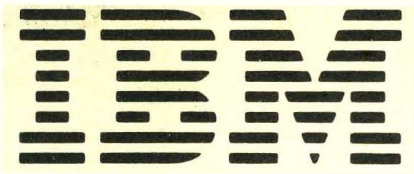
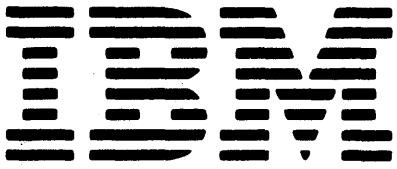


GA21-9331-1

File No. S38-01

IBM System/38

IBM System/38
Functional Reference Manual



GA21-9331-1

File No. S38-01

IBM System/38

IBM System/38
Functional Reference Manual

Second Edition (February 1981)

This is a major revision of, and makes obsolete, GA21-9331-0. This revision contains information about the 3203-5 Printer, secondary SDLC station support, and miscellaneous changes. Because the changes and additions are extensive, this publication should be reviewed in its entirety.

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.

Use this publication only for the purpose stated in the *Preface*.

Publications are not stocked at the address below. Requests for copies of IBM publications and for technical information about the system should be made to your IBM representative or to the branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

This publication describes the System/38 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.

The information in this publication is arranged as follows:

- Chapter 1 describes the basic information for coding instructions.
- Chapters 2 through 19 contain detailed descriptions of all the instructions.
- Chapter 20 contains explanations for the possible exceptions that error conditions may signal.
- Chapter 21 contains detailed descriptions of the events that the user can monitor.
- Chapter 22 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.
- Chapter 23 provides the information to create the objects necessary to support the input/output devices.
- Chapter 24 provides information for communication line connections.
- Chapter 25 provides the information to create the objects necessary to support the load/dump function.
- Appendix A describes the functions used for machine initialization.
- Appendix B provides a summary of all the instructions and an abbreviated format for each instruction.

It is assumed that you have read the *Functional Concepts Manual* in its entirety. The *Functional Concepts Manual* provides information for the machine interface and its functions.

How To Use this Publication

Refer to Chapters 2 through 19 to find the information needed to code the various instructions.

Refer to Chapters 20 through 22 to find detailed specifications for the exceptions, events, and program objects.

Refer to Chapters 23 through 25 to find specific information required to create the various objects necessary to support the input/output devices.

Refer to Appendix A for descriptions of the various functions used for machine initialization.

Refer to Appendix B 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.

Prerequisite Publication

IBM System/38 Functional Concepts Manual, GA21-9330

Related Publications

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

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

Contents

ABI	address bus in
ABO	address bus out
ACTLU	activate logical unit
ACTPU	activate physical unit
ACU	auto-call unit
AIMPL	alternate initial microprogram load
AIPL	alternate initial process load
ALU	arithmetic and logic unit
B	byte
Bin	binary
BOT	beginning of tape
BSTAT	basic status
CA	channel address
CD	controller description
Char	character
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
DS	data space
DSI	data space index
DSR	data set ready
DSTAT	device status
DTR	data terminal ready
EOF	end of file
EOT	end of tape
EOV	end of volume
EPA	encapsulated program architecture
ERP	error recovery procedure
FBR	feedback record
FIFO	first in, first out
FOB	function operation block
FMD	function manager data
IAR	instruction address register
IC	insert cursor
IDL	instruction definition list
I/O	input/output
IOC	input/output controller
IOM	input/output manager
IMPL	initial microprogram load
IMPLA	initial microprogram load abbreviated
IPL	initial program load
K	1024 bytes
L/D	load/dump
LEAR	lock exclusive allow read
LENR	lock exclusive no read
LIFO	last in, first out
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
MISR	machine initialization status record
MPL	multiprogramming level
MSCP	machine services control point
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
PDEH	process default exception handler
PSSA	process static storage area
PU	physical unit
RD	request descriptor
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
SBA	set buffer address
SCS	standard character stream
SDLC	synchronous data link control
SNA	system network architecture
S-PTR	system pointer
SSCP	system service control point
SSD	source/sink data
SSR	source/sink request
TH	transmission header
VAT	virtual address table
VTOC	volume table of contents
WSC	work station controller
XID	exchange identification

Chapter 1. Introduction

This chapter contains the following:

- Detailed descriptions of the System/38 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:

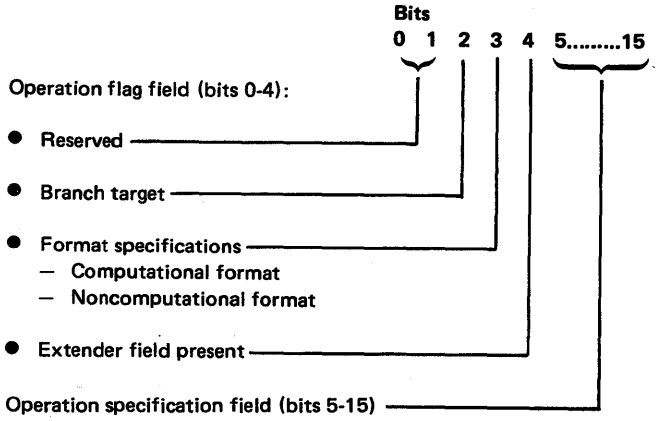
- Operation code
- Operation code extender
- Operand

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.

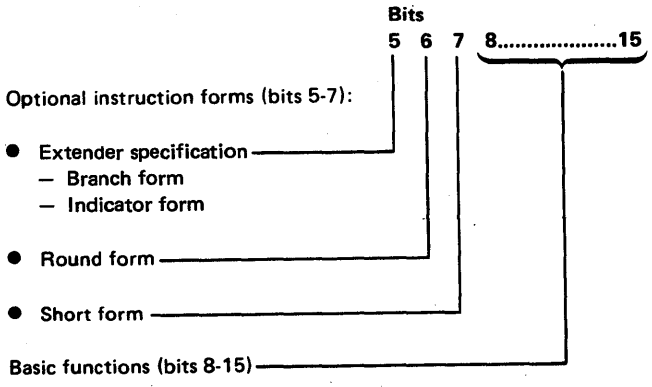
Operation Code Field

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.
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
0010	2	Low, negative, ones, not-zero and no carry, exception ignored
0011	3	Reserved
0100	4	Equal, zero, zeros, zero and no carry, signaled, exception deferred, dequeued, authorized
0101	5	Reserved
0110	6	Reserved
0111	7	Unequal, not-zero and carry
1000	8	Reserved
1001	9	Not high, not positive, not mixed, not-zero and carry
1010	A	Not low, not negative, not ones, not not-zero and no carry
1011	B	Reserved
1100	C	Not equal, not-zero, not zeros, not dequeued, not-zero and no carry, not signaled, not authorized
1101	D	Reserved
1110	E	Reserved
1111	F	Not unequal, not not-zero and carry

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 or low 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).

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. A not low condition with a single branch target accomplishes the same function.

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 2-byte fields that contain either a null operand specification, an immediate data value, or a reference to an ODT object.

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.

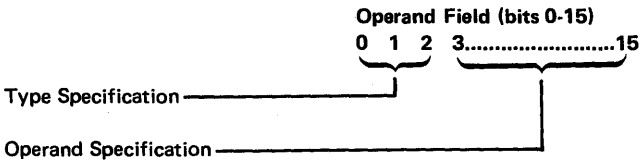
- Explicit base references

  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 (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 illustration shows the type specifications allowed for primary operands and secondary operands. Secondary operands may be simple ODT references or immediate data values.

Operand Function	**Primary Operand**		**Secondary Operand**			
	Type Bits	**Operand**	**Number of Secondary Operand**	**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 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, or an instruction definition list.

  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.

- 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 following are allowed as secondary operands. (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 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 length is not greater than 0 or if the value of 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 (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: • 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.

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 21.

Exceptions

The *Exceptions* sections contain a list of exceptions that can be caused by the instruction. (The detailed description of exceptions is in Chapter 20.) 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, or zoned decimal

- *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)

- *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 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 pointer 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 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 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, a character constant scalar object, or a character scalar defined by a data pointer. Character scalar operands can be in the substring compound operand form, but variable length secondary operands are not always allowed. The secondary operand can be a constant data object or an immediate operand value.

   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 B. 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.

The length of the operation is equal to the length of the longer of the two source operands. The length can be a maximum of 256 bytes. The shorter of the two operands is padded on the right with binary 0's.

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.

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

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	
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	
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 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 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.*

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.

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.

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: The algebraic value of the numeric scalar sum operand 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

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
03 Decimal point alignment		X	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				
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	
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.

Resultant Conditions: The bit values for the bits of the scalar receiver operand is either all zero or 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	
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	
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	
2C Program Execution		
04 Branch target invalid	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 target* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator target* – 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.

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

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	
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	
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 target* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator target* – 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.

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

Exception	Operands 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	
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	
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 target* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator target* – 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.

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

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	
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	
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 target* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator target* – 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.

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

Exception	Operands 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	
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	
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]
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*]:

- *Branch target* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator target* – 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: 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.

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*.

Resultant Conditions: The first compare operand has a higher, lower, or equal numeric 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

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data	X	X		
03 Decimal point alignment	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	
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     Calculates s1+(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 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	
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	

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.

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	
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	

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.

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	
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	

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 02 = Zoned decimal	
Hex 03 = Packed decimal	
– Scalar length	Bin(2)
If binary:	
Length (L)	Bits 0-15
(where L=2 or 4)	
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)

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	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	
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attribute invalid			X	
03 Scalar value invalid			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.

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	
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	
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attribute invalid			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.

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	
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 02	X	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 02 = Zoned decimal
    Hex 03 = Packed decimal
  - Scalar length — Bin(2)
    If binary:
      Length (L) — Bits 0-15
      (where L=2 or 4)
    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.

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		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	
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attribute invalid			X	
03 Scalar value invalid			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.

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	
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	
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.

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		
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
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.

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		
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
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.

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	
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	

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.

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	
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	

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.

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	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	
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.

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		
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	
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.

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	
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	

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.

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	
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	

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.

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	
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	

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.

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	
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	

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 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 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.

