whether or not the machine check is logged within the machine. Bit 0=1 Machine check is logged and can be retrieved via the machine service function. Bit 0=0 Machine check is not logged.
5-6	0-15	Machine check log length	The length of the machine check log.
7-8	0-15	Reserved (binary 0)	
9-16	0-63	Machine check timestamp	The time of day the machine check occurred. This field can be used to relate the machine check event to a machine check logged within the machine.

- Process-related data Char(18)
- Program System pointer
- Instruction number Bin(2)
- VLOG dump ID Char(8)
- Reserved Char(6)
- Timestamp (time of machine check) Char(8)
- Error code that indicates type of machine check (machine dependent) Bin(2)
- Reserved (binary 0) Char(6)
- VLOG ID Char(8)
- Error class Bin(2)
  Hex 0000=Unspecified abnormal condition
  Hex 0002=Logically invalid device sector
  Hex 0003=Device failure

- Auxiliary storage device indicator. Defined for error classes hex 0002 and hex 0003. It is the OU number of the failing device or 0 for a main storage failure. Bin(2)
- Reserved (binary 0) Char(100)

0201 Auxiliary Power Supply Activated

0202 Auxiliary Power Supply De-activated

0203 Auxiliary Power Supply De-activated for Mini-Uninterruptible Power Supply

Compare Value: None allowed

Event-Related Data:

- Time stamp (time of return to normal system operations) Char(8)
- Duration on uninterruptible power supply (number of seconds) Bin(4)
- Reserved (binary 0) Char(20)

0301 Device Error Data File Is 80% Full

0302 Device Accounting Data File Is 80% Full

0303 Device Activity Data File Is 80% Full

0304 Device Error Data File Is 100% Full

0305 Device Accounting Data File Is 100% Full

0306 Device Activity Data File Is 100% Full

Compare Value (for all subtypes): None allowed

Event-Related Data (for all subtypes): None

000E Network Description

0401 SDLC XID Failure or SSCP ID Failure

0402 BSC XID Failure

0403 Disconnect Failure

0404 BSC MTAM Eligibility Failure

0405 XID Negotiation Failure

Compare Value: Allowed
- System pointer to the network description

Event-Related Data:

– System pointer to the network description	
– Data length (hex 0000 to hex 0040)	Bin(2)
– Variable data (up to 64 bytes of XID data if primary or 6 bytes of SSCP ID data if secondary)	Char(*)

0501 Network Description Line Failure

0502 Network Description SNA Protocol Violation

Compare Value: Allowed
- System pointer to the network description

Event-Related Data:

– System pointer to the network description	
– Data length (hex 000E)	Bin(2)
– Variable data	Char(*)

000F Ownership

0101 Ownership Changed

Compare Value: None allowed

Event-Related Data:

– Object type	Char(1)
– Object subtype	Char(1)
– Object name	Char(30)
– Old user profile object type	Char(1)
– Old user profile object subtype	Char(1)
– Old user profile object name	Char(30)
– New user profile object type	Char(1)
– New user profile object subtype	Char (1)
– New user profile object name	Char (30)

0010 Process

0102 Process Initiated (signaled to initiating process)

Compare Value: None allowed

Event-Related Data:
- System pointer to the process control space pointer

0202 Process Terminated (signal to initiating process)

Compare Value: None allowed

Event-Related Data:

- System pointer to the process control space
- Termination type Char(1)
  Hex 01 = Process destroyed
  Hex 02 = Process failed to initiate
- Process status attributes Char(13)
  (See the *Materialize Process Attributes* instruction in Chapter 11 for the format of this scalar). This attribute has no meaning if termination type equals hex 02.
- Reserved Char(2)
- Exception-related data Char(*)
  This entry is used only if the process terminated as a result of an exception not being handled by the process. See *Chapter 10. Exception Management Instructions* for the details on exception-related data format.

0302 Process Suspended (signaled to initiating process)

Compare Value: None allowed

Event-Related Data:

- System pointer to the process control space

0402 Process Resumed (signaled to initiating process)

Compare Value: None allowed

Event-Related Data:

- System pointer to the process control space

0501 Process Time Slice Expired Without Entering Instruction Wait

Compare Value: Allowed

- System pointer to the process control space

Event-Related Data:

- System pointer to the process control space

0701 Maximum Processor Time Exceeded

Compare Value: Allowed

- System pointer to the process control space

Event-Related Data:

- System pointer to the process control space
- Current amount of processor time used Char(8)

0801 Process Storage Limit Exceeded

Compare Value: Allowed

- System pointer to the process control space

Event-Related Data: None

0011 Program

0301 Invocation Exit Bypassed Due to a RTNEXCP or a SIGEXCP Instruction

Compare Value: Allowed
- System pointer to the process control space

Event-Related Data:
- System pointer to the program that set the invocation exit that is being bypassed
- System pointer to the invocation exit program that is being bypassed
- System pointer to the program executing a RTNEXCP or a SIGEXCP instruction that caused this event

0302 Invocation Exit Bypassed Due to a Process Termination

Compare Value: Allowed
- System pointer to the process control space

Event-Related Data:
- System pointer to the program that set the invocation exit that is being bypassed
- System pointer to the invocation exit program that is being bypassed

0401 Failure to Invoke Program (Invocation Exit or Exception Handler)

Compare Value: Allowed
- System pointer to the process control space

Event-Related Data:

– Invocation type Hex 04 = External exception handler Hex 08 = Invocation exit	Char(1)
– Reserved	Char(15)
– Exception-related data This entry is used only if the process terminated as a result of an exception not being handled by the process. See *Chapter 10. Exception Management Instructions* for the details on exception-related data format.	Char(*)

0012 Queue

0301 Queue Message Limit Exceeded

Compare Value: Allowed
- System pointer to the queue

Event-Related Data:
- System pointer to the queue accessed with the Enqueue instruction
- Maximum number of messages from the queue attributes

0401 Queue Extended

Compare Value: Allowed
- System pointer to the queue

Event-Related Data:

– System pointer to the extended queue	
– New maximum number of messages value	Bin(4)

0014 Timer

0101 Time-of-Day Clock Reached or Exceeded Specific Value

Compare Value: Required

– Time-of-day clock value	Char(8)

Event-Related Data: None

0201 A Single Specific Time Interval Has Elapsed

This occurred since the event monitor was:
- Created enabled
- Enabled after being established disabled

Compare Value: Required

– Time interval	Char(8)

Event-Related Data: None

0301 A Repetitive Time Interval Has Elapsed

This occurred since the event monitor was:
- Established enabled
- Enabled
- Last signaled

The timer continues to be monitored for the next interval.

Compare Value: Required
- Time interval (minimum repetitive time interval is 1048 milliseconds) Char(8)

Event-Related Data: None

0016 Machine Observation

0101 Instruction Reference (signal to process only)

Compare Value: None allowed

Event-Related Data:
- System pointer to the associated program pointer (current invocation)
- Invocation attribute (current invocation) Char(16)
- Instruction number to be executed Bin(2)

0301 Invocation Reference (signal to process only)

Compare Value: None allowed

Event-Related Data:
- System pointer to the associated program pointer (old invocation)
- System pointer to the associated program pointer (new invocation)
- Invocation attribute (old invocation) Char(16)
- Invocation attribute (new invocation) Char(16)
- Old instruction number Bin(2)
- New instruction number Bin(2)
- Type of external reference Char(2)
  Hex 0001 = Call external
  Hex 0002 = Transfer control
  Hex 0003 = Event handler
  Hex 0004 = External exception handler
  Hex 0005 = Internal or branch point exception handler
  Hex 0006= Return from exception handler
  Hex 0007= Invocation exit
  Hex 0008= Return external
  Hex 0009= Invocation termination due to resignaling exception to a previous invocation
  Hex 000A= Invocation termination due to return from exception
  Hex 000B= Termination phase termination
  Hex 000C= Termination due to unhandled exception
  Hex 000E= Invocation termination
- Number of page reads into main storage associated with data base Bin(4)
- Number of page reads into main storage not associated with data base Bin(4)
- Number of page writes from main storage Bin(4)
- Number of transitions into ineligible wait state Bin(2)*
- Number of transitions into MI instruction wait Bin(2)*
- Number of transitions into ineligible wait from MI instruction wait Bin(2)*
- Processing unit time used Char(8)

The Bin(2) fields shown with an asterisk (*) have a limit of 32767. Should these limits be exceeded, the count is set to zero and no exception is signaled.

If there is no invocation for the old or new instruction number, the program pointer, invocation attributes, and instruction number is 0.

Reference types hex 0001 through hex 0007 are signaled if the trace invocations bit is set in the current (old) invocation.

Reference types hex 0002, 0008, 0009, 000A, 000B, 000C, and 000E are signaled if the trace returns bit is set in the current (old) invocation.

The following paragraphs describe each reference type.

Call External: The old invocation issued a CALLX instruction invoking the new invocation. The old instruction number locates the CALLX instruction. The new instruction number locates the entry point of the called program.

Transfer Control: The old invocation issued a XCTL instruction, terminating the old invocation and invoking the new invocation. The old instruction number locates the XCTL instruction. The new instruction number locates the entry point of the transferred-to program.

Event Handler: The old invocation sensed that an event had been issued which the process was monitoring, invoking the new invocation (event handler). The old instruction number locates the next instruction to execute when the old invocation resumes. The new instruction number locates the entry point of the event monitor.

External Exception Handler: An exception in the old invocation, or one of the invocations below it, caused an exception which this invocation was handling, causing a new invocation for the external exception handler. The old instruction number locates the excepting instruction, or the invoking instruction if the exception was resignaled to this invocation. The new instruction number locates the entry point of the exception handler.

Internal or Branch Point Exception Handler: An exception occurred which is handled in this invocation. No new invocation is created. The old and new invocations are the same. The old instruction number locates the excepting instruction, or the invoking instruction if the exception was resignaled to this invocation. The new instruction number locates the first instruction of the internal or branch point handler.

Return from Internal Exception Handler: A Return From Exception instruction was executed, causing this invocation to resume normal execution. No invocation is created or destroyed, but the instruction number may have changed. The old instruction number locates the last instruction executed in this invocation. The new instruction number locates the instruction at which control resumes. These may be the same. The old and new programs are the same.

Invocation Exit: A Return From Exception instruction, a Signal Exception instruction, or a Terminate Process instruction caused an invocation exit to be invoked.

Return External: The old invocation issued a Return External instruction, causing the old invocation to be destroyed, and control returned to the previous invocation. The old instruction number locates the Return External instruction. The new instruction number locates the instruction at which control resumes.

Invocation Termination Due to Resignaling Exception: An exception occurred in the old invocation that handled the exception by resignaling it to the previous invocation. The old instruction number locates the excepting instruction. The new instruction number locates the instruction to which control would have returned.

Invocation Termination Due to Return from Exception: A Return From Exception instruction was executed by an external exception handler, causing the invocation in which the external exception handler was running to be terminated. The old instruction number locates the Return From Exception instruction. The new instruction number locates the instruction to which a return to next operation would return.

Termination Phase Termination: The termination phase of the process is terminated, terminating the old invocation. The old instruction number locates the instruction at which the termination occurred. The new instruction number locates the instruction to which control would have returned. All other invocations on the stack will get type hex 000E events.

Termination Due to Unhandled Exception: An exception occurred for which no handler was specified. The old instruction number locates the excepting instruction. The new instruction number locates the instruction to which control would have returned. All other invocations on the stack gets type hex 000E events.

Intervening Invocation Termination: Some action occurred in an invocation below the old invocation, causing the old invocation to be terminated. The causes are:

- An exception was resignaled to the old invocation and it, in turn, resignaled the exception.
- A Return From Exception instruction at a lower level returned to an invocation above the old invocation.
- An unhandled exception occurred, causing all invocations to be terminated.
- The termination phase terminated, causing all invocations to be terminated.

The old instruction number locates the instruction which invoked the lower level invocation. The new instruction number locates the instruction to which control would have returned.

0017 Damage Set

0201 Machine Context Damage Set

Compare Value: None allowed

Event-Related Data:

– Reserved (binary 0)	Char(16)
– VLOG dump ID	Char(8)
– Error class This field indicates how the damage was detected: Hex 0000=Previously marked damaged Hex 0001=Detected abnormal condition Hex 0002=Logically invalid device sector Hex 0003=Device failure	Bin(2)
– Auxiliary storage device indicator This field is defined for error class 0002. It is the OU number of the failing device or 0 for main storage failure	Bin(2)
– Reserved (binary 0)	Char(100)

0401 System Object Damage Set

Compare Value: None allowed

Event-Related Data:
- System pointer to the object pointer
- VLOG dump ID Char(8)
- Error class Bin(2)
  This field indicates how the damage was detected:
  Hex 0000=Previously marked damaged
  Hex 0001=Detected abnormal condition
  Hex 0002=Logically invalid device sector
  Hex 0003=Device failure
- Auxiliary storage device indicator Bin(2)
  This field is defined for error class 0002. It is the OU number of the failing device or 0 for main storage failure
- Reserved (binary 0) Char(100)

0801 Partial System Object Damage Set

Compare Value: None allowed

Event-Related Data:
- System pointer to the system object
- VLOG dump ID Char(8)
- Error class Bin(2)
  This field indicates how the damage was detected:
  Hex 0000=Previously marked damaged
  Hex 0001=Detected abnormal condition
  Hex 0002=Logically invalid device sector
  Hex 0003=Device failure
- Auxiliary storage device indicator Bin(2)
  This field is defined for error class 0002. It is the OU number of the failing device or 0 for main storage failure
- Reserved (binary 0) Char(100)

0019 Service

0101 Machine Trace Table Full

Compare Value: None allowed

Event-Related Data: None

001A Journal Port

0301 Entry Not Journaled

Compare Value: Allowed
- System pointer to the journal port

Event-Related Data:
- System pointer to the journal port
- Return value code Char(1)
  Hex 01=No journal space was attached to the specified journal port.
  Hex 02=Unable to obtain sufficient space on the journal space for the entry.
  Hex 03=Journal port is damaged.
  Hex 04=All journal spaces attached to the journal port are damaged.
  Hex 05=The journal sequence number has reached its maximum allowed value.
  Hex 06=Journal failure.