Resultant Conditions: The algebraic value of the numeric scalar receiver operand 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

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	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computation			
02 Decimal data		X	
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	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
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 three branch targets (for branch options) or one to three 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.*

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, alignment of the assumed decimal point takes place if the dividend operand is of lesser precision than the precision of the divisor plus the precision of the quotient or if the divisor is of lesser precision than the precision of the dividend minus the precision of the quotient. The dividend is padded on the right with 0's to align it to the precision of the divisor plus the precision of the quotient. The divisor is padded on the right with 0's to align it to the precision of the dividend minus the precision of the quotient.

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 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.

Resultant Conditions: The algebraic value of the numeric scalar quotient operand 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

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
03 Decimal point alignment		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	
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	
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, alignment of the assumed decimal point takes place if the dividend operand is of lesser precision than the precision of the divisor operand plus the precision of the quotient operand or if the divisor operand is less than the precision of the dividend operand minus the precision of the quotient operand. The dividend operand is padded on the right with 0's to align it to the precision of the divisor operand plus the precision of the quotient operand. The divisor operand is padded on the right with 0's to align it to the precision of the dividend operand minus the precision of the quotient operand.

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

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	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0C Compuation					
02 Decimal data		X	X		
03 Decimal point alignment		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	
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 B3. Four of these control characters specify strings of characters to be inserted into the result field under certain conditions; one indicates the end of a string of characters; and the other five indicate that a digit from the source field should be checked and the appropriate action taken.

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

Hex AE — This control character 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. . .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 hex AE (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. . .hex AE negative string. . .hex AE hex B2. . .

Conditional String Character

Hex B0 This control character indicates the start of a conditional string, which consists of any characters delimited by hex AE (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 theresult field.

Conditional string format:

Hex B0 conditional string. . .hex AE

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 hex AE (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. . .hex AE

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

Figure 2-1 (Part 2 of 2). Valid Edit Conditions and Results

Notes:

1. Any character is a valid fill character, including hex AE.
2. Hex AF, hex B1, hex B0, and hex B3 strings must be terminated by hex AE 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.

The following figure indicates the valid ordering of control characters in an edit mask field.

AA, AB, AC, AD

Control Character Y

Control Character X	AF	B0	B1	B2	B3
	0	0	2	2	2	0
AF	0	0	0	0	0	0
B0	1	0	0	2	0	1
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 accomodate the longer of the two floating strings.

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	
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	
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.

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	
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	

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.

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 bit values for the bits of the scalar receiver operand are either all zero or 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	
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	
2C Program Execution				
04 Branch target invalid				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 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 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.

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, 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.

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 receiver operand is either positive 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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computation			
02 Decimal data		X	
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			
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	
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 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 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.*

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.

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.* If significant digits are truncated on the left end of the resultant value, a size 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: The algebraic value of the numeric scalar product 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

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	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				
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	
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 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 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.

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.

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 receiver operand 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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0C Computation			
02 Decimal data		X	
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			
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	
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.

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.

Resultant Conditions: The bit values for the bits of the scalar receiver operand are either all zero or 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	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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.

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 bit values for the bits of the scalar receiver operand are either all zero or 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	
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	
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, alignment of the assumed decimal point takes place if the dividend operand is of lesser precision than the divisor or if the divisor is of lesser precision than the dividend. The dividend is padded on the right with 0's to align it to the precision of the divisor. The divisor is padded on the right with 0's to align it to the precision of the dividend.

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

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
03 Decimal point alignment		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	
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 Form

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 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 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

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.

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 the scaling operation causes nonzero digits to be truncated on the left end of the intermediate field, a size exception is signaled.

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*. If nonzero digits are truncated off the left end of the resultant value, a size 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.

Source Operand Type	**Maximum Value of N**
Binary(2)	$-14 \leq N \leq 14$
Binary(4)	$-30 \leq N \leq 30$
Decimal(P,Q)	$-31 \leq N \leq 31$

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 Condition: The algebraic value of the receiver operand 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

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		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				
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	
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 (fixed-length).

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 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 character string value of the base operand is scanned for occurrences of the character string value of the compare operand.

The base and substring operands must both be character strings. The length of the substring 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 substring operand. The length of the comparisons are equal to the length of the substring 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 substring 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 substring 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 substring 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.

Resultant Conditions: The numeric value(s) of the receiver operand is either 0 or positive.

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	
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	
2C Program Execution				
04 Branch target invalid				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 (fixed-length) 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 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 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.

Resultant Conditions: The numeric value(s) of the receiver operand is either 0 or positive.

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	
08 Argument/Parameter					
01 Parameter reference violation	X	X	X	X	
0C Computation					
08 Length conformance			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	
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

Exception	Operands 1	2	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	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	
2C Program Execution			
04 Branch target invalid		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.

The length of the operation is equal to the length of the longer of the two source operands. The length can be a maximum of 256 bytes. The shorter of the two operands is padded on the right with 0's.

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.

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	
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	
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 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 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.*

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.

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.* If significant digits are truncated on the left end of the resultant value, a size 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: The algebraic value of the numeric scalar difference 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

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	
08 Argument/Parameter				
01 Parameter reference violation	X	X	X	
0C Computation				
02 Decimal data		X	X	
03 Decimal point alignment		X	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				
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	
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.

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	
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	

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 target* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator target* – 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.

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

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	
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	
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 (fixed-length).

Operand 2: Character scalar (fixed-length).

Operand 3: Character scalar or null (fixed-length).

Operand 4: Character scalar (fixed-length).

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.

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	
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	
0A Invalid operand length	X	X	X	X	
0C Invalid operand ODT reference	X	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 (fixed-length).

Operand 3: Character scalar (fixed-length).

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.

Resultant Conditions: The numeric value(s) of the receiver is either 0 or positive.

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

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	
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	
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 B. 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*]:

- *For Branch Form* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *For 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

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	
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	

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 or null	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]:

- *For Branch Form* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *For 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).

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

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

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	
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	
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 pointer, system pointer, instruction pointer, or null.

Description: 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.

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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	

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 0 = Temporary 1 = Permanent	Bit 0
– Space attribute 0 = Fixed-length 1 = Variable-length	Bit 1
– Reserved (binary 0)	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
• Recovery options	Char(4)
– Automatic damaged context rebuild option 0 = Do not rebuild at IMPL 1 = Rebuild at IMPL	Bit 0
– Reserved (binary 0)	Bits 1-32

- 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

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	
02 Object exceeds available space		X	
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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	
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	

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 — Char(1)
    (1 = materialize)
    Reserved (binary 0) — Bits 0-5
    System pointers — Bit 6
    Symbolic identification — Bit 7
  - Selection criteria — Char(1)
    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
- 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.

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 — 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 = Not a member of access group
    1 = Member of access group
  - Reserved (binary 0) — Bit 4-31
- Recovery options — Char(4)
  - Automatic damaged context rebuild option — Bit 0
    0 = Do not rebuild at IMPL
    1 = Rebuild at IMPL
- 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.

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	
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	
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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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.

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	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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.

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			
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	
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 08 = User profile
Hex 0A = Queue
Hex 0B = Data space
Hex 0C = Data space index
Hex 0D = Cursor
Hex 0E = Index
Hex 10 = Logical unit description
Hex 11 = Network description
Hex 12 = Controller description
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.

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

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	
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	
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 B. 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.

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 [4-6]	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	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	

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]:

- *For Branch Form* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *For 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.

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

Exception	Operands 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		
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

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, pointer, or pointer array.

Operand 3 [4-6]:

- *For Branch Form* – Instruction number, relative instruction number, branch point, or instruction pointer.
- *For 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.

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

Exception	Operands 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	
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 M				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

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.

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	
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	

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.

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	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		
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	

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 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 zoned decimal or packed decimal:
    Fractional digits (F) — Bits 0-7
    Total digits (T) — Bits 8-15
    (where $1 \leq T \leq 31$, $0 \leq F \leq T$)
    If character:
    Length (L, where $1 \leq L \leq 32767$)
  - Reserved (binary 0) — Bin(4)

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		
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	
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, pointer, pointer array.

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.

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	
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	
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, pointer, or pointer array.

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.

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	
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	

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.

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

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	
05 Invalid space reference		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	

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.

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

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	
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	
32 Scalar Specification			
01 Scalar type invalid	X	X	

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	
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	

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

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	
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	

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.

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	
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	
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 B. 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 — Bit 0
    - 0 = Temporary
    - 1 = Reserved
  - 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
  - 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
- 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 identifes 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 specifed 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 1) 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

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
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	
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	
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	
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	

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 — Bit 0
    - 0 = Temporary
    - 1 = Permanent
  - Space attribute — Bit 1
    - 0 = Fixed-length
    - 1 = Variable-length
  - Context — Bit 2
    - 0 = Addressability not in context
    - 1 = Addressability in context
  - Access group — Bit 3
    - 0 = Not member of access group
    - 1 = Member of 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 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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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.

Description: The space associated with the system object identified by operand 1 is set to equal the size specified by operand 2. Operand 1 may address any system object that has an associated space with the variable-length attribute.

Operand 2 is a binary value that specifies the total number of bytes that are to be addressable within the space. The extension and truncation of a space is done in multiples of 512 bytes. The size of a space is equal to the current size of the space plus or minus the number of 512-byte blocks necessary to retain a space of at least the requested size.

If the space associated with the object referenced by operand 1 has a fixed size, or if the value of operand 2 is negative, or if the value indicates a size larger than the largest space that can be associated with the object, the space extension/truncation exception is signaled.

Authorization Required

- Object management
  - 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	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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 B. 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 — 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 in context
    - 1 = Insert addressability in context
  - 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
- 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)

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
- Object Control
  - Operand 1 if being replaced
- 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

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
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	
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	
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	
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	

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	
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	
38 Template Specification					
01 Template value invalid			X	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	0003	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	
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	
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 — Bit 0
    - 0 = Temporary
    - 1 = Reserved
  - Space attribute — Bit 1
    - 0 = Fixed-length
    - 1 = Variable-length
  - Context — Bit 2
    - 0 = Addressability not in context
    - 1 = Addressability in context
  - Access group — Bit 3
    - 0 = Not a member of access group
    - 1 = Member of 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 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	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
38 Template Specification			
01 Template value invalid		X	
03 Materialization length exception	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

Exception	Operands 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	
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	
2E Resource Control Limit					
01 User profile storage limit exceeded		X			
38 Template Specification					
01 Template value 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 B. 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 — Bit 0
    1 = Permanent (required)
  - Space attribute — Bit 1
    0 = Fixed-length
    1 = Variable-length
  - 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

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
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	
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	
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	
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	
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.

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 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	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	
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
02 Scalar attributes invalid			X	
03 Scalar value 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

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	
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	
38 Template Specification				
03 Materialization length exception	X			

MATERIALIZE AUTHORIZED OBJECTS (MATAUOBJ)

Op Code (hex)	Operand 1	Operand 2	Operand 3
0153	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 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 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. 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)

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

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

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	
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	
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	
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).

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	
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	
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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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
– Reserved (binary 0)	Bits 8-15

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.

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

- Object control
  - 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	
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	
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	Required authority template

Operand 1: Character(2) scalar or null (fixed-length)

Operand 2: System pointer.

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. 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

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.

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

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	
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
2C Program Execution				
04 Invalid branch target				X
32 Scalar Specification				
01 Scalar type invalid	X	X	X	
03 Scalar 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

Exception	Operands 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	
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	
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	

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 B. 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 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.
  - Reserved (binary 0) — Bits 7-15
- Optimization options — Char(1)
  - Hex 00 = No optimization
  - Hex 80 = Optimization
- 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)
- Number of instructions — Bin(2)
- Number of ODV entries — 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)*

- 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 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.

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.

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.

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.

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.

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 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 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 22. Program Object Specifications.*

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 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.

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.

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.

When a new invocation or activation for a program is allocated, the automatic or static storage areas are initialized to binary 0's. The overhead for this service can be eliminated with two program attribute options which specify that this initialization is not to be done for this program.

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.

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.

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

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

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	
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	
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	
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	

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	
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	

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 Bin(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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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		
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 B. Instruction Summary*.

ACTIVATE PROGRAM (ACTPG)

Op Code (hex)	Operand 1	Operand 2
0212	Program or program activation entry	Program

Operand 1: Space pointer or system pointer.

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 PSSA base entry 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 PASA (process automatic storage area) base entry 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 fields 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.

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	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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).

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 — 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
  - 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.

A space pointer locating the PASA can be materialized by using the Materialize Process 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. 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.

The new invocation entry is updated as follows:

- The previous invocation entry pointer is copied from the most recent invocation entry in the PASA base entry. This pointer locates the calling invocation entry.
- 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 mark counter in the PASA base entry is incremented by 1 and the new value is copied to the invocation mark field.
- 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.

A space pointer (addressing the new invocation entry) is stored in the next invocation entry pointer of the invoking invocation.

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			
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	
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	
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	

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	
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	
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 must appear only as the last instruction of a program; it 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

000C 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
1C Machine-Dependent Exception	
03 Machine storage limit exceeded	X
20 Machine Support	
02 Machine check	X
03 Function check	X

MODIFY AUTOMATIC STORAGE ALLOCATION (MODASA)

Op Code (hex)	Operand 1	Operand 2
02F2	Storage allocation	Modification size

Operand 1: Space pointer 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 pointer in the PASA (process automatic storage area) base entry. If it is necessary to extend the space containing the PASA because of an extension of the current invocation, the instruction implicitly extends this space to contain the additional area.

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.

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	
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	
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.

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 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.

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

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	
08 Invalid operand value range	X	
0A Invalid operand length	X	
0C Invalid operand ODT reference	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

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	
02 Boundary alignment		X	
03 Range		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	
0B Invalid number of operands	X		
0C Invalid operand ODT reference	X	X	
32 Scalar Specification			
03 Scalar value 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	
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	
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. 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 or an event, the return instruction invalid exception is signaled.

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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X		
02 Boundary alignment	X		
03 Range	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	
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 B. 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)

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	
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	
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. Operand 2 cannot be specified as a character constant when operand 3 is not a constant.

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.

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	
03 Range		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	
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.

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	
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	
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 = Reexecute 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 invocations are terminated and 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 one of the following exceptions:

- 0C09 Significance
- 0C0A Size

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.

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

Exception	Operand 1	Other
06 Addressing		
01 Space addressing violation	X	
01 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
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	
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)

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

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	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	
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	
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 — Bit 0
    - 0 = Signal new exception.
    - 1 = Resignal currently handled exception (valid only for an external exception handler).
  - Invoke PDEH (process default exception handler) option — Bit 1
    - 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).
  - Exception description search control — Bit 2
    - 0 = Exception description search control not present
    - 1 = Exception description present
  - 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.

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 destroyed. 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

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 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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

Exception	Operands 1	2	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	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	
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 B. Instruction Summary*.

CREATE PROCESS CONTROL SPACE (CRTPRCS)

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 — 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 the context
    1 = Addressability is inserted into the context
  - Access group — Bit 3
    0 = Not created as member of access group
    1 = Created as member of 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.
  - 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

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
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			
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					
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			
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			
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	
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	

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
- Reserved (binary 0) Bits 11-31

- 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
  - Reserved (binary 0) Bits 44-63

- 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)

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) employs the user profile identified in the template 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, 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, 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.

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.

- 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 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 four 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 defintion 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
- Maximum addressable location in the space containing the PASA (initialized to the highest space address that may be allocated in the space for use by the PASA. This space pointer need not address a currently allocated byte in the space but must address the same space as the next available storage location entry. A process storage limit exceeded exception is signaled if the next available storage location pointer exceeds the maximum addressable location). — 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
- Maximum addressable location in space containing the PSSA (initialized to the highest space address that may be allocated in the space for use by the PSSA. This space pointer need not address a currently allocated byte in the space but must address the same space as the next available storage location entry. A process storage limit exceeded exception is signaled if the next available storage location pointer exceeds the maximum addressable location). — 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, 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.

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
– System pointer for each object lock to be transferred	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	Bit 5
0 = Transfer the current lock count	
1 = Transfer a lock count of 1	
– Reserved (binary 0)	Bit 6
– Entry active indicator	Bit 7
0 = Entry not active (do not use this entry and associated system pointer)	
1 = Entry active (transfer this lock)	

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 invalid lock state 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
  - Initiation phase program
  - Termination phase program
  - Problem phase program
  - Process default exception handler

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	
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					
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	
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	
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 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 — 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 end event option — Bit 3
    - 0 = Time slice expired without entering instruction wait event is not signaled.
    - 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 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 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
- Reserved (binary 0) — Bits 11-31

- 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
- Reserved (binary 0) — Bits 44-63

- 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
  - Reserved (binary 0) — Bits 3-7
  - Internal processing phase — Bits 8-10
    - 001 = Initiation phase
    - 010 = Problem phase
    - 100 = Termination phase
  - Reserved (binary 0) — Bits 11-15

- Process interrupt status (Bit = 1 denotes pending) — Char(2)
  - 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
  - Reserved (binary 0) — Bits 12-15

- 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. 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: This code is supplied by the termination code in operand 2 of the Terminate Process instruction. — Bits 8-23

- Process final termination status — Char(3)
  - Final termination reason: — Bits 0-7
    - 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.
  - Final termination code is assigned in one of the following ways: — Bits 8-23
    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.

    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)
- Time stamp 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 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.

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

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		
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		
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 immediacy of the effect is determined by the attribute being modified. For example, modification of an active process's priority immediately influences an active process's execution. 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

- 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.

The modification control indicators can not be modified through the Modify Process Attributes instruction.

Authorization Required

- Process Control Special Authorization
- Retrieve
  - Contexts referenced for address resolution

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.

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		
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		
32 Scalar Specification					
03 Scalar value invalid			X		
38 Template Specification					
01 Template value 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.

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

Exception	Operands 1	2	3	4	Other
06 Addressing					
01 Space addressing violation	X	X			
02 Boundary alignment	X	X			
03 Range	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				
05 Resume process invalid					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			
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.

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

Exception	Operands 1	2	3	4	Other
06 Addressing					
01 Space addressing violation	X	X			
02 Boundary alignment	X	X			
03 Range	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				
06 Suspend process invalid					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			
32 Scalar Specification					
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 designated process and all the subordinate processes.
    - 1 = Initiate process destruction against the designated process and all process subordinates.
  - 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.

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

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0101 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			
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			
32 Scalar Specification					
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 B. 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 — 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 = Member of access group is not created
    - 1 = Member of access group is created
  - 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(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 65 000 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.

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 65 000 bytes. The maximum size of a queue, excluding its associated space, cannot exceed 64 K bytes. 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

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	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
0A Authorization			
01 Unauthorized for operation		X	X
0E Context Operation			
01 Duplicte 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	
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 Target* – Branch point, instruction pointer, relative instruction number, or absolute instruction number.
- *Indicator Target* – 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: $\neq$, $>$, $<$, $\leq$, and $\geq$. 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.

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. 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:

- Time stamp 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 — Bit 0*
    0 = Access state is not modified.
    1 = Access state is modified.
    When leaving Dequeue wait — Bit 1*
    0 = Access state is not modified.
    1 = Access state is modified.
  - Multiprogramming level option — Bit 2*
    0 = Leave current MPL set at Dequeue wait
    1 = Remain in current MPL set at Dequeue wait
  - Reserved (binary 0) — Bit 3*
  - Actual key to input key relationship (for keyed queue) — Bits 4-7*
    0010: Greater than
    0100: Less than
    0110: Not equal
    1000: Equal
    1010: Greater than or equal
    1100: Less than or equal
- 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.

Operand 3 is a system pointer addressing the queue from which the message is to be dequeued.

Resultant Conditions: Message dequeued, message not dequeued.

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	
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	
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	
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	

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 65 536 bytes.

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

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	
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
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	
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 — Bit 0
    - 0 = Temporary
    - 1 = Permanent
  - Space attribute — Bit 1
    - 0 = Fixed-length
    - 1 = Variable-length
  - Initial context — Bit 2
    - 0 = Addressability not in context
    - 1 = Addressability in context
  - Access group — Bit 3
    - 0 = Not a member of access group
    - 1 = Member of 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.
  - 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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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 B. Instruction Summary*.

CREATE ACCESS GROUP (CRTAG)

Op Code (hex)	Operand 1	Operand 2
0366	Address-ability 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 = Insert addressability in context is not allowed.
    1 = Insert addressability in context is allowed.
  - 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

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	
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	
38 Template Specification			
01 Template value invalid		X	

CREATE DUPLICATE OBJECT (CRTDOBJ)

Op Code (hex)	Operand 1	Operand 2	Operand 3
0327	Address-ability 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 — 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 = Member of access group is not created.
    1 = Member of access group is created.
  - 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 (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.