0401 Journal Space Attached to a Journal Port Became Unusable

Compare Value: Allowed
- System pointer to the journal port

Event-Related Data:
- System pointer to the journal port
- System pointer to the journal space is unusable (binary 0 if undetermined)

001B Commitment Control

0301 Commit Block Attached at Process Termination

Compare Value: Allowed
- System pointer to the commit block

Event-Related Data:

– System pointer to the commit block	
– Reserved	Char(1)
– Commit block status	Char(4)
Damage	Bit 0
0 = Damaged	
1 = Not damaged	
Reserved (binary 0)	Bits 1-2
Partially damaged	Bit 3
0 = Not partially damaged	
1 = Partially damaged	
Reserved (binary 0)	Bits 4-15
Decommit	Bit 16
0 = Changes are decommitted	
1 = Changes are not decommitted	
Journal read errors	Bit 17
0 = No read errors	
1 = Read errors occurred during decommit	
Journal write errprs	Bit 18
0 = No write errors	
1 = Write error occurred during decommit	
Partial damage to data space	Bit 19
0 = No partial damage encountered	
1 = Partial damage encountered on one or more data spaces	
Damage to data space	Bit 20
0 = No damage encountered	
1 = Damage encountered on one or more data spaces	
Function check	Bit 21
0 = No function check encountered	
1 = Function check encountered	
Reserved (binary 0)	Bits 22-23
Constant = 010	Bits 24-26
Reserved (binary 0)	Bits 27-31
– Reserved (binary 0)	Char(7)
– Journal entry sequence number of start commit journal entry	Bin(4)

0401 Errors During Decommit

Compare Value: Allowed
- System pointer to the commit block

Event-Related Data:

– System pointer to the commit block	
– Reserved (binary 0)	Char(1)
– Commit block status	Char(4)
Damage	Bit 0
0 = Not damaged	
1 = Damaged	
Reserved (binary 0)	Bits 1-2
Partially damaged	Bit 3
0 = Not partially damaged	
1 = Partially damaged	
Reserved (binary 0)	Bits 4-15
Decommit	Bit 16
0 = Changes are decommitted	
1 = Changes are not decommitted	
Journal read errors	Bit 17
0 = No journal read errors	
1 = Journal read errors occurred during decommit	
Journal write errors	Bit 18
0 = No journal write errors	
1 = Journal write errors occurred during decommit	
Partial damage to data space	Bit 19
0 = No partial damage	
1 = Partial damage on one or more data spaces	
Damage to data space	Bit 20
0 = No damage	
1 = Damage on one or more data spaces	
Function check	Bit 21
0 = No function check	
1 = Function check	

Reserved (binary 0)	Bits 22-23
Constant = 100	Bits 24-26
Reserved (binary 0)	Bits 27-31
– Reserved (binary 0)	Char(7)
– Journal entry sequence number of start commit journal entry	Bin(4)

001C Journal Space

0301 Threshold Reached

Compare Value: Allowed

- System pointer to the journal space

Event-Related Data:

– System pointer to the journal space	
– System pointer to the journal port	
– Threshold limit	Bin(4)

001D User Qualified Timer

All of the user qualified timer events are only signaled to the requesting process.

The optional user data that may be supplied as part of the compare value is determined by the user data specified on the associated execution of the Monitor Event instruction.

0101 Time-of-Day Clock Reached or Exceeded Specific Value

Compare Value: Required

– Time-of-day clock value	Char(8)
– User data (optional)	Char(8)

Event-Related Data: None

0201 A Single Specific Time Interval as Elapsed

This occurred since the event monitor was:

- Created enabled
- Enabled after being established disabled (any unexpired interval is discarded)

Compare Value: Required

– Time interval	Char(8)
– User data (optional)	Char(8)

Event-Related Data: None

0301 A Repetitive Time Interval Has Elapsed

This occurred since the event monitor was:

- Established enabled
- Enabled (any unexpired time interval is discarded)
- Last signaled

The timer continues to be monitored for the next interval.

Compare Value: Required

– Time interval (minimum repetitive time interval is 1048 milliseconds)	Char(8)
– User data (optional)	Char(8)

Event-Related Data: None

Chapter 25. Program Object Specification

All attributes, specifications, and ODT (object definition table) formats for each program object in the System/38 Instruction Set are discussed in this chapter. Charts in this chapter illustrate the combinations of attributes and specifications. The detailed formats for the ODV (ODT directory vector) and the OES (ODT entry string) are also specified in this chapter.

GENERAL ODT DESCRIPTION

A program template is composed of a header followed by several components, including an instruction stream component and an object definition table (ODT) component. The ODT contains the views of all objects referred to in the instruction stream other than those objects that are immediate value operands in the instructions. The following objects are ODT definable:

- Data object
  - Scalar data object
  - Pointer data object
- Constant data object
- Entry point
- Branch point
- Instruction definition list
- Operand list
- Exception description

The ODT entry consists of the ODV and the OES.

ODV

The ODV is a vector of 4-byte character string entries in a standard format. An ODV entry describes an object completely or partially. If the ODV entry does not completely describe the object, it must contain an offset into the OES where the object is described completely.

An ODV entry is required for each object described in the ODT. The index value for a particular object ODV entry is used as an operand for instructions that operate on the object. An ODT can contain 8191 entries in template version 0 and 65 526 entries in template version 1. The first entry has an index value of 1.

The structure of the ODV is designed to allow a complete definition of commonly used objects. An object that cannot be completely described in an ODV entry must have an OES entry to complete its definition.

Each ODV entry generally consists of the following:

- Type information
  - The first 2 bytes of each ODV entry contain information identifying the type and general attributes of the object.
- OES offset or attribute information
  - The last 2 bytes of each ODV entry contain either detailed attribute information or an offset into the OES where the detailed attribute information is found. The OES contains a 4-byte OES length entry at the beginning of the OES component. This means that the minimum valid offset is 4 bytes.

Object references in the System/38 instructions consist of instruction operands that contain index values into the ODV.

OES

The OES consists of a series of variable-length entries that complete an object's description.

If an OES entry exists for an object, its offset value into the OES is specified in the ODV entry for that object.

Several ODV entries for different objects with identical definitions can share the same OES entry. OES entries do not exist for those objects that can be completely described in the ODV.

Each OES entry consists of the following:

- OES header
  - One byte indicating which OES appendages are present. A bit is included for each possible OES appendage. A binary zero value for the bit means the appendage is not present. A binary one value for the bit means the appendage is present.
- OES appendages
  - A series of variable-length fields each containing a specific collection of information about the object.

The following are examples of object attributes specified in an OES entry.

- Object names
- Length/number of elements
- Explicit bases
- Explicit positions
- Initial values

When an OES entry is required to complete an object's description, its appendages must be in the same order as are the bits that indicated their presence in the OES header.

For example, assume the following OES header:

```
B ' 1 0 0 0 0 0 0 1 '
    |             |
    |             |
   Name          Initial Value
```

The name appendage must immediately follow the OES header, and the initial value appendage must immediately follow the name appendage.

The OES may consist of 0 to 16 777 215 bytes. Because the OES offset value may be a maximum of 65 535 bytes, a means is provided to address an OES offset beyond this maximum. A special object type value ('1111'B) in the ODV denotes an object description in which:

- A 3-byte offset to the OES entry is specified.
- The entire object description is specified in the OES entry (ODV, the OES header, and OES appendages).

See *References to OES Offsets Greater Than 64 K-1,* later in this chapter for a detailed description of this format.

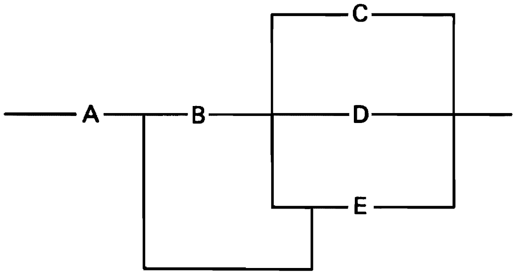
ODT ENTRIES IN DETAIL

In this section, the detailed definitions for the various ODT entries are discussed by object type. Each object type description contains the following information about its respective objects:

- Attribute combination charts–Summarize both the attributes of a given object and the valid combination of those attributes.

  In the attribute combination charts, the following rules are used:

  - A combination of attributes is allowed if the attributes lie on a single path that progresses from left to right through the diagram. For example:



    The attribute A can be used with B and C, B and D, B and E, or E only; but C cannot be used with D or E.

  - Optional attributes are noted where a solid line bypasses one or more attributes.

- ODV Format–Describes the various bit settings of the 4-byte ODV entry relative to the specific object type.

- OES Format–Describes the various OES header bit settings relative to the specific object type.

- Notes–Describe any unique characteristics concerning the specifications of the object.

Note: Reserved bits are those bits not being used currently and should always be set to binary 0.

Combinations of attributes not defined in these specifications cause a create program exception-invalid ODT exception to be signaled during the execution of the Create Program instruction.

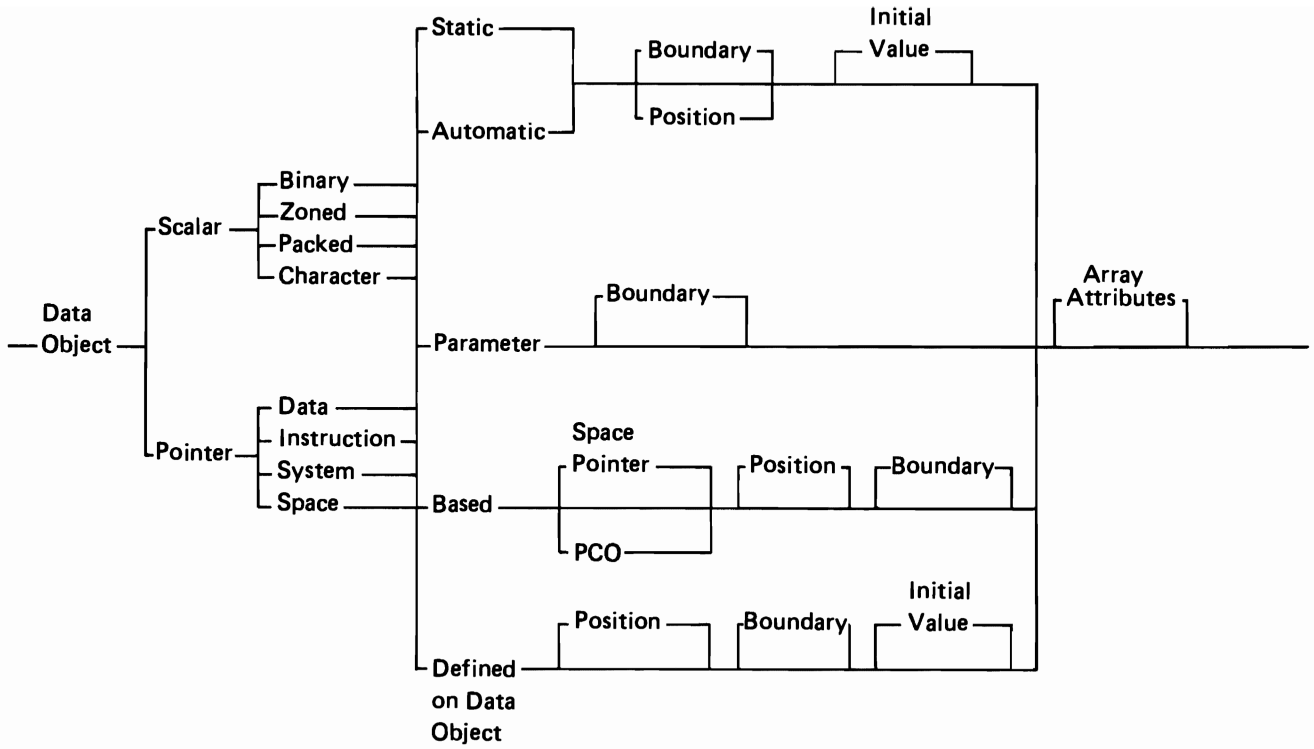
Data Object

Data objects provide operational and, possibly, representational characteristics to data in a space. Scalar data objects and pointer data objects are the two basic categories of data contained in the space.

Scalar data objects provide operational and representational characteristics for numeric and character data contained in a space.

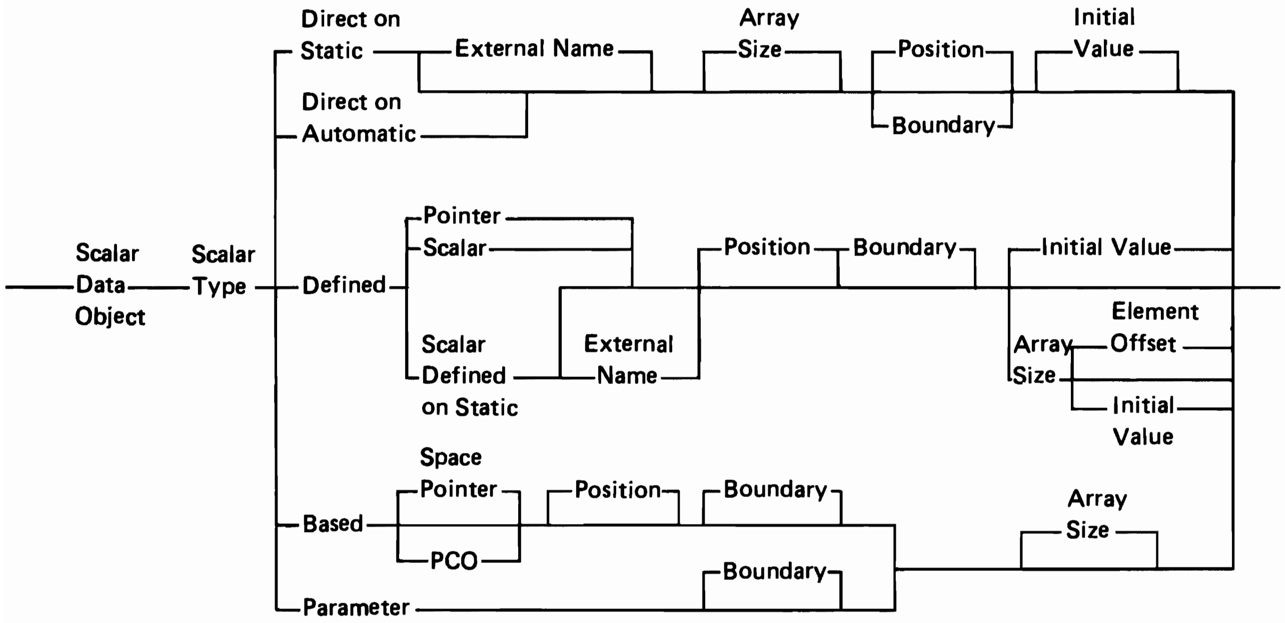
Pointer data objects provide operational characteristics for pointer data contained in a space.

The following chart shows the general characteristics of data objects.



Scalar Data Object

Attribute Combinations



ODV Format

Bits Meaning

0-3 Object type
0000 = Scalar data object

4 OES present
0 = OES is not present.
1 = OES is present because one or more of the following is true:
- Object is named and external.
- Object has initial value (not system default).
- Object has based or defined addressability.
- Object has direct addressability with explicit position.
- Object is an array.

5-7 Addressability type
000 = Direct static
001 = Direct automatic
010 = Based
011 = Defined
100 = Parameter
101 = Based on PCO (process communication object) space pointer
All others reserved

8 Optimization of value
0 = Normal value (can be optimized across several instructions)
1 = Abnormal value (cannot be optimized for more than a single reference because the value may be modified in a manner not detectable by the Create Program instruction)

9-11 Boundary
000 = None
001 = Multiple of 2
010 = Multiple of 4
011 = Multiple of 8
100 = Multiple of 16
101-111 = Reserved

Boundary is assumed to be specified for indirectly addressed program objects. A higher boundary alignment can improve performance when the program object is referenced.

12 System default initial value
0 = Do not use the system default initial value.
1 = Use the system default initial value.
- Numeric zero value for binary, packed, zoned, or floating-point
- Blank character value (hex 40) for character strings

13-15 Scalar type
000 = Binary
001 = Floating-point
010 = Zoned decimal
011 = Packed decimal
100 = Character
101-111 = Reserved

16-31 OES offset or scalar length

- If bit 4 of the ODV is 1 (OES is present), then bits 16-31 specify the offset to the OES entry for this object.

- If bit 4 of the ODV is 0 (OES not present), bits 16-31 represent the scalar length of the object as follows:

  If binary, then:

  Bits 16-31:Precision
  Hex 0002 = 2
  (binary only)
  Hex 0004 = 4
  All others reserved

  If floating-point, then:

  Bits 16-31:Precision
  Hex 0004 = 4
  Hex 0008 = 8
  All others reserved

  If zoned or packed decimal, then:

  Bits **Meaning**

  16-23 Digits (D) to the right of assumed decimal point, where 0 c D c T

  24-31 Total digits (T) in field, where 1 c T c 31

  If character string scalar, then:

  Bits 16-31: String length (L), where
  1 ≤ L ≤ 32 767

OES Format

OES Header

Bits **Meaning**

0 Name and external
0 = Object is not named and is not externally accessible.
1 = Object is named and is externally accessible.

1 Scalar length present
1 = Length is present (required).

2 Array information present
0 = Array information is not present.
1 = Array information is present.

3 Base present
0 = Base is not present.
1 = Base is present.

4 Position present
0 = Position is not present (required if boundary is specified).
1 = Position is present.

5 Initial value present in OES
0 = Initial value is not present.
1 = Initial value is present in OES.

6 Replications present in OES
0 = No replications in initial value.
1 = Replications in initial value (bit 5 = 1).

7 Reserved

Name Appendage

Bytes **Meaning**

0-1 Length (L) of name, where 1 c L c 32

2-L L characters of symbolic name

Note: Names of external data objects and the name of the program must be unique.

Scalar Length Appendage

Bytes 0-1: Scalar length

If binary, then:

Bytes 0-1: Precision
Hex 0002 = 2
(binary only)
Hex 0004 = 4
All others reserved

If zoned or packed decimal, then:

Byte	Meaning
0	Digits (D) to the right of assumed decimal point, where 0 c D c T
1	Total digits (T) in field, where 1 c T c 31

If floating-point, then:

Bytes 0-1: Precision
Hex 0004 = 4
Hex 0008 = 8
All others reserved

If character string scalar, then:

Bytes 0-1: String length (L),
1 ≤ L ≤ 32 767

Array Appendage

Bytes	Meaning
0-3	Number (N) of elements in the array, where 1 c N c 16 777 215
4-5	Array element offset If the array element offset attribute is specified (bytes 4-5 are nonzero), this field specifies the offset between initial bytes of the elements of a defined on array.

Base Appendage

Bytes 0-1: ODT reference for:

- Pointer data object or space pointer machine object, if based
- Scalar data object or pointer data object if defined

Position Appendage

Bytes 0-3: Position value for:

- Direct if not defaulting to next available byte or if no boundary defined
- Based if not 1
- Defined if not 1

Note: Position value is in terms of bytes with the first byte in position 1.

Initial Value Appendage

Bytes 0-L: Initial value in format and length as determined by scalar type

In the initial value appendage, a noncharacter string scalar must have an initial value of the proper size and format (for example, 2-byte binary value for a 2-byte binary scalar).

For arrays and character strings, if the replication bit in the OES is binary 1, the initial value portion must consist of components of the following form:

2 bytes:	Number of replications of associated value
2 bytes:	Length (L) of associated value
L bytes:	Associated value

The entire object must be initialized contiguously and byte by byte.

If the replication bit for an array is binary 0, the initial value appendage must have the following form:

4 bytes:	Length of initial value (less than or equal to the total number of bytes in the array)
L bytes:	The initial value of proper size and format to specify the initial values for each element of the array that is to be initialized

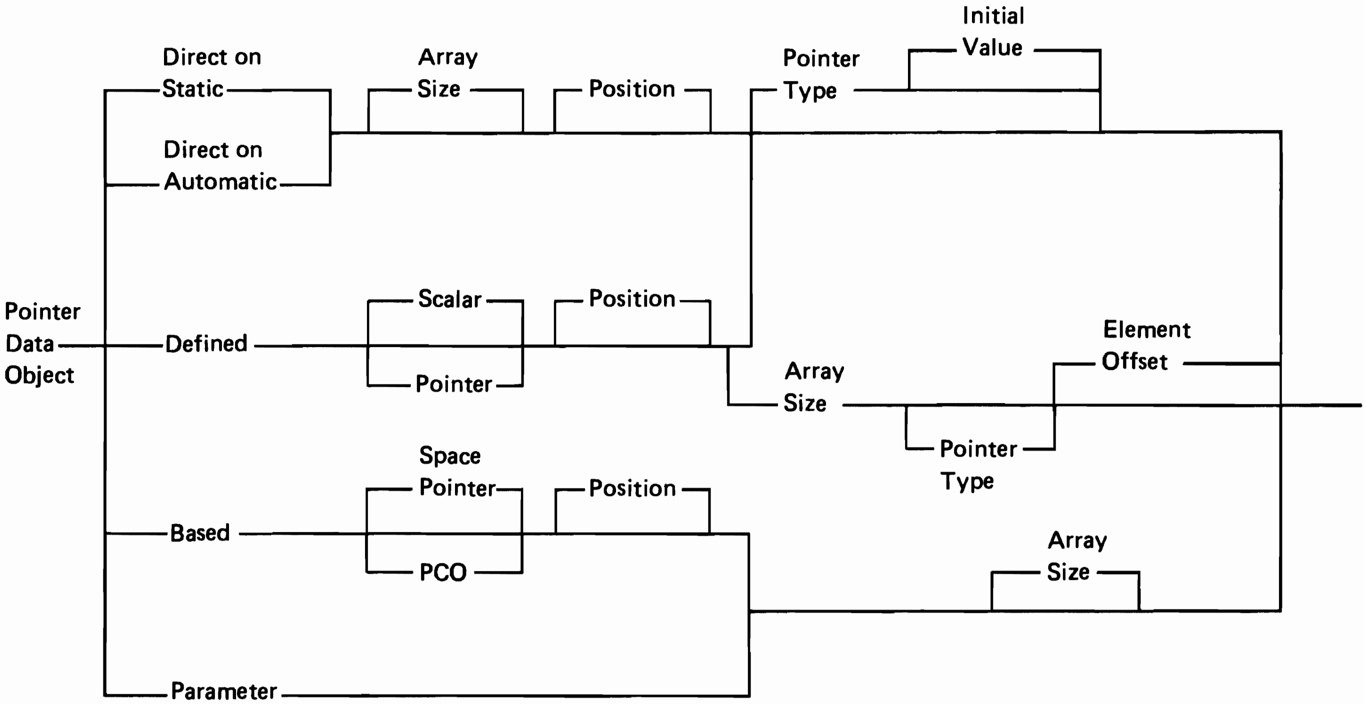
If the replication bit for a character string scalar is binary 0, the initial value appendage must be a byte string with a length equal to the object length.

Notes:

1. Scalar data objects with the external attribute must be mapped (direct or defined on direct) onto the static space. The names must be unique within the program template.
2. When used for address resolution, the name of an external data object is implicitly padded to 32 bytes by extending on the right with blank characters (hex 40).
3. See *Data Object Notes* later in this chapter for general notes concerning data objects.

Pointer Data Objects

Attribute Combinations



ODV Format

Bits	Meaning
0-3	Object type 0001 = Pointer data object
4	OES present 0 = OES is not present. 1 = OES is present because the object has initial value, base, or position, or the object is an array.
5-7	Addressability type 000 = Direct static 001 = Direct automatic 010 = Based 011 = Defined 100 = Parameter 101 = Based on PCO (process communication object) space pointer All others reserved
8	Optimization of value 0 = Normal value (can be optimized across several instructions) 1 = Abnormal value (cannot be optimized for more than a single reference because the value may be modified in a manner not detectable by the Create Program instruction)
9-11	Reserved
12-15	Pointer type (ignored unless initial value) 0001=Space pointer 0010=System pointer 0011=Data pointer 0100=Instruction pointer All others reserved
16-31	OES offset • If bit 4 of the ODV contains a binary 0, no OES is present, and bits 16-31 contain a value of binary 0. • If bit 4 of the ODV contains a binary 1, then an OES header is present in the OES at the offset specified in bits 16-31.

OES Format

OES Header

Bits	Meaning
0	Reserved
1	Initial value appendage modifier 0 = Instruction pointer initial value has the basic format. 1 = Instruction pointer initial value has the extended format. **Note**: This bit must be 0 for noninstruction pointer initial values.
2	Array information present 0 = Array information is not present. 1 = Array information is present.
3	Base Present 0 = Base is not present. 1 = Base is present.
4	Position present 0 = Position is not present. 1 = Position is present.
5	Initial value present 0 = Initial value not present. 1 = Initial value is present.
6-7	Reserved

Array Appendage

Bytes	Meaning
0-3	Number (N) of elements in the array, where 1 c N c 1 000 000
4-5	Array element offset If the array element offset attribute is specified (bytes 4-5 are nonzero), this field specifies the offset between initial bytes of the pointers of a defined array. Value must be a multiple of 16.

Base Appendage

Bytes 0-1: ODT reference for:
- Pointer data object or space pointer machine object, if based.
- Data object or pointer object if defined. Resulting location must be a multiple of 16.

Position Appendage

Bytes 0-3: Position value for:
- Direct if not defaulting to next available byte (must be a multiple of 16)
- Based if not 1
- Defined if not 1

Initial Value Appendage

If ODV bits 12-15 indicate instruction pointer:

If the initial value appendage modifier equals 0, the basic format is:

Bytes 0-1: Instruction stream reference; indirect or direct reference to an instruction number

- If bit 0 is 0, then bits 1-15 specify an indirect reference that contains the ODV number of a branch point that specifies the instruction number to be referenced.

- If bit 0 is 1, then bits 1-15 specify a direct reference that contains the instruction number to be referenced.

If the initial value appendage modifier equals 1, the basic format is:

Bytes 0-1: Instruction stream reference; indirect or direct reference to an instruction number

- If bit 0 is 0, then bits 8-23 specify an indirect reference that contains the ODV number of a branch point that specifies the instruction number to be referenced.

- If bit 0 is 1, then bits 8-23 specify a direct reference that contains the instruction number to be referenced.

Bits 1-7 are reserved (binary 0).

If ODV bits 12-15 indicate data pointer:

Bytes 0-1: Number of names in name list. One or two names may be specified as the initial value.