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. 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

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				
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	
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	
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	
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	
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	

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

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	
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	

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 — Bit 0
    - 0 = Temporary
    - 1 = Reserved
  - Space attribute — Bit 1
    - 0 = Fixed-length
    - 1 = Variable-length
  - Context — Bit 2
    - 0 = Addressability not in context
    - 1 = Addressability 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
  - 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)
- Amount of available space in the access group — 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 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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 04 = Materialize storage transient pool information
  Hex 05 = Materialize storage pool information
  Hex 06 = Storage management counters
  Hex 0A = Materialize MPL control 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.

Transient Storage 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.

Main 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.

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.

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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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 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)
  - Reserved — Char(14)*

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 8.

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.

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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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
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	
32 Scalar Specification			
02 Scalar attribute invalid		X	
03 Scalar value invalid		X	
38 Template Specification			
01 Template value 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.
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 entry length 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.

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	
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	
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

The following instructions can reference objects that have been suspended:

- Destroy (all suspendable objects)
- Grant Authority
- Lock Object
- Materialize Authority
- Materialize Authorized Users
- 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
- Suspend (restricted)
  - Special authorization and object control authority on 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

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	
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	

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 B. 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) — Bit 4*
  - Reserved (binary 0) — Bits 5-7
- 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.

- 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.

  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.

- 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.

- Asynchronous Request – This option allows the requesting process to proceed with execution while the machine asynchronously attempts to satisfy the lock request.

  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. Bit 41 of the parameter represents 1024 microseconds. 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	
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	
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.

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 is modified.

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
  0401 System object
  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	
08 Argument/Parameter			
01 Parameter reference violation	X	X	
10 Damage Encountered			
04 System object			X
44 Partial system object damage			X
IC 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	
2E Resource Control Limit			
02 Process storage limit exceeded			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 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.

Description: The current lock status of the object identified by operand 2 is materialized into the template (operand 1). 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 implicity 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
– Locks 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.

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	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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 activity is materialized for the process issuing the instruction. The materialization identifies each object 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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
38 Template Specification			
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.

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 or waited for by the process) — Char(2)
  - Lock state — Char(1)
    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
  - Status of lock — Char(1)
    Reserved (binary 0) — Bits 0-5
    Implicit lock — Bit 6
    0 = Not implicit lock
    1 = Is implicit lock
    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.

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
  0701 Maximum processor time exceeded

0016 Machine observation
  0101 Instruction reference

0017 Damage set
  0401 System object
  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	
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 state invalid		X	
2A Program creation			
06 Invalid operand type	X	X	
07 Invalid operand attribute	X	X	
08 Invalid operand value rnage	X	X	
0A Invalid operand ldngth	X		
0C Invalid operand ODT reference	X	X	
B0 Resource Control Limit			
02 Process storage limit exceeded			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 (binary 0) — Char(1)*
- 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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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 2 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 — Bit 4
    - 0 = Specific locks now allocated to the process
    - 1 = All locks the process is waiting for asynchronously
  - Reserved (binary 0) — Bits 5-7
  - 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, 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.

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.

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.

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	
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	
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.

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

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
- 0401 System object
- 0801 Partial system object damage set

Exceptions

Exception	Operands 1	2	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	X		X
44 Partial system object damage			X
IA 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		
2E Resource Control Limit			
02 Process storage limit exceeded			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 B. 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 is as follows:

- 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. The operand must be 16-byte aligned.

If the compare value length entry is 0, the compare value entry is ignored. If the event monitor has a compare value qualifier, the compare value length and compare values must be identical to that specified in the monitor.

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
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	
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.

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 has 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. The operand must be 16-byte aligned.

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.

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
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	
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 by the Disable Monitor Event instruction.

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 has 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. The operand must be 16-byte aligned.

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.

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
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	
32 Scalar Specification		
02 Scalar attributes invalid	X	
03 Scalar value invalid	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 (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

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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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 has 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)

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 adapted 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.

  - The monitor domain 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.
    a. Enabled/disabled initial state – This option specifies whether 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.
    b. Signal retention option – This option specifies whether 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.
    c. 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.
    d. Event handler qualifier – This indicator specifies whether 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.
    e. Compare value content – This option denotes the presence or absence of a system pointer in the compare value. The indicator is ignored if the compare value length is 0.

- 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 to 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. 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 21. 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 is 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 scalar 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 is not required and is not present, the event monitor receives signals regardless of the signaled compare value. *Chapter 21. 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	
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	
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. 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 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(24)
- 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 00 = Signal by machine
  - Hex 01 = Signal by Signal Event instruction
- 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.
- 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 .

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.

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	
0A Invalid operand length	X	
0C Invalid operand ODT reference	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 0 = Machine-wide domain 1 = Process domain	Bit 0
– Compare value content 0 = System pointer not present 1 = System pointer present	Bit 1
– 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's signal event control mask with a non-zero 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.

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	
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	
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 format for the template addressed by operand 2 is as follows:

- 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. The operand must be 16-byte aligned.

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) — System pointer

  This attribute is ignored if the event signal is not related to a process action. For example, this attribute is ignored for a timer event.
- 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.

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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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
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	Access state modification option

Operand 1: Space pointer.

Operand 2: Character scalar (fixed-length).

Operand 3: Character(8) scalar (fixed-length).

Operand 4: Character(1) 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. If the number of event monitors is 0, the wait is preempted by any event occurrence monitored by the process.

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. If the number of event monitors is 0, the wait is completed by a signal to any event monitor that does not have an event handler specified.

If the number of event monitors field in operand 4 is 0, the event monitor template in operand 2 is ignored. The number that is specified in the number of event monitors field in operand 4 is the number of times that the event monitor template in operand 2 will be repeated. The format of the operand 2 template is as follows:

- 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 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 two or more event monitors have the same priority, the earliest signaled 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 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 attribute is ignored if the event signal is not related to a process action. For example, this attribute is ignored for a timer event or deadlock detected event.
- 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. 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.

Operand 4 is a character(1) scalar specifying the access state modification option. The operand has the following values and meaning:

- Access state modification option — 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.
  - Reserved (binary 0) — Bit 2-7

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.

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	
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	
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 B. 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 and data space index specified in operand 2 or addressed by the cursor specified in operand 1 are implicitly locked LSRD (lock shared read) by the machine.

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.

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) — Bits 4-7
  - 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)

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. 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 in the data space list. A value of binary 0 causes the cursor to use the data space index over which it was created. If the cursor was not created over a data space index, this field is ignored.

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 LSRD.**
  - **Data space index referenced is 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

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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			
07 Data space index invalid			X
16 Data space not addressed by cursor		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	
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. Data space entries may be selected or omitted from the copy based on values in the entries. Data space entries may be placed into the receiving data space in an order other than their retrieval order on the basis of a set of resequencing specifications. 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)
• Number of exceptions recorded	Bin(2)
• Maximum number of entries	Bin(4)
• Maximum number of exceptions	Bin(2)
• 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, no entries are copied.

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 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 maximum number of exceptions indicates the upper limit on the number of certain types of exceptions to be allowed before terminating the copy. If the record exceptions field is binary 1, this field must contain a value greater than 0, 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 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 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. If either of the cursors is set, 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 any cursor, other than those identified by operand 1 and operand 3 are active over the receiver data space, an exception is signaled.

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.

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

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	
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 exeeded				X
07 Data space index invalid				X
08 Incomplete key description			X	
09 Duplicate key value				X
20 Copy data space entries termination				X
21 Unable to maintain unique key DSI				X
23 Data space index select routine failure				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	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	
38 Template Specification				
01 Template value invalid		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 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
- Mapping templates list pointer — Space pointer
- Data space list pointer — Space pointer
- Length of data space list — Bin(2)
- Cursor attributes — Char(2)
  - Reserved (binary 0) — Bit 0
  - Data space index — Bit 1
    - 0 = No data space index provided
    - 1 = Access through a data space index
  - Replace values — Bit 2*
  - Reserved (binary 0) — Bits 3-7
  - 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)*

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.

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.

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.

The ensure activity attribute allows the cursor user to indicate at creation or activation of the cursor his intent to use the Ensure Data Space Entry instruction.

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.

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 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.

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. 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.

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 a data space is as follows:

• Number of bytes in the mapping template	Bin(4)
• Data space mapping type	Char(2)
– Input mapping type	Char(1)
– Output mapping type	Char(1)
• Input mapping table (optional)	Char(0-n)
– Number of fields described	Bin(2)
– Field specification (repeated for each field in template)	Char(6)
Field location	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 location	Bin(2)
Field attributes	Char(4)
Field type	Bin(2)
Field length	Bin(2)

The number of bytes in the mapping template indicates the total number of bytes included in the number of bytes field, the data space mapping type field, 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

Output Mapping Type

Hex 00 = Direct mapping
Hex 01 = Mapping table provided
Hex 02 = Same as input mapping

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 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.

The input mapping table must be present only if the input mapping type code specifies mapping table provided. Similarly, the output mapping table must be present only if the output mapping type code specifies mapping table provided.

The number of fields entry specifies the number of fields that are to be mapped between the interface buffer and the data space. This entry must equal the number of field specification entries in the associated mapping table.

The field specification entry must be repeated for each field in the template. The order of the field specification entries in the mapping table implicitly specifies the order of the fields in the interface buffer. The field location entry is the relative location of the associated field in the data space established by the Create Data Space instruction. A value of 1 identifies the first field, and a value of n identifies the nth field.

The following field types and specification codes are allowed:

Field Type	Specification Code (hex)
Binary	0000
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
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

Character fields may not be specified as being mapped to or from any of the numeric field types. Character fields are padded with blanks (or truncated) on the right when needed. Numeric fields are truncated or padded with 0's on the left or right as necessary.

The dummy field type indicates the number of bytes to be skipped in the interface buffer when a data space entry is being mapped to or from that buffer. When a dummy field type is specified, the field location entry must be 0.

If the cursor is over a data space index, the key requested (operand 4 of the Set Cursor instruction) must have data attributes that match the output mapping template. The key materialized by the Materialize Cursor Attributes instruction and the key returned (operand 3 of the Set Cursor instruction) have the fields ordered as specified in the data space index (minus the fork characters) and have the key field attributes as specified in the cursor output mapping template. Fork characters are never present in the request or materialized key and are inserted by the machine during construction and maintenance of the index.