- If one name is specified, it must be in the name specification of the data object in the following format:
  - Number of namesBin(2) (value of 1)
  - Scalar data objectBin(2) name length (N)
  - Scalar data objectChar(N) name string, where $1 \leq N \leq 32$

- If two names are specified, the name of a program to be searched and the name of the external data object are specified as follows:
  - Number of namesBin(2) (value of 2)
  - Program typeChar(1) (hex 02)
  - Program subtypeChar(1)
  - Program nameBin(2) length (M)
  - Program nameChar(M) string, where $1 \leq M \leq 30$
  - Scalar data objectBin(2) name length (P)
  - Scalar data objectChar(P) name string, where $1 \leq P \leq 32$

If ODV bits 12-15 indicate space pointer:

Bytes 0-1: ODT number of a data object or pointer object that is direct or defined on direct.

If ODV bits 12-15 indicate system pointer, one or two names may be specified as the initial value.

- If one name is specified, it must be the name specification of the object in the following format:

– Number of names (value of 1)	Bin(2)
– Object type code	Char(1)
– Object subtype code	Char(1)
– Minimum authority code	Char(2)
– Object name length (N)	Bin(2)
– Object name string, where 1 ≤ N ≤ 30	Char(N)

- If two names are specified, the entry must be a context name and an object name in the following format:

– Number of names (value of 2)	Bin(2)
– Context type code (value of hex 04)	Char(1)
– Context subtype code	Char(1)
– Context name length (M)	Bin(2)
– Context name string, where 1 ≤ M ≤ 30	Char(M)
– Object type code	Char(1)
– Object subtype code	Char(1)
– Minimum authority code	Char(2)
– Object name length (P)	Bin(2)
– Object name string, where 1 ≤ P ≤ 30	Char(P)

Notes:

1. The object type codes that may be specified for system pointer initial values are as follows:

Code (Hex)	**Object Type**
01	Access group
02	Program
04	Context
07	Journal space
08	User profile
09	Journal Port
0A	Queue
0B	Data space
0C	Data space index
0D	Cursor
0E	Index
0F	Commit block
10	Logical unit description
11	Network description
12	Controller description
13	Dump space
19	Space
1A	Process control space

   All other codes are reserved and, if specified, cause an exception to be signaled.

2. The minimum authority codes that may be specified for system pointer initial values are as follows:

Bit	**Meaning**
0	Object control
1	Object management
2	Authorized pointer
3	Space authority
4	Retrieve
5	Insert
6	Delete
7	Update
8	Ownership
9-15	Reserved

   A value of binary 1 indicates that the object must have the specified authority in order for resolution to be performed. Zero or more authority bits may be specified, and if any are specified, all must be satisfied.

Reserved bits must have a value of binary 0.

A pointer with based addressability need not have a base specified. If a base is specified, the pointer can, in turn, be based on another pointer. An exception is signaled if the final pointer in the chain is not direct, is not defined on a direct data object, is not a parameter, or is not defined on a parameter. An exception is signaled if a base pointer in the chain is based on a pointer that was previously specified in the chain of based pointers.

3. A static space pointer may not be initialized to an automatic scalar data or pointer object.

   The last initialized pointer data object appearing in the ODT for a given storage location overlays all previous pointer object initial values for that location.
4. A pointer data object defining an array may not be initialized. See *Data Objects Notes* later in this chapter.
5. When the initial value specified for a system pointer or a data pointer is to be used for address resolution, the name string entry is implicitly extended to the standard length by padding with blank characters (hex 40). The standard length for system object names is 30 bytes and for external scalar data objects is 32 bytes.
6. See *Data Object Notes* later in this chapter for general notes concerning data objects.

Data Object Notes

The following notes apply to all declarations of data objects, scalars, and pointers. The term *data object* applies to either scalar data objects or pointer data objects unless explicitly qualified.

Notes:

1. Any specification of position uses position 1 as the first position in storage. A position value of 0 is invalid.
2. Data objects that are defined on other data objects must follow (not necessarily immediately) their associated bases in the ODT. If any data objects in a chain of defined-on objects have an initial value, none of the objects in the chain can have based or parameter addressability, and the first object in the chain must be direct on the static or automatic space. An initial value associated with a defined data object overlays all initial values associated with data objects that preceded it in the chain. The portion of a scalar data object initial value that overlays any part of an initialized pointer data object is ignored.

   If more than one data object initializes the same byte (or bytes) in a space, the value associated with the data object appearing last in the ODT overlays the others.
3. For data objects with the direct mapping type but no explicit position, the Create Program instruction provides default position (position in the static or automatic allocation) ODT information. ODT entries for defaulting direct objects must appear in the order desired. Declarations for other program objects can be interleaved with these defaulting direct data and pointer objects.

   Two examples follow:

These ODT entries	Would be treated as these:
A Char(2) Direct Static	A Char(2) Direct Static Pos(1)
B Pkd(3,3) Direct Static	B Pkd(3,3) Direct Static Pos(3)
D Pointer Direct Static	D Pointer Direct Static Pos(17)

These ODT entries	Would be treated as these:
A Char(4) Direct Static	A Char(4) Direct Static Pos(1)
B Char(4) Direct Static Pos(20)	B Char(4) Direct Static Pos(20)
C Char(4) Direct Static	C Char(4) Direct Static Pos(24)
D Char(4) Direct Static Pos(10)	D Char(4) Direct Static Pos(10)
E Char(2) Defined B	E Char(2) Direct Static Pos(20)
F Char(3) Direct Static	F Char(3) Direct Static Pos(28)

The default value for the position depends on whether the boundary attribute is specified. Pointer objects always have a default boundary attribute that is a multiple of 16.

A boundary specification for a direct data object causes the data object to be located at the next available position having an offset value that is a multiple of the boundary specified. If boundary is specified, a value may not be specified for the position attribute. The boundary specification for a direct pointer object is always a multiple of 16. A position specified for a direct pointer object must be a multiple of 16.

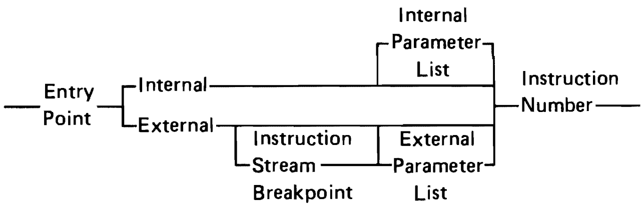
If neither position nor boundary is specified for a direct data object, the object is located at the next available position without regard to boundary alignment.

The size for the static and automatic spaces can also be defaulted if the size of static (automatic) storage entry in the program template is 0. The value used is such that the space is large enough to contain all data objects with defaulted and explicit direct mapping. An exception is signaled if a size that is insufficient to contain defaulted and direct mapped data objects is specified.

To summarize, the rules related to positioning data objects and setting of the size through defaulted directed data objects are:
- A defined-on data object may extend beyond the end of the data object it is defined on. It may not, however, extend beyond the total allocated space to which it maps.
- Explicit positioning of a data object may be intermixed with implicit positioning. That is, for any ODT, a specification of the position for a data object does not preclude a succeeding data object requiring a default position.
- A defaulted scalar data object is assigned to the highest assigned position for a direct data object plus 1. A defaulted pointer data object is assigned to the next available 16-byte aligned position beyond the highest assigned position. It is possible, through explicit positioning, to create a gap in a space. This gap is not filled in through default positioning of a following data object or pointer object. The defaulted object follows the explicitly positioned data object.

Entry Point

Attribute Combinations



ODV Format

Bits	Meaning
0-3	Object type 0010=Entry point
4	OES present 0 = OES is not present 1 = OES is present because of entry point parameters or the instruction stream breakpoint
5-14	Reserved (binary 0)
15	Scope 0 = Internal 1 = External
16-31	OES offset or entry point value OES offset if OES present (bit 4=1). Entry point value equals the instruction number if no entry point parameters are listed

OES Format

OES Header

Bits	Meaning
0	Instruction stream breakpoint 0 = Appendage not present 1 = Instruction stream breakpoint appendage present
1-2	Reserved (binary 0)
3	Parameter information present 0 = Parameter information not present 1 = Parameter information is present
4-6	Reserved (binary 0)
7	Initial value present 1 = Initial value is required

Parameter Appendage

Bytes 0-1: ODT reference for operand list describing entry point parameters

Initial Value Appendage

Bytes 0-1: Instruction number for entry point

Instruction Stream Breakpoint Appendage

Bytes 0-1: Instruction number for instruction stream breakpoint identifies first instruction of latter half of program that is not executed in the normal path

Notes:

1. More than one internal entry point can reference the same instruction.
2. An internal entry point and an external entry point cannot reference the same instruction.
3. Only one external entry point can be defined in a program.
4. An internal entry point can only reference an internal parameter list; likewise, an external entry point can only reference an external parameter list.
5. An instruction stream breakpoint can only be defined with an external entry point. The instruction number specified must be greater than that of the external entry point.
6. If no entry point is declared, the first instruction is assumed to be the entry point with no parameters.
7. If external parameters are declared in a program, an external entry point must be declared. If one is not declared, an ODT relational error exception is signaled by the Create Program instruction.

Branch Point

Attribute Combinations

— Branch——Instruction Number
Point

ODV Format

Bits	Meaning
0-3	Object type 0011=Branch point
4	OES present 0 = OES is never present
5-15	Reserved (binary 0)
16-31	Value
	Instruction number (N) of branch point, where 1 c N c 32 767.

OES Format

No OES is needed for branch points.

Instruction Definition List

Attribute Combinations

— Instruction—— Number of———N Instruction———
Definition Elements (N)Reference
List

ODV Format

Bits	Meaning
0-3	Object type 0100=Instruction definition list
4	OES present 1 = OES is present (always present to contain the list information)
5-15	Reserved (binary 0)
16-31	OES offset

OES Format

OES Header

Bits	Meaning
0	Reserved (binary 0)
1	Initial value appendage modifier 0 = Initial value instruction references have the basic format. 1 = Initial value instruction references have the extended format.
2-6	Reserved (binary 0)
7	Initial value present 1 = Initial value is required

Initial Value Appendage

Bytes	Meaning
0-1	Number (N) of list elements, where 1 c N c 255

If the initial value appendage modifier equals 0, the basic format:

2-3	Instruction reference element 1
.	
.	
.	
X-X+1	Instruction reference element N

If the initial value appendage modifier equals 1, the basic format:

2-4	Instruction reference element 1
.	
.	
.	
X-X+1	Instruction reference element N

Note: An instruction reference element has the following formats:

If the initial value appendage modifier equals 0, the basic format is:

Bytes 0-1: Instruction stream reference; indirect or direct reference to an instruction number

- If bit 0 is 0, then bits 1-15 specify an indirect reference that contains the ODV number of a branch point that specifies the instruction number to be referenced.

- If bit 0 is 1, then bits 1-15 specify a direct reference that contains the instruction number to be referenced.

If the initial value appendage modifier equals 1, the basic format is:

Bytes 0-1: Instruction stream reference; indirect or direct reference to an instruction number

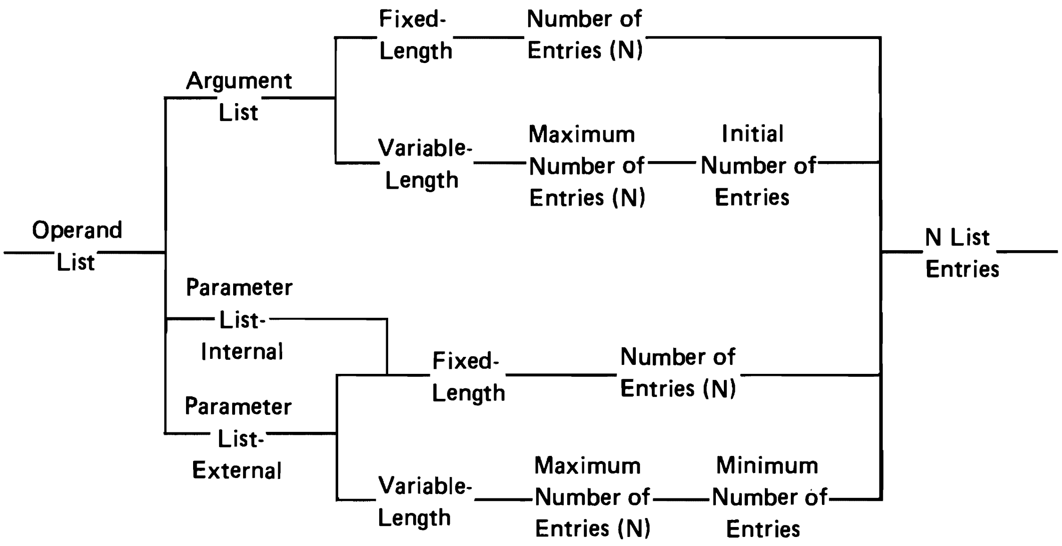
- If bit 0 is 0, then bits 8-23 specify an indirect reference that contains the ODV number of a branch point that specifies the instruction number to be referenced.

- If bit 0 is 1, then bits 8-23 specify a direct reference that contains the instruction number to be referenced.

Bits 1-7 are reserved (binary 0).

Operand List

Attribute Combinations



ODV Format

Bits	Meaning
0-3	Object type 0101 = Operand list
4	OES present 1 = OES is always present because it contains the operand list entries.
5	Argument list 0 = Not used as argument list 1 = Argument list (bits 6-7 must be binary 0)
6-7	Parameter list (if not binary 0, bit 5 must be binary 0) 00 = Not parameter list 01 = Reserved 10 = Internal parameter list 11 = External parameter list
8	Length attribute 0 = Variable-length 1 = Fixed-length
9-15	Reserved (binary 0)
16-31	OES offset OES offset = OES is always present

OES Format

OES Header

Bits	Meaning
0-6	Reserved (binary 0)
7	Initial value present 1 = Initial value is required

Initial Value Appendage

Bytes **Meaning**

0-3 Number (N) of list elements, where 1 c N c 255

- For fixed-length lists (argument or parameter)

Bytes	Meaning
0-1	Number (N) of elements, where 1 c N c 255
2-3	Reserved (binary 0)

- For variable-length lists

Bytes	Meaning
0-1	Maximum number (N) of elements that the list can contain, where 1 c N c 255
2-3	For argument lists, the initial number (M) of elements to be passed on a Call External or Transfer Control instruction, where 0 c M c N For parameter lists, the minimum number (M) of elements to be received on entry, where 0 c M c N

4-5 ODT reference 1

.
.
.