If all of the key fields for a data space do not appear in the output mapping template for that data space, then the Set Cursor instruction that has a rule option requiring a search which utilizes a user-supplied key, signals an exception. When key fields are absent from the output mapping template, the Materialize Cursor Attributes and Set Cursor instructions cannot materialize a key.

[1]The number of fractional digits to the right of the decimal point.

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

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	
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			
13 Invalid mapping 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	
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	
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	
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 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
    - 1 = Permanent (required)
  - 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 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)
  - Reserved (binary 1) — 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
  - Reserved (binary 0) — Bits 8-15
- 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)

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.

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. If initial allocation is not specified (binary 0), a default initial allocation and extension allocation are used.

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. The unit return bit (binary 1) indicates that the data portion of the data space resides on the requested auxiliary storage unit. A unit return bit (binary 0) indicates that some of the data space is not on the requested unit. If the unit identification parameter is 0, the unit return bit of 0 is returned.

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 (undeleted) 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. 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 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 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 the Insert Data Space Entry instruction and the Insert Sequential Data Space Entries instruction 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 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
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
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

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	
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	
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 — Bit 0
    1 = Permanent (required)
  - 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 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 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)
  - Reserved (binary 0) — Bits 0-7
  - 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
  - Reserved (binary 0) — Bit 13
  - Duplicate key rules — Bits 14-15
    00 = Unique keys required
    01 = LIFO duplicates permitted
    10 = FIFO duplicates permitted
    11 = Reserved
- Unit identification — Char(1)
- Reserved (binary 0) — Char(1)
- 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 location — Bin(2)
  - Key field attributes — Char(2)
    Reserved (binary 0) — Bits 0-7
    Ordering option — Bit 8
      0 = Ascending sequence
      1 = Descending sequece
    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

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 data space index is created as a permanent object and exists in the machine until explicitly destroyed.

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.

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.

Subsequently, the ordering of the data space pointers in the data space list is used 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 1, 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.

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. 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.

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.

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.

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 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. Decending 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 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.

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 field within the data space entry 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.

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 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.

The selection routine is a program to be invoked each time addressability to a data space entry is to be placed in or removed from the data space index. This routine must satisfy the criteria for a user exit routine (see *Chapter 8. Program Management Instructions*). To ensure addressability at all times, a copy of the routine is made and bound permanently to the data space index during creation of the data space index. The selection routine pointer is required for a Create Data Space Index instruction and is not materialized by a Materialize Data Space Index Attributes instruction.

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.

When an entry's key is to be put into the data space index, the selection routine is given a space pointer that 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 selection routine failure exception is signaled, and the data space entry is neither inserted nor 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.
  - 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

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

Exception	Operands 1	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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			
0B Duplicate key value detected while building a unique data space index		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	
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	
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
0482	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 and 07 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

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 previously assigned ordinal numbers are available for reassignment. Every data space index referencing this data space must be invalid. If the data space was defined with the initial allocation attribute and if contiguous storage was available and allocated to the data space, the contiguous storage is not truncated. If contiguous storage was not allocated initially, the existing storage is released, and an initial allocation, based on the current maximum number of entries, is acquired. If an initial allocation is not specified for the data space, all storage is released, and a default initial allocation is obtained. 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-Active 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. Every data space index defined over the data space must be invalid.

Inserting, updating, and deleting data space entries can occur concurrently with creating a data space index over the data space.

Authorization Required

- Object Management
  - Data space (reset or insert entries option)
  - Data space index (invalidate option)
- Retrieve
  - Contexts referenced for address resolution
- Delete
  - Data space (reset option)
  - 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

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	
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				
04 Data space entry limit exceeded	X			
07 Data space index invalid	X			
0B Duplicate key value detected while building a unique data space index	X			
22 Data space index with selection routine build determination	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	
2E Resource Control Limit				
01 User profile storage limit exceeded	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 changed 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.

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
  - Implicit LSUP lock removed from data space for each locked data space entry

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

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	
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	

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.

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

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	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	
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	
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	

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.

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

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	
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	
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	

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, 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

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	
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	

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 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	
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	

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 unensured 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

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	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	

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.

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 Data Space Entry 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

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	X	
03 Range	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
21 Unable to maintain unique key DSI				X
23 Data space index select routine failure				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			
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	
2E Resource Control Limit				
01 User profile storage limit exceeded				X
32 Scalar Specification				
01 Scalar type invalid				X
02 Scalar attributes invalid		X		
03 Scalar value invalid		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.

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

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	
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
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	
2E Resource Control Limit				
01 User profile storage limit exceeded				X
38 Template Specification				
01 Template value invalid		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 creation template is specified, a similar template is materialized. (See the Create Cursor instruction, earlier in this chapter, for a definition of the template.)

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 0-14
    Cursor addressability set — Bit 15
      0 = Cursor not set
      1 = Cursor set
  - 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 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.

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.

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	
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
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	
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 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 operational statistics.

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, unit return, 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.

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

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.

Deleted entries occupy space in a data space, and the number of deleted entries provides an indication of how much space they occupy in this data space.

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 system pointer is provided for each distinct data space index addressing the specified data space.

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	
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	
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 time stamp and counts; and hex 02 signifies the operational statistics with the resetting of the time stamp 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. 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.
- Access group is binary 0.
- Replace option is binary 0.
- Size of space is set to 0.
- Context pointer is null.
- Access group 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)
- Time stamp of this materialization — Char(8)
- Time stamp acquired from the data space index — Char(8)
- Data space index status — Char(2)
  - Reserved (binary 0) — Bits 0-14
  - Data space index status — Bit 15
    0 = Valid
    1 = Invalid
- 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 entries not addressed by the index — Bin(4)
  - Number of accesses to the data space using this index — Bin(4)

The size of the data space index indicates the total space occupied on auxiliary storage by the data space index. The time stamp of this materialization is the current machine time stamp. The current time stamp is also stored in the data space index if materialization option hex 02 is specified. The time stamp from the data space index is the time stamp stored in the object at creation or at the last materialization with option hex 02 on this data space index. Time stamps 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. If the data space index is invalid, the data space status values are 0.

For each data space addressed by the data space index, the data space status indicates the number of entries in the data space index and the number of accesses to the data space index.

The number of entries for the data space does not include entries that have been omitted by the data space index selection routine. The number of entries not addressed by the index indicates the number of entries omitted by the selection routine. 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.

The number of accesses to the data space is the number of times that the data space index (operand 2) was employed in order to access an entry residing in the specified 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 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.

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	
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	
32 Scalar Specification				
03 Scalar value invalid			X	
38 Template Specification				
03 Materialization length exception	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 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 unlocked; the respective LSUP (lock shared update) implicit locks are removed from the data spaces. If the option field has a value of hex 01, only the first entry in the queue (the entry that has been locked the longest) is unlocked and removed from the queue; the implicit LSUP lock is removed from the data space. No exception is signaled if there are no entries currently in the cursor's locked entry queue.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolutior

- 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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X		
03 Range	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	
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 Set Cursor instruction 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 Set Cursor instruction 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 Set Cursor instruction in addressing the entry (that is, the cursor is over a data space index and a set cursor option other than relative or ordinal was used) and that key has changed since the Set Cursor instruction, 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.

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

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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
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	

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 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 is then placed in the interface buffer (operand 1) in the manner described by the output mapping template specifications 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. The entries are in the interface buffer in the exact order that they were retrieved. Each entry in the interface buffer has up to three 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 mapped data space entry

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 ordinal number — Bit 1
  - Materialize key — Bit 2
  - Reserved (binary 0's) — Bits 3-15
- 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
  - Reserved (binary 0) — Bits 3-15
- Interface buffer position — Bin(2)
- Number of data spaces in restricted search list — Bin(2)
- Data space to be included in a restricted search list (0 to n) — Bin(2)

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. 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 data space and ordinal entry position 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 position field. A value of binary 0 in the materialize key field will result in the key not being mapped into the interface buffer and the key position field being ignored.

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 the first byte of 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. 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 data space and ordinal entry position 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 key for the data space entry is returned in the buffer entry beginning in the position designated by the key position field. This field, if specified, must contain a value greater than or equal to 0 and less than the buffer entry length.
- The data space entry is then presented, as defined by the 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 position field. This field must be provided and must have a value greater than or equal to 0 and less than the buffer entry length.

Note: If the remaining area in the buffer entry is not large enough for the entry 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 retrieved and 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 not positioned indicator is returned with a value of binary 0, then the cursor is set to the last successfully retrieved entry that has been placed in the interface buffer and has been indicated as returned in the interface buffer position field. If the cursor not positioned indicator is returned with a value of binary 1, then the cursor is set to the next sequential entry, but that entry is not returned in the interface buffer and the interface buffer position field does not reflect the data space entry to which the cursor is now set.

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, if requested, and the key, if requested, contain values which identify the data space entry the cursor is set to, but the entry has not been returned in the buffer.

- 1202–Key mapping error

  The cursor is positioned to the last retrieved data space entry. The data space and ordinal entry number, if requested, has been created in the buffer entry, but the key and data space entry is not placed in the buffer entry.

- 120A–End of path

  The number of entries retrieved field designates the number of valid entries retrieved.

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.

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 entry or entries in the data space or data space 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.

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

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	
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
0A End of path				X
19 Invalid rule option				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	
38 Template Specification				
01 Template value invalid			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 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 field count — Char(1)
- 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)

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:	
Rule Option	**Value (hex)**	**Data Space Index**	**Data Space(s) 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 – equai or before	0A	X	
Key – equal	0B	X	
Key – equal or after	0C	X	
Key – after	0D	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 associated with the next key in the data space index after the key currently referenced by the cursor. The entry must have a key value that is different (as qualified by the key field count) from the key value of the current entry.
- Previous unique – 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. The entry must have a key value that is different (as qualified by the key field count) from 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 an entry in the data space indicated by the first entry in the restricted data space search list having the ordinal number specified by the relative or ordinal number field. 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. The number of data spaces in the restricted search list field must have a valid value (from 1 through 32).

- Key operations – In the following key operations, the key field count indicates the number of fields provided in the operand 4 argument. 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. 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.

For a cursor activated directly over data spaces, the meaning of the valid rule options is as follows. (For multiple data spaces, the search continues into the next data space when needed, except when restrict to requested data spaces 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 addressed by the cursor 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 results in an end of path exception.

- Ordinal – The cursor is set to address 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. The number of data spaces in the restricted search list field must be greater than 0.

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
Reserved (binary 0)	Bits 6-7

- 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 = relative

  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.

- Trailing fork characters – This attribute is considered only when the key field count field is used. A value of binary 0 indicates that fork characters following the last key field indicated by key field count are not to be used during the search. A value of binary 1 indicates that the fork characters immediately following the last key field indicated by the key field count 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 or ordinal. They result in entry not found exceptions for relative or ordinal because a 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.

- 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 or ordinal.

- 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.

The control attributes are options that control the status of the entry upon completion of the Set Cursor instruction. The control attributes field has the following format:

- Forced write option — Bit 0
- Reserved (binary 0) — Bits 1-3
- 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.

- 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.

- 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 entry is already locked, a data space entry locked exception is immediately signaled.

- 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 to another process, 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 shared update) lock is applied to the data space. If the implicit lock cannot be obtained, an exception is signaled immediately and the entry is not locked. A lock option value of binary 10 results in a template value invalid exception being signaled.

- 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. 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.

- 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 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 field count designates the number of fields assumed to be in the key value (operand 4) to be used for searching the data space index.

The key field count entry only includes data fields supplied by the user for the Set Cursor instruction; it does not include fork characters. If the key field count is less than the actual number of fields in the key, then a generic key search is performed. A key field count of 0 is legal only if a fork character is defined as the first element of the key. If the key field count is 0, then operand 4 is ignored. The key field count is required when the rule option is any of the key operations or next unique or previous unique.

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.

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.

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 not set (exception condition)
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 set				X
Relative	1. Cursor set	X			
	2. New cursor setting would be set outside of data space bounds.			X	
	3. Cursor not set				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			

Notes:
1. An adjacent data space must not be empty and must contain at least one nondeleted entry unless deleted entry significance is specified.
2. The cursor setting remains unchanged for all exception conditions.

Results of Set Cursor With Data Space Index:

Set Cursor Rule Options to Address a Data Space Index	Current State Explanation before Set Cursor Request	Results of Set Cursor			
		Normal	Entry not found (exception condition)	End of path (exception condition)	Cursor not set (exception condition)
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 set				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

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	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	
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
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
19 Invalid rule option					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				
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	
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			

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.

Any data sensitive mapping error encountered while presenting the modified data space entry to the user exit routine associated with a select/omit data space index referencing the data space not only causes the data space index to be invalidated, but also causes the data space index invalidated and data space entry not addressed by data space index events to be signaled.

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. The implicit LSUP (lock shared update) lock on the data space caused by locked entries is applied only when the number of currently locked entries within this data space to this cursor goes from 0 to 1. This LSUP lock is removed only when the number of the currently locked entries to this cursor from this data space goes from 1 to 0. At the successful completion of the instruction, the updated entry is unlocked and the implicit LSUP lock is removed from the data space. If no entry is locked, then an exception is signaled. Errors in this instruction cause the entry to remain locked.

If a deleted entry is being updated and the addition of this entry causes the maximum number of undeleted entries residing in the data space to be exceeded, the entry is not updated or unlocked and a data space entry limit exceeded exception is signaled. The Data Base Maintenance instruction can then be used to increment the maximum and the update can be reissued.

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

Exceptions

Exception	Operands 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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
04 Data space entry limit exceeded			X
09 Duplicate key value in existing data space entry			X
0D No entries locked	X		
21 Unable to maintain unique key DSI			X
23 Data space index select routine failure			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		
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	
2E Resource Control Limit			
01 User profile storage limit exceeded			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 B. 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 — 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)
- CD definition data — Char(16)
  - CD type — Char(2)
    Hex 0000 = not attached to ND
    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)
CD (station) address	Bit(16)
ND (OU number) address (nonswitched line) (If switched line, binary 0)	Bit(16)
– 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(6)*
– Reserved (binary 0)	Char(10)
• 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	Char(21)
Byte 0	Char(1)
XID format	Bits 0-3
PU 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	Bits 0-1
10 = SDLC	
00 = Other or not applicable	
Switched network	Bit 2
0 = No (nonswitched network or not applicable)	
1 = Yes (switched network)	
Role	Bit 3
0 = Secondary SDLC station	
1 = Primary SDLC station	
Switched network backup feature (on nonswitched network)	Bit 4
0 = No	
1 = Yes	
Data rate select feature	Bit 5
0 = No	
1 = Yes	
Reserved (binary 0)	Bits 6-7
– Reserved (binary 0)	Char(2)
– 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	
Reserved (binary 0)	Bits 4-15
– Delayed contact control Hex 0000 = No Hex 0100 = Yes	Char(2)
– Reserved (binary 0)	Char(12)

- 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(*)

- 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): Station ID sequence for establishing identity of the secondary 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 and the specified ID are 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:
  a. Line discipline: This entry defines the protocol to be used for link level communications. All stations communicating over the link must follow the same protocol at any point in time.
  b. 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.
  c. Role: This entry indicates whether this CD represents a primary or a secondary SDLC station.
  d. 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.
  e. 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 delayed contact control indicator is used by nonswitched or loop 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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* for the details concerning this entry.

XID information area: This entry can be materialized and defines the XID (exchange ID) data area. 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. 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. Additional information about this entry is contained in *Chapter 24. Communications and Locally Attached Work Stations* and *Chapter 23. Source/Sink Specialization and Programming Considerations For Local Devices*.

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

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	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	
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	
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 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)

- 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 — Char(2)
    Hex 0000 = No
    Hex 0100 = Yes
  - Reserved (binary 0) — Char(6)
- State/status definition — Char(16)
  - State change/status field — Char(3)*
  - Reserved (binary 0) — Char(13)
- Session definition data — Char(32)
  - Session information — Char(20)
    Pacing (inbound) — Bin(2)
    Pacing (outbound) — Bin(2)
    RU size (buffer size) — Bin(2)
    Reserved (binary 0) — Bin(2)
    ACTLU required — Char(1)
      Hex 00 = No
      Hex 01 = Yes
    ACTLU parameters — Char(3)
    ACTLU response — Char(8)*
  - Reserved (binary 0) — Char(12)
- Load/dump definition data — Char(16)
  - 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 on materialize — Char(3)*
  - Reserved (binary 0) — Char(9)
- 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) to show the current status of the LUD. This entry can be modified (Modify Logical Unit Description) 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 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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* 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 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. Only the load/dump device indicator and the load/dump operating mode indicators are used on a create instruction. The load/dump pending field can be materialized and is 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 *Chapter 23. Source/Sink Specialization and Programming Considerations For Local Devices* and *Chapter 24. 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 *Chapter 23. Source Sink Specialization and Programming Considerations For Local Devices* and *Chapter 24. Communications and Locally Attached Work Stations.*

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

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	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	
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	
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(3)*
– Reserved (binary 0)	Char(13)

- Line definition data — Char(16)
  - Line definition — Char(10)
    - Line data — Char(4)
      - Line discipline — Bits 0-3
        - 1000 = SDLC
      - 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
      - SDLC role — Bit 7
        - 0 = Primary SDLC station
        - 1 = Secondary SDLC station
      - Reserved (binary 0) — Bits 8-11
      - NRZI — Bit 12
        (non-return-to-zero [inverted])
        - 0 = No
        - 1 = Yes
      - Reserved (binary 0) — Bit 13
      - 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 — Bit 18
        - 0 = Point-to-point
        - 1 = Multipoint
      - Reserved — Bit 19
      - Reserved (binary 0) — Bits 20-3.
    - 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 — Bit 5
        (Far-end modem that requires 2025 hertz answer tone)
        - 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) — Char(2)
      - SDLC address — Bits 0-7
      - Reserved (binary 0) — Bits 8-15
  - Reserved (binary 0) — Char(6)

- Communications initialization data — Char(16)
  - Initialization data — Char(8)*
  - Reserved (binary 0) — Char(8)

- Exchange identification data (binary 0) — Char(16)*

• 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	
Reserved (binary 0)	Bits 4-7
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 only)	Bit 6
0 = No time-out	
1 = 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(6)
SDLC idle state detection timer	Bin(2)
Nonproductive receive timer	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.

- 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.

- Role (primary/secondary) – When this entry is set, 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.

- 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).

- 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 – 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.

- 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 capabiliy.

- 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. 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. The address can be specified in vlaues hex 01 through hex FE in the first byte of this entry. For primary stations, specify hex 00.

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 is uniquely defined for this system when the system is installed, and it contains the exchange identification used by System/38 when acting as an SDLC secondary station on a network. This entry can be materialized.

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.

- 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. 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.

- 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 receiver 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 receiver 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 receiver 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	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 *Chapter 24. Communications and Locally Attached Work Stations*.

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

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	
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	
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	
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	
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	
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	
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	
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	
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		Bit(3)	
4 Diagnostic mode		Bit(1)	
5 Diagnostic active indicator		Bit(1)	
6-7 Reserved		Bit(2)	
• Reserved		Char(10)	
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

Elements Contained in the CD Template (CD Types 00, and 10) for Materialize CD	Element Length	Sub-Element Length	Materialize Option Values
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.

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.

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 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	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	
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		Bit(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		Bit(6)	
3-5		**Reserved**		Char(3)	
• Reserved				Char(10)	
Session definition data			Char(32)		z002
Load/dump definition data			Char(16)		z004
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.

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 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X	X	
02 Boundary alignment	X	X	X	
03 Range	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	
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		Bit(6)	
4-5		**Reserved**		Char(2)	
• Reserved				Char(10)	
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
Reserved group			Char(32)		z010
Specific characteristics			Char (y + 2)		100D
Retry value sets			Char (6y + 2)		z020

Elements Contained in the ND Template (ND Type 00) for Materialize ND	Element Length	Sub-Element Length	Materialize Option Values
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.

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

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	
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	
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.

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		Bit(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		Bit(3)	
	4	Set diagnostic mode		Bit(1)	
	5	Reset diagnostic mode		Bit(1)	
	6-7	Reserved		Bit(2)	
• Reserved				Char(10)	
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 Figure 17-1 (Part 1 of 2) for a description of the states that exist for the CD object and also for the valid state changes that can be made by the Modify Controller Description instruction. Figure 17-1 (Part 2 of 2) shows the valid relationship for modifying other elements in the CD.