X-X+1 ODT reference N
N elements are required.

Notes:

1. An operand list cannot be both an argument list and a parameter list.
2. Argument lists referenced on Call Internal instructions must be fixed length.
3. Parameter lists referenced by internal entry points must be fixed-length internal parameter lists.
4. Internal parameter lists and argument lists used on internal calls can only be fixed-length.
5. The same object cannot appear in more than one parameter list (internal or external) in a program template.
6. All the ODT entries for the elements of an operand list must appear before the ODT entry for that operand list.
7. Variable-length lists must define ODT references for every entry in the list.
8. Objects referenced in a parameter list must have the parameter attribute.

Constant Data Object

Attribute Combination

—Constant——— Scalar————Value
Data Attributes
Object

ODV Format

Bits **Meaning**

0-3 Object type
0110 = Constant data object

4 OES present
0 = OES is not present (value in bits 8-15).
1 = OES is present because the value does not fit in bits 8-15, and the system default initial value is not used.

5 0 = No system default initial value
1 = Use system default initial value
– Numeric 0 for binary, packed, zoned, or floating-point
– Blank character value (hex 40) for character

6 Value in bits 8-15
0 = Value not in 8-15 in OES, or system default value is to be used.
1 = Value to be propogated in each byte is in bits 8-15, and scalar type is character.

7 Reserved (binary 0)

8-15 Value specification

- If bit 6 is 1, then this byte contains a value to be given to each byte in the constant.
- If bit 6 is 0, then:

Bits **Meaning**

8-12 Reserved (binary 0)

13-15 Scalar type
000 = Binary
001 = Floating-point
010 = Zoned decimal
011 = Packed decimal
100 = Character
101-111 = Reserved

ODV Format (Continued)

Bits **Meaning**

16-31 OES offset or scalar length

- If bit 4 of the ODV is 1 (OES is present), then bits 16-31 represent the offset to the OES entry for this object.
- If bit 4 of the ODV is 0 (OES is not present), bits 16-31 represent the scalar length of the object.

If binary, then:

Bits 16-31: Precision
Hex 0002 = 2
(binary only)
Hex 0004 = 4
All others reserved

If floating-point, then:

Bits 16-31: Precision
Hex 0004 = 4
Hex 0008 = 8
All others reserved

If zoned or packed decimal, then:

Bits **Meaning**

16-23 Digits (D) to the right of assumed decimal point, where 0 c D c T

24-31 Total digits (T) in field, where 1 c T c 31

If character string scalar, then:

Bits 16-31: String length (L), where
1 c L c 32 767

OES Format

OES Header

Bits	Meaning
0	Reserved (binary 0)
1	Length information present 1 = Scalar length information (required if there is an OES)
2-5	Reserved (binary 0)
6	Value present 1 = Value is present (required if there is an OES)
7	Replications present in OES 0 = No replications in initial value 1 = Replications in initial value (bit 6 = 1)

Length Information Appendage

Bytes	Meaning
0-1	Scalar length

- If binary, then:

Bytes 0-1:	Precision Hex 0002 = 2 (binary only) Hex 0004 = 4 All others reserved

- If floating-point, then:

Bytes 0-1:	Precision Hex 0004 = 4 Hex 0008 = 8 All others reserved

- If zoned or packed decimal, then:

Bytes	Meaning
0	Digits (D) to the right of assumed decimal point, where 0 c D c T
1	Total digits (T) in field, where 1 c T c 31

- If character string scalar, then:

Bytes 0-1:	String length 0 = Length (L) beyond 2047, where 1 c L c 2047

Value Appendage

Bytes 0-L: Value in format and length as determined by constant data object scalar type.

- For noncharacter scalars, a value of the proper size and format is required; for example, a 2-byte binary value is required for a 2-byte binary data object.

- For character strings, if the replication attribute specified in the OES is binary 1, the value must consist of components of the following form:

  2 bytes: Number of replications of associated value

  2 bytes: Length (L) of associated value

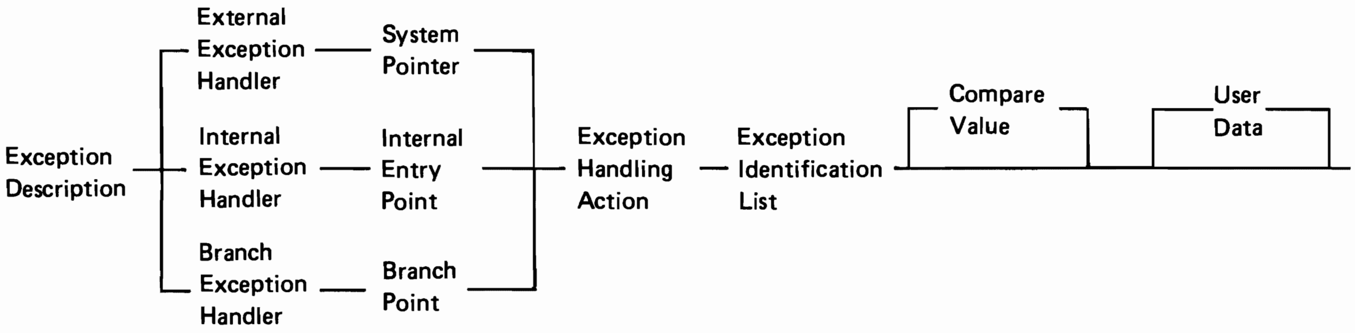
  L bytes:Associated value

  The total number of bytes specified through all replications must equal the length of the string (1 to 2047).

If the replication attribute is binary 0, the value for a character constant view is a byte string of length equal to the object length.

Exception Descriptions

Attribute Combinations



ODV Format

Bits	Meaning
0-3	Object type 0111=Exception description
4	OES present 1 = OES present (required for exception description)
5	Return exception data 0 = Exception data is returned 1 = Exception data is not returned
6-7	Reserved (binary 0)
8-9	Exception handler type 00 = External entry point 01 = Internal entry point 10 = Internal branch point 11 = Reserved
10-12	Exception handling action 000 = Do not handle–ignore occurrence of exception and continue processing. 001 = Do not handle–continue search for another exception description to handle the exception. 010 = Do not handle–continue search for an exception description by resignaling the exception to the immediately previous invocation. 011 = Reserved. 100 = Defer handling–save exception data for later exception handling. 101 = Pass control to the specified exception handler. 110-111 = Reserved.
13-15	Reserved (binary 0)
16-31	OES offset (required)

OES Format

Bits	Meaning
0	Target 1 = Present (always required for exception description)
1	Target appendage modifier 0 = Target appendage for a branch point exception handler has the basic format. 1 = Target appendage for a branch point exception handler has the extended format.
2-4	Reserved (binary 0)
5	Compare value 0 = Compare value not present 1 = Compare value present
6	User data 0 = User data not present 1 = User data present
7	Exception identifications 1 = List of exception identifications (always present)

Target Appendage

For external and internal entry point exception handlers the target appendage modifier must be 0. The appendage has the following form:

Bytes 0-1: ODT reference to
- Pointer data object if the exception handler is an external entry point. This pointer data object must be either in the automatic or static storage of this program and must be directly referenced.
- Internal entry point if the exception handler is an internal entry point.
- Branch point if the exception handler is an internal branch point.

For branch point exception handlers the target appendage has the following form:

If the initial value appendage modifier equals 0, the basic format is:

Bytes 0-1:	Instruction stream reference; indirect or direct reference to an instruction number. • If bit 0 is 0, then bits 1-15 specify an indirect reference that contains the ODV number of a branch point that specifies the instruction number to be referenced. • If bit 0 is 1, then bits 1-15 specify a direct reference that contains the instruction number to be referenced.

If the initial value appendage modifier equals 1, the basic format is:

Bytes 0-1:	Instruction stream reference; indirect or direct reference to an instruction number. • If bit 0 is 0, then bits 8-23 specify an indirect reference that contains the ODV number of a branch point that specifies the instruction number to be referenced. • If bit 0 is 1, then bits 8-23 specify a direct reference that contains the instruction number to be referenced. Bits 1-7 are reserved (binary 0).

User Data Appendage

Bytes 0-1	ODT reference to either – Pointer data object – Scalar data object This data object must be in either the automatic or the static storage of this program and must be directly referenced.

Compare Value Appendage

Bytes	**Meaning**
0-1	Compare value length (maximum value of 32)
2-N	Compare value

Exception Number Appendage

Bytes	**Meaning**
0-1	Number (N) of exception numbers
2-(2n + 1)	N 2-byte exception numbers

Notes:

1. A pointer or scalar data object identified by the exception description (external exception handler or user data) must appear before the ODT entry for the exception description.
2. The exception descriptions are searched in the same order as they appear in the ODT when an exception has been signaled. Because of this, the first exception description that meets the conditions of the exception directs subsequent execution.

References to OES Offsets Greater Than 64 K - 1

ODV Format (References to OES Offset Greater Than 64 K - 1)

Bits	Meaning
0-3	Object type 1111=References to OES greater than 64 K - 1 (65 535)
4-7	Reserved (binary 0)
8-31	OES offset (3 bytes)

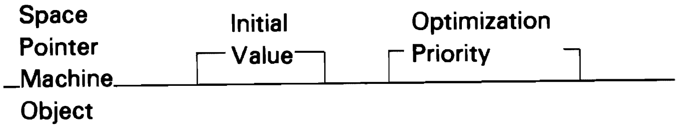
OES Format (Reference to OES Offset Greater Than 64 K - 1)

OES Header

Bytes	Meaning
0-1	First 2 bytes of standard ODV entry for this object
2	OES header for this object
3-N	OES appendages for this object

Space Pointer Machine Object

Attribute Combinations



ODV Format

Bits	Meaning
0-3	Object type 1000=Space pointer machine object
4	OES present 0 = OES is not present. 1 = OES is present because an initial value, or optimization priority is specified.
5-11	Reserved (binary 0)
12-15	Pointer type 0001 = Space pointer (required)
16-31	OES offset • If bit 4 of the ODV is 1 (OES is present), then bits 16-31 represent the offset to the OES entry for this object. • If bit 4 of the ODV is 0 (OES is not present), bits 16-31 contain a value of binary zero.

OES Format

OES Header

Bits	Meaning
0-4	Reserved (binary 0)
5	Initial value present 0 = Initial value is not present 1 = Initial value is present
6	Optimization priority value present 0 = Priority value is not present 1 = Priority value is present
7	Reserved (binary zero)

Initial Value Appendage

Bytes	Meaning
0-1	ODV number of a scalar or pointer data object that is direct or defined on direct.

Optimization Priority Appendage

BytesMeaning	
0	Optimization priority value. Hex FF = Highest priority . . . Hex 00 = Lowest priority
1-3	Reserved (binary 0)

This attribute should be used with caution. The beneficial effects of the normal machine optimizations can potentially be negated through incorrect prioritization of pointers relative to their influence on a program's performance.

Normally, optimizations are performed on pointers according to a priority established by a machine analysis of pointer usage within the program. The optimization priority attribute allows a space pointer machine object to be given a priority attribute which overrides the normal prioritization of space pointer machine objects performed by the machine when programs are created. Through this facility, space pointer machine objects can be specified as being of high optimization priority even though the analysis performed by the machine would determine the space pointer machine object to be of low usage within the program. Specifying a priority value indicates that the pointer is of higher priority than those with a lower value or those for which no value is specified. A particular priority value can be specified for multiple pointers to indicate that they are of equal priority relative to each other.

The number of space pointer machine objects for which special optimizations can be done is quite low, normally 2 or 3 (maximum of 8) depending on the particular program. The number of space pointer machine objects that can be specified in the program is limited by the amount of storage available for them in the internal work space associated with the program.

Space pointer machine objects are allowed in the ODV component of the program template only as an optimization option. This is because of the additional processing overhead necessary to provide support for them during program creation.

If no priority attribute is specified, the normal prioritization of space pointer data objects and space pointer machine objects controls the optimizations performed on them. When the priority attribute is specified, the effect it may have on the optimizations being performed is dependent upon the particular implementation of the machine. A beneficial effect may only be realized for some of the pointers carrying the attribute due to constraints on the optimizations which can be performed for a particular machine.

Appendix A. Instruction Summary

This appendix provides an abbreviated format of all the instructions. The instructions are listed alphabetically by instruction mnemonic.

The summary list includes the following items for each instruction.

- *Operation Description*–The name of the instruction.
- *Mnemonic*–The mnemonic assigned to the instruction.
- *Operation Code*–The 2-byte hexadecimal operation code assigned to the instruction. If an instruction allows any optional forms a bit value of 0 is specified for those positions. All instructions assume a bit value of 0 for the branch target bit.
- *Number of Operands*–The number of operands (excluding the extender) in the instruction.
- *Extender*–A description of the use of the extender field.
- *Operand Syntax*–The objects allowed as operands in the instruction.
- *Resulting Conditions*–The conditions that can be set at the end of the standard operation in order to perform a conditional branch or set a conditional indicator.
- *Optional Forms*–A notation for the optional forms that are allowed for the computational instructions.

Note: This summary list can also be used as an index to identify the page where a complete description of each instruction can be found in this manual. The page number is the last item included with each instruction in this summary.