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.

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 or loop, 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:

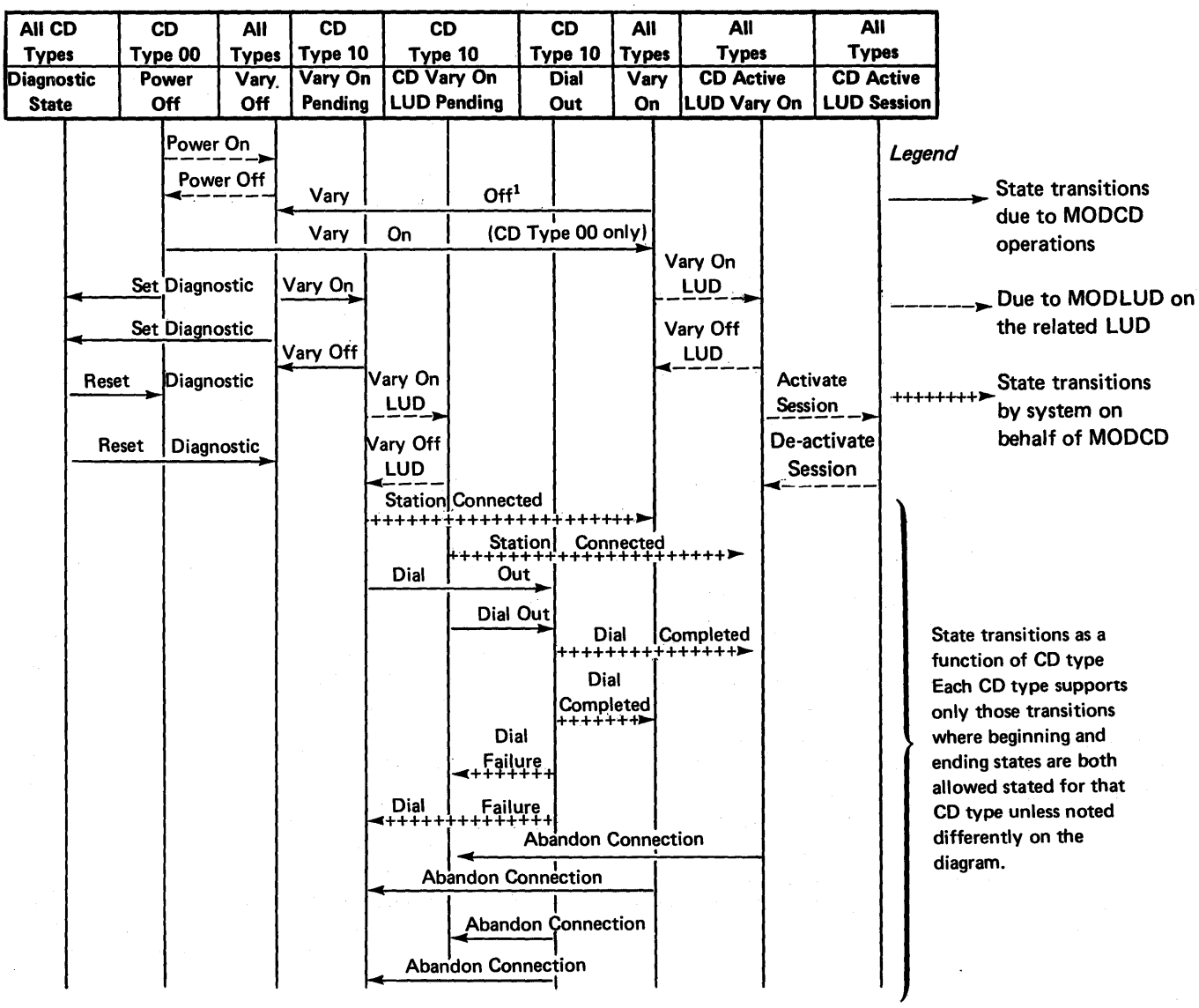
Dial In

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, 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 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 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 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.



[1] For CD type 10 (switched line), a vary off from this state will cause an implicit abandon connection and then a vary off.

Figure 17-1 (Part 1 of 2). CD State Change Rules

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 (Part 2 of 2). CD State Change 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.

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	
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	
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.

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		Bit(7)	
	7	Activate session		Bit(1)	
1		**Commands**			
	0	Suspend session		Bit(1)	
	1	Quiesce session		Bit(1)	
	2	Reset session		Bit(1)	
	3	Deactivate 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		Bit(6)	
3-5		**Reserved**		Char(3)	
• Reserved				Char(10)	
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 Figure 17-2 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.

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, deactivate 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.

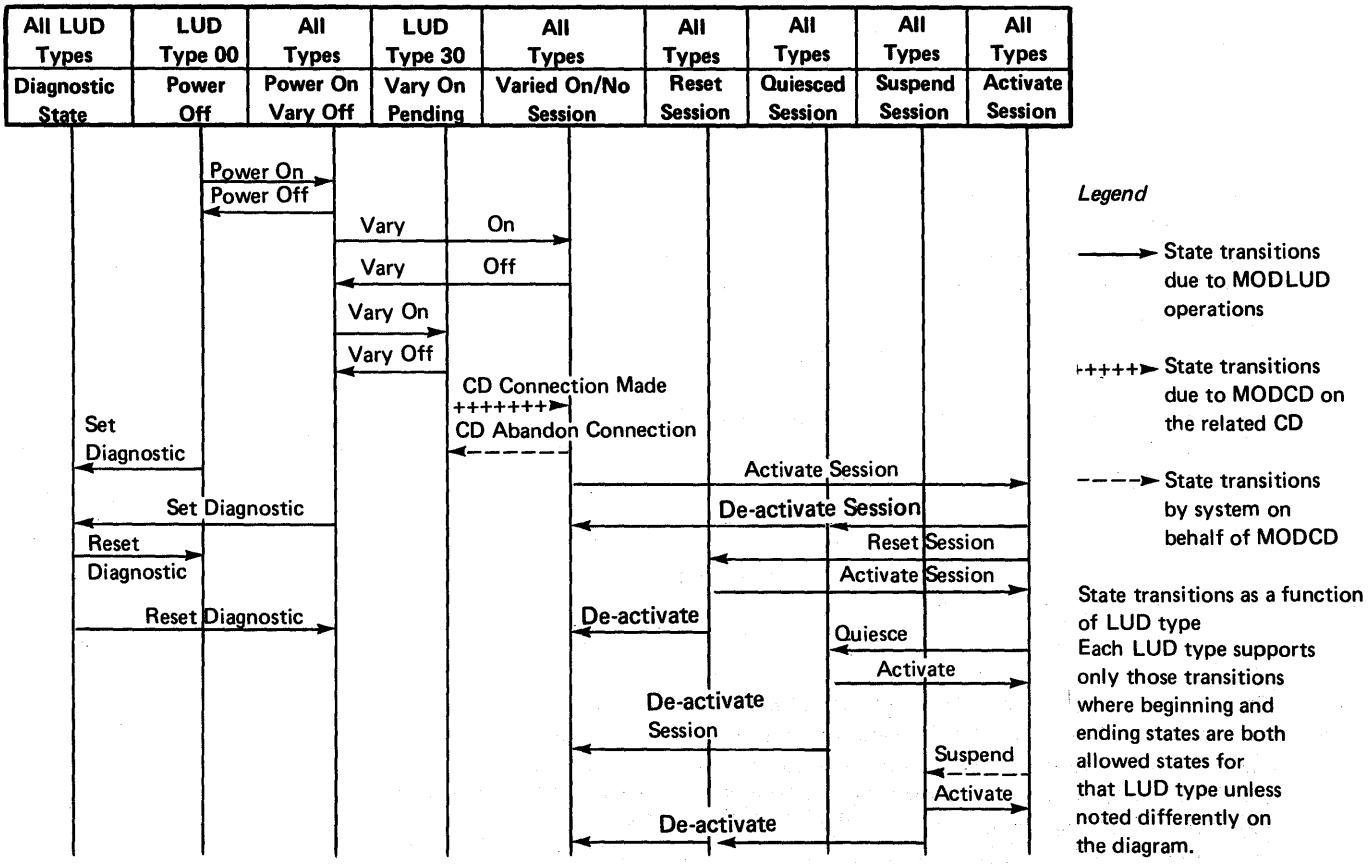
If the attachment method indicates that the LUD is attached to both a CD and an ND (LUD type 30) and the CD represents a station on a switched network, the following conditions apply:

- 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.



Notes:

1. De-activate from active state causes a state change first to quiesced state and then to varied on state. De-activate from suspeneded 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 Figure 17-2 (Part 3 of 4) for load/dump change rules.

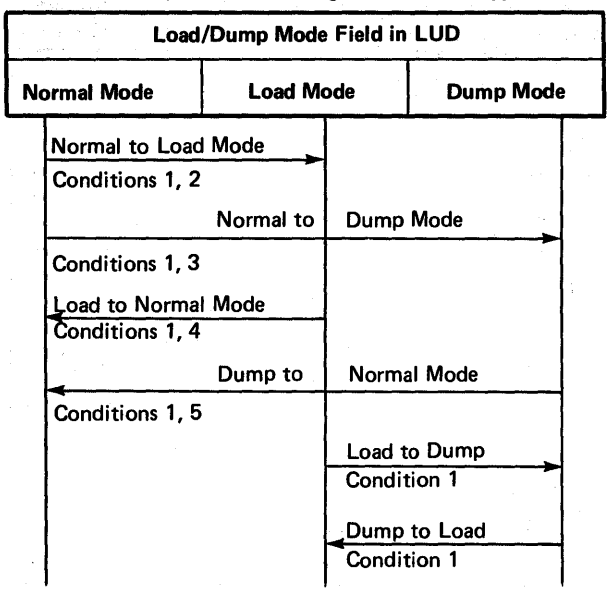
Figure 17-2 (Part 1 of 4). LUD State Change Rules

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]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 17-2 (Part 2 of 4). LUD State Change 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 17-2 (Part 3 of 4). LUD State Change Rules

<table>
<tr><th>Load/Dump Indicator Field (hex)</th><th>Diagnostic State</th><th>Power Off</th><th>Power On</th><th>Vary On Pending</th><th>Varied On</th><th>Reset Session</th><th>Quiesced Session</th><th>Suspended Session</th><th>Active Session</th></tr>
<tr><td>Load/Dump Device
00 = Not a Load/Dump Device
01 = Noninterruptible/ Nonexchangeable
11 = Interruptible
21 = Exchangeable</td><td colspan="9">Not Modifiable Data
(ignored by MODLUD instruction)</td></tr>
<tr><td>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)</td><td>No</td><td colspan="4">Yes</td><td colspan="3">Note 2</td><td>No</td></tr>
<tr><td>Load/Dump Pending
0000 = Normal
0100 = Load Pending
0200 = Dump Pending</td><td colspan="9">Not Modifiable Data
(ignored by MODLUD instruction)</td></tr>
<tr><td>Corresponding Primary Address (note 1)
nnnn = Logical unit address of the primary device when this device is an alternative mode device</td><td>No</td><td colspan="4">Yes</td><td colspan="4">Ignored by MODLUD instruction</td></tr>
<tr><td>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 address of the previous current device
000000 = No modification requested</td><td>No</td><td colspan="7">No
(template must contain hex 000000)</td><td>Yes</td></tr>
</table>

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 17-2 (Part 4 of 4). LUD 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 exeuction 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.