The following paragraphs further describe the summary list format of the last five items in the previous list.

Number Of Operands

Certain computational instructions allow a variable number of operands and are identified in the summary list by the following form:

number + B

The number defines the number of fixed operands. The B indicates the existence of variable operands (branch targets or indicator operands). A pair of braces around the letter indicates that the variable operands are optional.

Extender Usage

Instructions that use an extender field have a brief description of the use of the extender. Hyphens indicate that the extender is not used. Brackets indicate that the extender is optional. The abbreviation BR/IND is used to mean branch or indicator options. The extender field defines the use of the branch or indicator operands with respect to the resulting conditions of the instruction.

Resulting Conditions

Resulting conditions are the status result of the operation that is used for determining a branch target, if any.

The following conditions are indicated in the instruction summary.

P, N, Z	Positive, negative, zero
Z, NZ	Zero, not zero
H, L, E	High, low, equal
E, NE	Equal, not equal
P, Z	Positive, zero
H, L, E, U	High, low, equal, unequal
Z, O, M	Zero, ones, mixed
[N]Z[N]C	Zero and no carry, not zero an no carry, zero and carry, not zero and carry
S, NS	Signaled, not signaled
DE, I	Exception deferred, exception ign
DQ, NDQ	Dequeued, not dequeued

Optional Forms

All instructions are classified as computational or noncomputational format. The format determines how the operation code is interpreted and whether optional forms of the instruction are allowed. (See *Instruction Format* in *Chapter 1. Introduction*).

Certain computational instructions allow optional forms. The following optional forms can be specified:

- *B (Branch Form)*–The resulting conditions of the operation are compared with the branch options specified in the extender field. If one of the options is satisfied, a branch is executed to the branch target corresponding to the branch option.

- *I (Indicator Form)*–The resulting conditions of the operation are compared with the indicator options specified in the extender field. If one of the options is satisfied, the indicator corresponding to that option is assigned a value of hex F1. The other indicators referred to by the operation are assigned a value of hex F0.

- *S (Short Form)*–The operand that acts as a receiver in the instruction can also be one of the source operands.

- *R (Round Form)*–If the result of the operation is to be truncated before being placed in the receiver, rounding is performed.

INSTRUCTION STREAM SYNTAX

In this instruction summary, the following metalanguage is used to describe the machine interface instruction set operand syntax.

Metasymbol	Meaning
{ }	Choose from a series of alternatives
[ ]	Enclose an optional entry or entries
|	OR – used to separate alternatives
.N.	Repeat previous entry, up to N times
::=	Is defined as – define a metavariable Metavariable ::= Metadefinition
DESC-{ }	Description of a metavariable in English

Notes:

1. Some of the computational op codes require an extender field while on other op codes an extender field is optional. Some computational op codes may be optionally short, or round. When extender fields or different instructional forms are present, the second digit of the op code changes:

Extender and/or Form	Second Digit of Op Code
Short	1
Round	2
Short, round	3
Indicator	8
Indicator, short	9
Indicator, round	A
Indicator, short, round	B
Branch	C
Branch, short	D
Branch, round	E
Branch, short, round	F

2. If an instruction is the target of a branch instruction, then the third bit of the op code is turned on.

Program Object Definitions

ARG-LIST ::= DESC-{operand list which defines an argument list}

B-ARRAY ::= DESC-{array of binary variables}
B-PT ::= DESC-{branch point}
BIN ::= DESC-{binary}
BIN[N] ::= DESC-{binary object with precision N}
BT ::= DESC-{instruction number | relative instruction number | instruction pointer | branch pointer | IDL element | null}

C-ARRAY ::= DESC-{array of character string variables}
CHAR ::= DESC-{character string which is either variable or constant}
CHAR[N] ::= DESC-{string at least N bytes long}
CHARV ::= DESC-{char variable}
CHARC ::= DESC-{char constant}

D-PTR ::= DESC-{data pointer}

EXCP-DESC ::= DESC-{exception description}

F-BT ::= DESC-{instruction number | relative instruction number | branch point}
F-P ::= DESC-{floating-point value}

IDL ::= DESC-{instruction definition list}
IT ::= DESC-{char|numeric variable used as an indicator target}
I-ENT PT ::= DESC-{internal entry point}
I-PTR ::= DESC-{instruction pointer}

NULL ::= DESC-{indicates a null operand [X'0000']}
NUMERIC ::= DESC-{binary | zoned | packed | numeric scalar}
N-ARRAY ::= DESC-{array of numeric variable}

OP-LIST ::= DESC-{operand list}

PROCESS ::= DESC-{character string that names a process}
PTR ::= DESC-{a 16-byte, 16-byte-boundary-aligned pointer element}
P-ARRAY ::= DESC-{an array of 16 bytes, 16-byte-boundary-aligned pointer(s)}

SPDO ::= DESC-{space pointer data object}
S-PTR ::= DESC-{system pointer}
SPP ::= DESC-{space pointer}
SPP-ARRAY ::= DESC-{an array of space pointer variables}

Notes:
1. NUMERIC, CHAR, and BIN may be followed by the special characters S, C, V. These characters further qualify the object as being scalar, constant or variable, respectively.
2. All array objects are variable.

System Object Declarations

AG ::= DESC-{S-PTR that addresses an access group}
ACTV ENTRY ::= DESC-{SPP that addresses an activation}

CD ::= DESC-{S-PTR that addresses a controller description}
CONTEXT ::= DESC-{S-PTR that addresses a context}
CTR ::= DESC-{S-PTR that addresses a counter}
CURSOR ::= DESC-{S-PTR that addresses a cursor}

DATA SPACE ::= DESC-{S-PTR that addresses a data space}
DS-INDEX ::= {S-PTR that addresses a data space index}

INDEX ::= DESC-{S-PTR that addresses an index}

LUD ::= DESC-{S-PTR that addresses a logical unit description}

ND ::= DESC-{S-PTR that addresses a network description}

PCS ::= DESC-{S-PTR to process control space}
PROGRAM ::= DESC-{S-PTR that addresses a program}

SPACE ::= DESC-{a system pointer pointing to a space object}

QUEUE ::= DESC-{S-PTR that addresses a queue}

USER PROFILE ::= DESC-{S-PTR that addresses a user profile}

Resulting Conditions Definitions

ZC ::= DESC-{zero with carry}

[N]ZC ::= DESC-{[not] zero with carry}

Z[N]C ::= DESC-{zero with [no] carry}

[N]Z[N]C ::= DESC-{[not] zero with [no] carry}

CR ::= DESC-{completed record}

DE ::= DESC-{deferred}

DEN ::= DESC-{denormalized}

DQ ::= DESC-{dequeued}

NDQ ::= DESC-{not dequeued}

ECE ::= DESC-{escape code encountered}

E ::= DESC-{equal}

H ::= DESC-{high}

I ::= DESC-{ignored}

IN ::= DESC-{infinity}

L ::= DESC-{low}

M ::= DESC-{mixed}

N ::= DESC-{negative}

NaN ::= DESC-{symbolic not-a-number}

NE ::= DESC-{not equal}

NRN ::= DESC-{normalized real number}

NS ::= DESC-{not signaled}

NZ ::= DESC-{not zero}

O ::= DESC-{ones}

P ::= DESC-{positive}

RO ::= DESC-{receiver overrun}

S ::= DESC-{signaled}

SE ::= DESC-{source exhausted}

TR ::= DESC-{truncated record}

U ::= DESC-{unequal}

UN ::= DESC-{unordered}

Z ::= DESC-{zero}

INSTRUCTION SUMMARY (Alphabetical Listing by Mnemonic)

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Activate Cursor	ACTCR	0402	2	–	CURSOR, {SPP | NULL}	–	–	16-1
Activate Program	ACTPG	0212	2	–	{ACTV ENTRY | PROGRAM},PROGRAM	–	–	9-1
Add Logical Character	ADDLC	1023	3+[B]	[BR | IND]	CHARV, CHARS.2., [BT.4. | IT.4.]	[N]Z[N]C	[B | I, S]	2-1
Add Numeric	ADDN	1043	3+[B]	[BR | IND]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S, R]	2-2
Add Space Pointer	ADDSPP	0083	3	–	SPP.2., BINS	–	–	4-1
And	AND	1093	3+[B]	[BR | IND]	CHARV, CHARS.2., [BT.3. | IT.3.]	Z, NZ	[B | I, S]	2-4
Apply Journal Changes	APYJCHG	05AA	2	–	SPP, SPP	–	–	20-1
Branch	B	1011	1	–	BT	–	–	2-6
Compute Array Index	CAI	1044	4	–	BINV, BINS.3.	–	–	2-25
Call Internal	CALLI	0293	3	–	I-ENT PT, {ARG LIST | NULL}, I-PTR	–	–	9-9
Call External	CALLX	0283	3	–	PROGRAM, {ARG LIST | NULL}, {IDL | NULL}	–	–	9-5
Cancel Event Monitor	CANEVTMN	03D1	1	–	CHARS[48]	–	–	15-1
Cancel Invocation Trace	CANINVTR	0581	1	–	CHARS{4}	–	–	18-1
Cancel Trace Instructions	CANTRINS	0562	2	–	PROGRAM, {SPP | NULL}	–	–	18-2
Concatenate	CAT	10F3	3	–	CHARV, CHARS.2.	–	–	2-37
Cipher	CIPHER	10EF	3	–	SPP, CHARV[32], SPP	–	–	2-7
Cipher Key	CIPHERKY	10FF	3	–	CHARV[8], CHARV[64], CHARS[8]	–	–	2-10
Clear Invocation Exit	CLRIEXIT	0250	0	–	–	–	–	9-10
Compute Math Function Using One Input Value	CMF1	100B	3	–	NUMERICV, CHARS[2], NUMERICS	P, N, Z, UN	[B | I]	2-26
Compute Math Function Using Two Input Values	CMF2	100C	4	–	NUMERICV, CHARS[2], NUMERICS, NUMERICS	P, N, Z, UN	[B | I]	2-33
Compare Bytes Left-Adjusted	CMPBLA	10C2	2+B	BR | IND	{CHARS | NUMERICS}.2., {BT.3. | IT.3.}	H, L, E	{B | I}	2-15
Compare Bytes Left-Adjusted With Pad	CMPBLAP	10C3	3+B	BR | IND	{CHARS | NUMERICS}.3., {BT.3. | IT.3.}	H, L, E	{B | I}	2-17
Compare Bytes Right-Adjusted	CMPBRA	10C6	2+B	BR | IND	{CHARS | NUMERICS}.2., {BT.3. | IT.3.}	H, L, E	{B | I}	2-19
Compare Bytes Right-Adjusted With Pad	CMPBRAP	10C7	3+B	BR | IND	{CHARS | NUMERICS}.3., {BT.3. | IT.3.}	H, L, E	{B | I}	2-21
Compare Numeric Value	CMPNV	1046	2+B	BR | IND	NUMERICS.2., {BT.3. | IT.3.}	H, L, E	{B | I}	2-23
Compare Pointer for Space Addressability	CMPPSPAD	10E6	2+B	BR | IND	{SPP | D-PTR}, {NUMERICV | CHARV | C-N-ARRAY | SPP | D-PTR}	H, L, E, U	{B | I}	4-2
Compare Pointer for Object Addressability	CMPPTRA	10D2	2+B	BR | IND	{D-PTR | SPP | S-PTR | I-PTR}.2.	E, NE	[B | I]	3-1
Compare Pointer Type	CMPPTRT	10E2	2+B	BR | IND	{D-PTR | SPP | S-PTR | I-PTR}, {CHARS[1]NULL}	E, NE	{B | I}	3-3
Compare Space Addressability	CMPSPAD	10F2	2+B	BR | IND	{CHARV | C-ARRAY | NUMERICV | N-ARRAY | PTR | P-ARRAY.2.	H, L, E, U	{B | I}	4-4
Commit	COMMIT	05D2	2	–	S-PTR, SPP	–	–	21-1
Copy Bytes Left-Adjusted	CPYBLA	10B2	2	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}	–	–	2-80
Copy Bytes Left-Adjusted With Pad	CPYBLAP	10B3	3	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}.2.	–	–	2-81
Copy Bytes Overlap Left-Adjusted	CPYBOLA	10BA	2	–	{NUMERICV | CHARV}.2.	–	–	2-82
Copy Bytes Overlap Left-Adjusted With Pad	CPYBOLAP	10BB	3	–	{NUMERICV | CHARV}.2., {NUMERICS | CHARS}	–	–	2-83
Copy Bytes Right-Adjusted	CPYBRA	10B6	2	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}	–	–	2-85
Copy Bytes Right-Adjusted With Pad	CPYBRAP	10B7	3	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}.2.	–	–	2-86
Copy Bytes Repeatedly	CPYBREP	10BE	2	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}	–	–	2-84
Copy Bytes With Left Logical Shift	CPYBTLLS	102F	3	–	{CHARV | NUMERICV}, {CHARS | NUMERICS}, CHARS.2.	–	–	2-77

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Copy Bits With Right Logical Shift	CPYBTRLS	103F	3	–	{CHARV | NUMERICV}, {CHARS | NUMERICS}, CHARS.2.	–	–	2-78
Copy Bytes With Pointers	CPYBWP	0132	2	–	{CHARV | PTR}, {CHARV | PTR | NULL}	–	–	3-4
Copy Data Space Entries	CPYDSE	048F	3	–	CURSOR,SPP,CURSOR	–	–	16-6
Copy Hex Digit Numeric to Numeric	CPYHEXNN	1092	2	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}	–	–	2-87
Copy Hex Digit Numeric to Zone	CPYHEXNZ	1096	2	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}	–	–	2-88
Copy Hex Digit Zone to Numeric	CPYHEXZN	109A	2	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}	–	–	2-89
Copy Hex Digit Zone to Zone	CPYHEXZZ	109E	2	–	{NUMERICV | CHARV}, {NUMERICS | CHARS}	–	–	2-90
Copy Numeric Value	CPYNV	1042	2+[B]	[BR | IND]	NUMERICV, NUMERICS, [BT.3. | IT.3.]	P, N, Z	[B | I, R]	2-91
Create Access Group	CRTAG	0366	2	–	AG, SPP	–	–	13-1
Create Commit Block	CRTCB	05C2	2	–	S-PTR, SPP	–	–	21-3
Create Controller Description	CRTCD	0496	2	–	CD, SPP	–	–	17-1
Create Cursor	CRTCR	044A	2	–	CURSOR, SPP	–	–	16-11
Create Context	CRTCTX	0112	2	–	CONTEXT, SPP	–	–	3-6
Create Dump Space	CRTDMPS	04D2	2	–	S-PTR, SPP	–	–	22-1
Create Duplicate Object	CRTDOBJ	0327	3	–	S-PTR, SPP, S-PTR	–	–	13-4
Create Data Space	CRTDS	045A	2	–	DATA SPACE, SPP	–	–	16-30
Create Data Space Index	CRTDSINX	046A	2	–	DS-INDEX, SPP	–	–	16-36
Create Independent Index	CRTINX	0446	2	–	INDEX, SPP	–	–	6-1
Create Journal Port	CRTJP	05A2	2	–	S-PTR, SPP	–	–	20-7
Create Journal Space	CRTJS	05AE	2	–	S-PTR, SPP	–	–	20-11
Create Logical Unit Description	CRTLUD	049A	2	–	LUD, SPP	–	–	17-9
Create Network Description	CRTND	049E	2	–	ND, SPP	–	–	17-16
Create Process Control Space	CRTPCS	0322	2	–	PCS, SPP	–	–	11-1
Create Program	CRTPG	023A	2	–	PROGRAM, SPP	–	–	8-1
Create Queue	CRTQ	0316	2	–	QUEUE, SPP	–	–	12-1
Create Space	CRTS	0072	2	–	S-PTR, SPP	–	–	5-1
Create User Profile	CRTUP	0116	2	–	USER PROFILE, SPP	–	–	7-1
Convert BSC to Character	CVTBC	10AF	3	[BR | IND]	CHARV, CHARV[3], CHARS	CR, SE, TR	[B | I]	2-38
Convert Character to BSC	CVTCB	108F	3	[BR | IND]	CHARV, CHARV[3], CHARS	SE, RO	[B | I]	2-40
Convert Character to Hex	CVTCH	1082	2	–	CHARV, CHARS	–	–	2-42
Convert Character to MRJE	CVTCM	108B	3	[BR | IND]	CHARV, CHARV[13], CHARS	SE, RO	[B | I]	2-43
Convert Character to Numeric	CVTCN	1083	3	–	NUMERICV, CHARS, CHARS[7]	–	–	2-47
Convert Character to SNA	CVTCS	10CB	3	–	CHARV, CHARV[15], CHARS	SE, RO	[B | I]	2-49
Convert Decimal Form to Floating-Point	CVTDFFP	107F	3	–	F-PS, NUMERICS, NUMERICS	–	–	2-56
Convert External Form to Numeric Value	CVTEFN	1087	3	–	NUMERICV, CHARS, {CHARS[3] | NULL}	–	–	2-58
Convert Floating-Point to Decimal Form	CVTFPDF	10BF	3	–	NUMERICV, NUMERICV, F-PS	–	Round	2-60
Convert Hex to Character	CVTHC	1086	2	–	CHARV, CHARS	–	–	2-62
Convert MRJE to Character	CVTMC	10AB	3	[BR | IND]	CHARV, CHARV[6], CHARS	SE, RO	[B | I]	2-63
Convert Numeric to Character	CVTNC	10A3	3	–	CHARV, NUMERICS, CHARS[7]	–	–	2-67
Convert SNA to Character	CVTSC	10DB	3	–	CHARV, CHARV[14], CHARS	SE, RO, ECE	[B | I]	2-68
Disable Event Monitor	DBLEVTMN	0399	1	–	CHARS[48]	–	–	15-2
Data Base Maintenance	DBMAINT	0413	2	–	{DATA SPACE | DS-INDEX}, CHARS[1],BINS | NULL	–	–	16-50
De-activate Cursor	DEACTCR	0401	1	–	CURSOR	–	–	16-54
De-activate Program	DEACTPG	0225	1	–	PROGRAM | NULL	–	–	9-10
Decommit	DECOMMIT	05D1	1	–	S-PTR	–	–	21-5

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Delete Data Space Entry	DELDSEN	0481	1	–	CURSOR	–	–	16-55
Delete Program Observability	DELPGOBS	0211	1	–	PROGRAM	–	–	8-8
Dequeue	DEQ	1033	3+[B]	[BR | IND]	CHARV, SPP, QUEUE, [BT.2. | IT.2.]	DQ, NDQ	[B | I]	12-5
Destroy Access Group	DESAG	0351	1	–	AG	–	–	13-7
Destroy Commit Block	DESCB	05CD	1	–	S-PTR	–	–	21-6
Destroy Controller Description	DESCD	04A1	1	–	CD	–	–	17-26
Destroy Cursor	DESCR	0429	1	–	CURSOR	–	–	16-57
Destroy Context	DESCTX	0121	1	–	CONTEXT	–	–	3-9
Destroy Dump Space	DESDMPS	04D1	1	–	S–PTR	–	–	22-4
Destroy Data Space	DESDS	0421	1	–	DATA SPACE	–	–	16-58
Destroy Data Space Index	DESDSINX	0425	1	–	DS-INDEX	–	–	16-60
Destroy Independent Index	DESINX	0451	1	–	INDEX	–	–	6-5
Destroy Journal Port	DESJP	05AD	1	–	S-PTR	–	–	20-14
Destroy Journal Space	DESJS	05A1	1	–	S-PTR	–	–	20-15
Destroy Logical Unit Description	DESLUD	04A9	1	–	LUD	–	–	17-27
Destroy Network Description	DESND	04AD	1	–	ND	–	–	17-29
Destroy Process Control Space	DESPCS	0311	1	–	PCS	–	–	11-4
Destroy Program	DESPG	0221	1	–	PROGRAM	–	–	8-9
Destroy Queue	DESQ	0325	1	–	QUEUE	–	–	12-8
Destroy Space	DESS	0025	1	–	S-PTR	–	–	5-4
Destroy User Profile	DESUP	0125	1	–	USER PROFILE	–	–	7-4
Diagnose	DIAG	0672	2	–	BINS, SPP	–	–	19-1
Divide	DIV	104F	3+[B]	[BR | IND]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S, R]	2-92
Divide with Remainders	DIVREM	1074	4+[B]	[BR | IND]	NUMERICV, NUMERICS.2., NUMERICV	P, N, Z	[B | I, S, R]	2-95
Enable Event Monitor	EBLEVTMN	0369	1	–	CHARS[48]	–	–	15-4
Extended Character Scan	ECSCAN	10D4	4	–	B-ARRAY, CHARS, CHARS, CHARS[1]	P, Z, ECE	[B | I]	2-108
Edit	EDIT	10E3	3	–	CHARV, NUMERICS, CHARS	–	–	2-97
End	END	0260	0	–	–	–	–	9-12
Enqueue	ENQ	036B	3	–	QUEUE, CHARS, SPP	–	–	12-9
Ensure Data Space Entries	ENSDSEN	0499	1	–	CURSOR	–	–	16-61
Ensure Object	ENSOBJ	0381	1	–	S-PTR	–	–	13-8
Estimate Size of Data Space Index Key Range	ESTDSIKR	0432	2	–	SPP, S-PTR	–	–	16-62
Exchange Bytes	EXCHBY	10CE	2	–	{CHARV | NUMERICV}.2.	–	–	2-105
Extract Exponent	EXTREXP	1072	2	–	BINV, F-PS	NRN, DEN, IN, NaN	[B | I]	2-111
Extract Magnitude	EXTRMAG	1052	2+[B]	[BR | IND]	NUMERICV, NUMERICS, [BT.3. | IT.3.]	P, Z	[B | I, S]	2-113
Find Independent Index Entry	FNDINXEN	0494	4	–	SPP, INDEX, SPP.2.	–	–	6-6
Grant Authority	GRANT	0173	3	–	{USER PROFILE | NULL}, S-PTR,CHARS[2]	–	–	7-6
Grant-Like Authority	GRNTLIKE	0174	4	–	S-PTR, S-PTR, CHARS[8], {SPP | NULL}	–	–	7-8
Initiate Process	INITPR	0324	4	–	PCS, SPP, {ARG-LIST | NULL}, {SPP | NULL}	–	–	11-5
Insert Dump Data	INSDMPD	04D3	3	–	S-PTR, CHARS[16], SPP	–	–	22-5
Insert Data Space Entry	INSDSEN	0483	3	–	CURSOR, CHARV[7], SPP	–	–	16-66
Insert Independent Index Entry	INSINXEN	04A3	3	–	INDEX, SPP.2.	–	–	6-8
Insert Sequential Data Space Entries	INSSDSE	0487	3	–	CURSOR,SPP,SPP	–	–	16-69

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Journal Data	JRNLD	05B2	2	–	S-PTR, SPP	–	–	20-17
Journal Object	JRNLOBJ	05BA	2	–	S-PTR, SPP	–	–	20-19
Lock Object	LOCK	03F5	1	–	SPP	–	–	14-1
Lock Space Location	LOCKSL	03F6	2	–	SPP,CHARS[1]	–	–	14-4
Materialize Access Group Attributes	MATAGAT	03A2	2	–	SPP, AG	–	–	13-9
Materialize Allocated Object Locks	MATAOL	03FA	2	–	SPP, {S-PTR | SPDO}	–	–	14-6
Materialize Authority	MATAU	0153	3	–	SPP, S-PTR, {USER PROFILE | NULL}	–	–	7-11
Materialize Authorized Objects	MATAUOBJ	013B	3	–	SPP, USER PROFILE, CHARS[1]	–	–	7-13
Materialize Authorized Users	MATAUU	0143	3	–	SPP, S-PTR, CHARS[1]	–	–	7-17
Materialize Commit Block Attributes	MATCBATR	05C7	3	–	SPP, SPP, CHARS[1]	–	–	21-8
Materialize Controller Description	MATCD	04B3	3	–	SPP, CD, CHARS[2]	–	–	17-30
Materialize Cursor Attributes	MATCRAT	043B	3	–	SPP, CURSOR, CHARS[1]	–	–	16-72
Materialize Context	MATCTX	0133	3	–	SPP, {CONTEXT | NULL}, CHARS	–	–	3-10
Materialize Dump Space	MATDMPS	04DA	2	–	SPP, S-PTR	–	–	22-8
Materialize Data Space Record Locks	MATDRECL	032E	2	–	SPP, SPP	–	–	14-8
Materialize Data Space Attributes	MATDSAT	0437	3	–	SPP, DATA SPACE, CHARS[1]	–	–	16-75
Materialize Data Space Index Attributes	MATDSIAT	0433	3	–	SPP, DS-INDEX, CHARS[1]	–	–	16-78
Materialize Event Monitors	MATEVTMN	0379	1	–	SPP	–	–	15-5
Materialize Exception Description	MATEXCPD	03D7	3	–	SPP, EXCP-DESC,CHARS[1]	–	–	10-1
Materialize Instruction Attributes	MATINAT	0526	2	–	SPP, CHARS	–	–	18-3
Materialize Invocation	MATINV	0516	2	–	SPP.2.	–	–	18-7
Materialize Invocation Entry	MATINVE	0547	3	–	CHARV, {CHARV.1. | NULL}, {CHARS.1. | NULL}	–	–	18-10
Materialize Invocation Stack	MATINVS	0546	2	–	SPP, {S-PTR | NULL}	–	–	18-13
Materialize Independent Index Attributes	MATINXAT	0462	2	–	SPP, INDEX	–	–	6-10
Materialize Journaled Object Attributes	MATJOAT	05B6	2	–	SPP, S-PTR	–	–	20-27
Materialize Journaled Objects	MATJOBJ	05A7	3	–	SPP, S-PTR, CHAR[1]	–	–	20-29
Materialize Journal Port Attributes	MATJPAT	05A6	2	–	SPP, S-PTR	–	–	20-21
Materialize Journal Space Attributes	MATJSAT	05BE	2	–	SPP, S-PTR	–	–	20-23
Materialize Logical Unit Description	MATLUD	04BB	3	–	SPP, LUD, CHARS[2]	–	–	17-35
Materialize Machine Attributes	MATMATR	0636	2	–	SPP, CHARS[2]	–	–	19-2
Materialize Network Description	MATND	04BF	3	–	SPP, ND, CHARS[2]	–	–	17-38
Materialize Object Locks	MATOBJLK	033A	2	–	SPP, S-PTR	–	–	14-10
Materialize Program	MATPG	0232	2	–	SPP, PROGRAM	–	–	8-10
Materialize Process Attributes	MATPRATR	0333	3	–	SPP, {PCS | NULL}, CHARS [1]	–	–	11-14
Materialize Process Record Locks	MATPRECL	031E	2	–	SPP, SPP	–	–	14-14
Materialize Process Locks	MATPRLK	0312	2	–	SPP, {PCS | NULL}	–	–	14-12
Materialize Pointer	MATPTR	0512	2	–	SPP, {S-PTR | D-PTR | SPP | I-PTR}	–	–	18-16
Materialize Pointer Locations	MATPTRL	0513	3	–	SPP.2., BINS	–	–	18-19
Materialize Queue Attributes	MATQAT	0336	2	–	SPP, QUEUE	–	–	12-11
Materialize Queue Messages	MATQMSG	033B	3	–	SPP, S-PTR, CHARS.16.	–	–	12-14

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Materialize Resource Management Data	MATRMD	0352	2	–	SPP, CHARS[8]	–	–	13-11
Materialize Space Attributes	MATS	0036	2	–	SPP, S-PTR	–	–	5-6
Materialize Selected Locks	MATSELLK	033E	2	–	SPP, {S-PTR | SPP}	–	–	14-16
Materialize System Object	MATSOBJ	053E	2	–	SPP, S-PTR	–	–	18-21
Materialize User Profile	MATUP	013E	2	–	SPP, USER PROFILE	–	–	7-19
Monitor Event	MNEVT	0371	1	–	SPP	–	–	15-8
Modify Addressability	MODADR	0192	2	–	{CONTEXT | NULL}, S-PTR	–	–	3-13
Modify Automatic Storage Allocation	MODASA	02F2	2	–	{SPP | NULL}, BINS	–	–	9-13
Modify Commit Block	MODCB	05C6	2	–	S-PTR, SPP	–	–	21-11
Modify Controller Description	MODCD	04C3	3	–	CD, SPP, CHARS[2]	–	–	17-43
Modify Dump Space	MODDMPS	04D6	2	–	S-PTR, CHARS[32]	–	–	22-10
Modify Data Space Atrributes	MODDSAT	043A	2	–	S-PTR, SPP	–	–	16-81
Modify Data Space Index Attributes	MODDSIA	047A	2	–	S-PTR, SPP	–	–	16-84
Modify Exception Description	MODEXCPD	03EF	3	–	EXCP-DESC, SPP, CHARS[1]	–	–	10-4
Modify Independent Index	MODINX	0452	2	–	S-PTR, CHARS[4]	–	–	6-13
Modify Journal Port	MODJP	05AB	3	–	{SPP | NULL}, SPP, S-PTR	–	–	20-31
Modify Logical Unit Description	MODLUD	04CB	3	–	LUD, SPP, CHARS[2]	–	–	17-52
Modify Machine Attributes	MODMATR	0646	2	–	SPP, CHARS[2]	–	–	19-12
Modify Network Description	MODND	04CF	3	–	ND, SPP, CHARS[2]	–	–	17-62
Modify Process Event Mask	MODPEVTM	0372	2	–	{BINV[2] | NULL}, {BINS[2] | NULL}	–	–	15-7
Modify Process Attributes	MODPRATR	0337	3	–	{PCS | NULL}, SPP, CHARS[1]	–	–	11-22
Modify Resource Management Controls	MODRMC	0326	2	–	SPP, CHARS[8]	–	–	13-18
Modify Space Attributes	MODS	0062	2	–	S-PTR, BINS	–	–	5-8
Modify System Object	MODSOBJ	053A	2	–	S-PTR, CHARS[16]	–	–	18-23
Modify User Profile	MODUP	0142	2	–	USER PROFILE, SPP	–	–	7-22
Multiply	MULT	104B	3+[B]	[BR | IND]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S, R]	2-115
Negate	NEG	1056	2+[B]	[BR | IND]	NUMERICV, NUMERICS, [BT.3. | IT.3.]	P, N, Z	[B | I, S]	2-117
No Operation	NOOP	0000	0	–	–	–	–	2-119
No Operation and Skip	NOOPS	0001	1	–	–	–	–	2-119
Not	NOT	108A	2+[B]	[BR | IND]	CHARV, CHARS, [BT.3. | IT.3.]	Z, NZ	[B | I, S]	2-120
Or	OR	1097	3+[B]	[BR | IND]	CHARV,CHARS.2.,[BT.3. | IT.3.]	Z, NZ	[B | I, S]	2-122
Reclaim Lost Objects	RECLAIM	0686	2	–	SPP, CHARS[2]	–	–	19-17
Remainder	REM	1073	3+[B]	[BR | IND]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S]	2-123
Rename Object	RENAME	0162	2	–	S-PTR, CHARS	–	–	3-15
Request I/O	REQIO	0471	1	–	SPP	–	–	17-70
Request Path Operation	REQPO	0475	1	–	SPP	–	–	17-78
Reset Access Group	RESAG	0365	1	–	S-PTR	–	–	13-21
Resume Process	RESPR	0386	2	–	{PCS | NULL}, CHARS[1]	–	–	11-27
Retrieve Dump Data	RETDMPD	04D7	3	–	SPP, CHARV[16], S-PTR	–	–	22-12
Retrieve Data Space Entry	RETDSEN	048A	2	–	SPP, CURSOR	–	–	16-89
Retrieve Event Data	RETEVTD	0375	1	–	SPP	–	–	15-12
Retrieve Exception Data	RETEXCPD	03E2	2	–	SPP, CHARS[1]	–	–	10-7
Retrieve Journal Entries	RETJENT	05CA	2	–	SPP, SPP	–	–	20-34

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Retract Authority	RETRACT	0193	3	–	{USER PROFILE | NULL}, S-PTR, CHARS[2]	–	–	7-24
Retrieve Sequential Data Space Entries	RETSDSE	048B	3	–	SPP, CURSOR, SPP	–	–	16-90
Release Data Space Entries	RLSDSEN	048E	2	–	CURSOR, CHARS[1]	–	–	16-87
Remove Independent Index Entry	RMVINXEN	0484	4	–	{SPP | NULL}, INDEX, SPP.2.	–	–	6-14
Resolve Data Pointer	RSLVDP	0163	3	–	D-PTR, {CHARS[32] | NULL}, {S-PTR | NULL}	–	–	3-16
Resolve System Pointer	RSLVSP	0164	4	–	S-PTR, {CHARS[34] | NULL}, {S-PTR | NULL}, {CHARS[2] | NULL}	–	–	3-18
Return From Exception	RTNEXCP	03E1	1	–	SPP	–	–	10-9
Return External	RTX	02A1	1	–	{BINS | NULL}	–	–	9-14
Scale	SCALE	1063	3+[B]	[BR | IND]	NUMERICV, NUMERICS, BINS, [BT.3. | IT.3.]	P, N, Z	[B | I, S]	2-126
Scan	SCAN	10D3	3+[B]	[BR | IND]	{BINV | B-ARRAY}, CHARS.2., [BT.3. | IT.3.]	P, Z	[B | I]	2-128
Scan With Control	SCANWC	10E4	4	[BR | IND]	SPP, CHARV[8], CHARS[4], BT	–	–	2-130
Search	SEARCH	1084	4+[B]	[BR | IND]	{BINV | B-ARRAY}, {N-ARRAY | C-ARRAY}, CHARS | NUMERICS,BINS	P, Z	[B | I]	2-137
Set Access State	SETACST	0341	1	–	SPP	–	–	13-22
Set Argument List Length	SETALLEN	0242	2	–	ARG-LIST, BINS	–	–	9-16
Set Cursor	SETCR	048C	4	–	CURSOR, SPP, CHARV[16], {CHARV | NULL}, {CHARS | NULL}	–	–	16-97
Set Data Pointer	SETDP	0096	2	–	D-PTR, {NUMERICV | N-ARRAY | CHARV | C-ARRAY}	–	–	4-6
Set Data Pointer Addressability	SETDPADR	0046	2	–	D-PTR, {NUMERICV | N-ARRAY | CHARV | C-ARRAY}	–	–	4-7
Set Data Pointer Attributes	SETDPAT	004A	2	–	D-PTR, CHARS[7]	–	–	4-8
Set Invocation Exit	SETIEXIT	0252	2	–	S-PTR, ARG LIST NULL	–	–	9-17
Set Instruction Pointer	SETIP	1022	2	–	I-PTR, F-BT	–	–	2-139
Set System Pointer From Pointer	SETSPFP	0032	2	–	S-PTR, {D-PTR | SPP | S-PTR | I-PTR}	–	–	4-14
Set Space Pointer	SETSPP	0082	2	–	SPP, {CHARV | C-ARRAY | NUMERICV | N-ARRAY | PTR | P-ARRAY	–	–	4-9
Set Space Pointer With Displacement	SETSPPD	0093	3	–	SSP, {CHARV | C-ARRAY | NUMERICV | | N-ARRAY | PTR | P-ARRAY	–	–	4-10
Set Space Pointer From Pointer	SETSPPFP	0022	2	–	SPP, {S-PTR | D-PTR | SPP}	–	–	4-11
Set Space Pointer Offset	SETSPPO	0092	2	–	SPP, BINS	–	–	4-13
Signal Event	SIGEVT	0345	1	–	SPP	–	–	15-14
Signal Exception	SIGEXCP	10CA	2+[B]	[BR | IND]	SPP.2., [BT.2. | IT.2.]	I, DE	[B | I]	10-14
Sense Exception Description	SNSEXCPD	03E3	3	–	SPP.3.	–	–	10-11
Store and Set Computational Attributes	SSCA	107B	3	–	CHARV[5], {CHARS[5] | NULL}, {CHARS[5] | NULL}	–	–	2-140
Store Parameter List Length	STPLLEN	0241	1	–	BINV	–	–	9-18
Store Space Pointer Offset	STSPPO	00A2	2	–	BINV, SPP	–	–	4-15
Subtract Logical Character	SUBLC	1027	3+[B]	[BR | IND]	CHARV, CHARS.2., [BT.3. | IT.3.]	[N]Z[N]C	[B | I, S]	2-143
Subtract Numeric	SUBN	1047	3+[B]	[BR | IND]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S, R]	2-144
Subtract Space Pointer Offset	SUBSPP	0087	3	–	SPP.2., BINS	–	–	4-16
Suspend Object	SUSOBJ	0361	1	–	S-PTR	–	–	13-25
Suspend Process	SUSPR	0392	2	–	{PCS | NULL}, CHARS[1]	–	–	11-29
Terminate Instruction	TERMINST	0342	2	–	S-PTR, CHARS[3]	–	–	11-31
Terminate Machine Processing	TERMMPR	0622	2	–	CHAR[2], {SPP | NULL}	–	–	19-19
Terminate Process	TERMPR	0332	2	–	{PCS | NULL}, CHARS[3]	–	–	11-33

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Test Authority	TESTAU	10F7	3	–	{CHARV[2] | NULL}, {S-PTR | SPDO}, CHARS[2]	–	–	7-26
Test Event	TESTEVT	10FA	2+[B]	[BR | IND]	SPP, {CHARS[48] | NULL}, [BT.2. | IT.2.]	S, NS	[B | I]	15-16
Test Exception	TESTEXCP	104A	2+[B]	[BR | IND]	SPP, EXCP-DESC, [BT.2. | IT.2.]	S, NS	[B | I]	10-18
Trim Length	TRIML	10A7	3	–	NUMERICV, CHARS, CHARS[1]	–	–	2-152
Trace Instructions	TRINS	0552	2	–	PROGRAM, {SPP | NULL}	–	–	18-25
Trace Invocations	TRINV	0551	1	–	CHARS{4}	–	–	18-26
Test Bits under Mask	TSTBUM	102A	2+B	BR | IND	{CHARS | NUMERICS}.2., {BT.3. | IT.3.}	Z, O, M	{B | I}	2-147
Test and Replace Characters	TSTRPLC	10A2	2	–	CHARV, CHARS	–	–	2-146
Unlock Object	UNLOCK	03F1	1	–	SPP	–	–	14-20
Unlock Space Location	UNLOCKSL	03F2	2	–	SPP,CHARS[1]	–	–	14-22
Update Data Space Entry	UPDSEN	0492	2	–	CURSOR, SPP	–	–	16-109
Verify	VERIFY	10D7	3+[B]	[BR | IND]	{BINV | B-ARRAY}, CHARS.2., [BT.3. | IT.3.]	P, Z	[B | I]	2-153
Wait On Event	WAITEVT	0344	4	–	SPP, CHARS, CHARS[8], CHARS[3]	–	–	15-19
Wait On Time	WAITTIME	0349	1	–	CHARS.16.	–	–	11-37
Transfer Control	XCTL	0282	2	–	PROGRAM, {ARG LIST | NULL}	–	–	9-19
Transfer Object Lock	XFRLOCK	0382	2	–	PCS, SPP	–	–	14-18
Transfer Ownership	XFRO	01A2	2	–	USER PROFILE, S-PTR	–	–	7-29
Translate	XLATE	1094	4	–	CHARV, CHARS, {CHARS | NULL}, CHARS	–	–	2-149
Translate With Table	XLATEWT	109F	3	–	CHARV, CHARS, CHARS	–	–	2-150
Exclusive Or	XOR	109B	3+[B]	[BR | IND]	CHARV, CHARS.2., [BT.3. | IT.3.]	Z, NZ	[B | I, S]	2-106

Index

IBM System/38
Functional Reference Manual—Volume 1

GA21-9331-6

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your nearest IBM branch office. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

□ If your comment does not need a reply (for example, pointing out a typing error) check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

□ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): **Comment(s):**

Please contact your nearest IBM branch office to request additional publications.

Name ______________________________

Company or
Organization ______________________________

Address ______________________________

____
City State Zip Code

Phone No. ( ) ______________________________
Area Code

No postage necessary if mailed in the U.S.A.

Fold and tape. **Please do not staple.**

Cut Along Line

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS / PERMIT NO. 40 / ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Development
Department 245
Rochester, Minnesota, U.S.A. 55901

Fold and tape. **Please do not staple.**

Cut Along Line

IBM

IBM®

IBM System/38 Functional Reference Manual—Volume 1 (File No. S38-01) Printed in U.S.A. GA21-9331-6

GA21-9331-6

GA21-9331-6