Events

0002 Authorization
  0101 Object authorization violation

000B Logical unit description
  0601 Logical unit description contact successful (for all Modify LUD-Vary On instruction)
  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	
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	
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.

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		Bit(6)	
4-5		**Reserved**		Char(2)	
• Reserved				Char(10)	
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 Figure 17-3 (Part 1 of 2) for a description of the states that exist for the ND object and also for the valid state changes that can be made by the Modify Network Description instruction. Figure 17-3 (Part 2 of 2) shows the valid relationship for modification of the other elements in the ND.

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.

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.

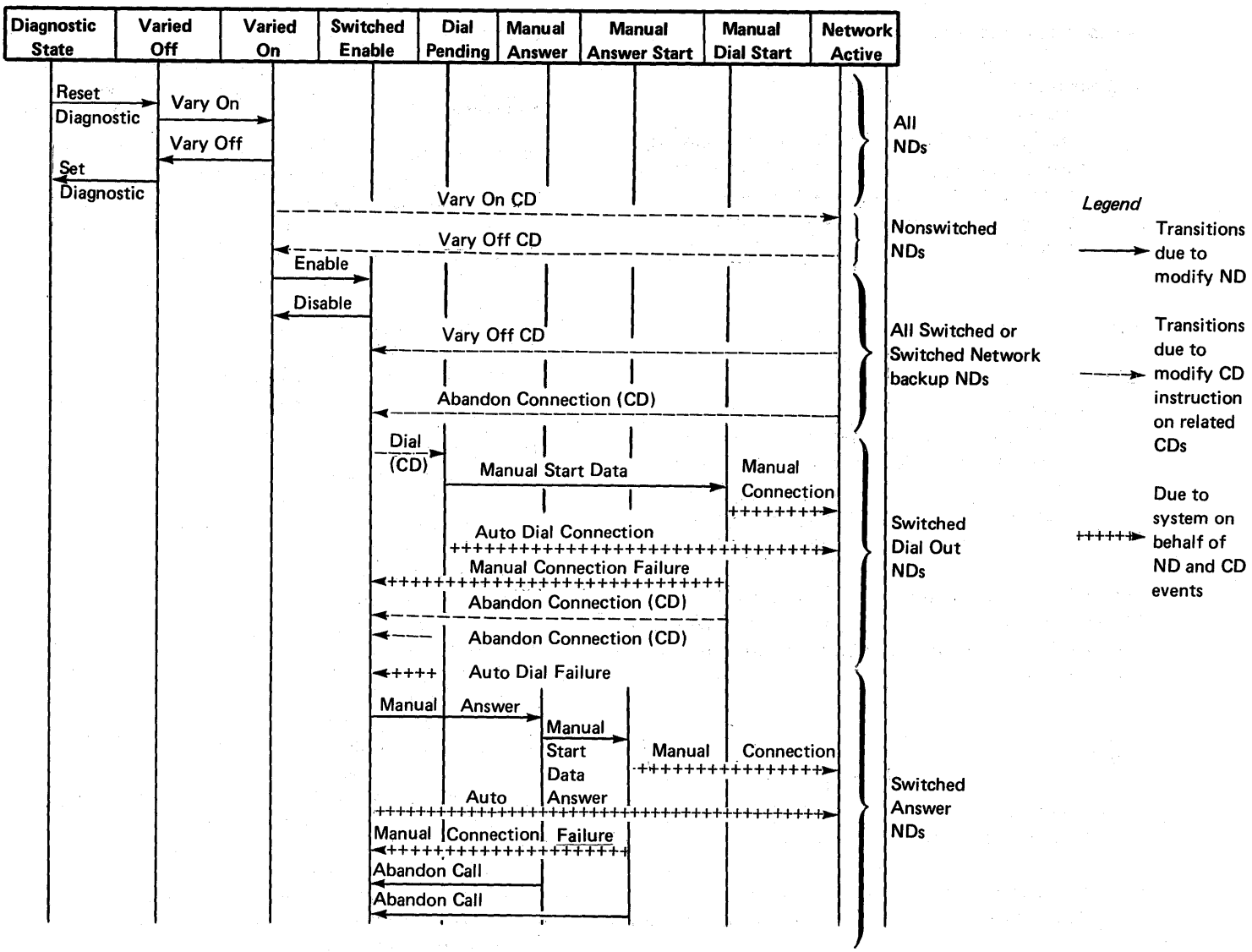


Figure 17-3 (Part 1 of 2). ND State Change Rules

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 17-3 (Part 2 of 2). ND State Change 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.

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

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	
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	
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)
Reserved (binary 0)	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(16)
Request priority	Bin(2)
Request ID	Bin(2)
Function field	Char(1)
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)
Reserved (binary 0)	Char(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)

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.

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 service 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.

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 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.

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.

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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices*.

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 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.

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.

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 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.

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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* for the contents of request descriptors for specific devices.

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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* of this publication. 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:

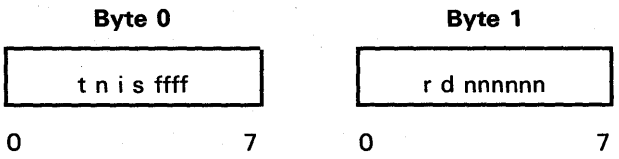
Field	Format
Source/sink request address	Space pointer
Request ID	Bin(2)
Error summary	Bin(2)
RD number	Bin(2)
RIU segment count	Bin(2)
Device-dependent status	Char(40)

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 – 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 *Chapter 25. 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 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 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 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 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 = Insufficient space to activate load/dump
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 D0 = 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 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. Whenever possible, the user should provide time-out mechanisms for Request I/O operations 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 *Chapter 25. 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 asynchronously 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

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	
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	
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	
3C Service		
01 Invalid service session state		X
02 Unable to start service session		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 B. 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.

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	
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	
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

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X		
02 Boundary alignment	X		X	
03 Range	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		
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		
32 Scalar Specification				
01 Scalar type invalid		X		
38 Template Specification				
01 Template value invalid		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 ODT numbers — Bin(2)
- Offset to list of exception descriptions — Bin(4)
- Number of exception description ODT (object definition table) numbers — Bin(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 Bin(2) ODT numbers. The number of parameter ODT numbers and the number of exception description ODT 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 — Bit 0
    - 0 = Not tracing new invocations
    - 1 = Tracing new invocations
  - Return trace — Bit 1
    - 0 = Not tracing returns
    - 1 = Tracing returns
  - Invocation trace propagation — Bit 2
    - 0 = Not propagating invocation trace
    - 1 = Propagating invocation trace
  - Return trace propagation — Bit 3
    - 0 = Not propagating return trace
    - 1 = Propagating return trace
  - Reserved (binary 0) — Bits 4-15
- Instruction number — Bin(2)
- Offset to parameter values — Bin(4)
- Offset to exception description values — Bin(4)
- 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	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
0B Invalid number of operands			X
0C Invalid operand ODT reference	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 POINTER (MATPTR)

Op Code (hex)	Operand 1	Operand 2
0512	Receiver	Pointer

Operand 1: Space pointer.

Operand 2: System pointer, space pointer, 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 = Data pointer
  Hex 02 = Space pointer
  Hex 03 = System 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
08	User profile
0A	Queue
0B	Data space
0C	Data space index
0D	Cursor
0E	Index
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 02 = Zoned decimal
      - Hex 03 = Packed decimal
      - Hex 04 = Character
    - Scalar length Char(2)
      - If binary or character:
        - 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.

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	
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	
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

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	
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	
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
  - 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)
- Time-stamp 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)

The time-stamp field is materialized as an 8-byte unsigned binary number in which bit 41 is equal to 1024 microseconds.

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
08	User profile
0A	Queue
0B	Data space
0C	Data space index
0D	Cursor
0E	Index
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 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X		
02 Boundary alignment	X		X	
03 Range	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	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		
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		
32 Scalar Specification				
01 Scalar type invalid	X	X		
38 Template Specification				
03 Materialization length exception	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

Exception	Operands 1	2	3	Other
06 Addressing				
01 Space addressing violation	X	X		
02 Boundary alignment	X		X	
03 Range	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	X
44 Partial system object damage	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
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		
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
- 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.

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	
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	
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 B. 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	Operands 2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	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	
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.

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 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 0 = Normal 1 = Automatic – 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 – Damaged machine context Bit 7 0 = Not damaged 1 = Machine context damaged – Power control initialization Bit 8 0 = Successful 1 = Failed – Recovery object list status Bit 9 0 = Complete 1 = Incomplete – Recovery phase completion Bit 10 0 = Complete 1 = Incomplete

Selection Value	Attribute Description
Hex 0108 (continued)	
	– 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 – Reserved (binary 0) Bits 14-15
	• Number of damaged main storage units Bin(2)
	• Number of entries in recovery object list Bin(4)
	• Address of recovery object 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 recovery object 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 Recovery object 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 recovery object list entry indicates how many objects are listed in the space located by the address of recovery object list entry.

The address of recovery object 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 or until the next IMPL when a new list is generated. The number of such objects is indicated by the number of entries in the recovery object 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 the objects that were partially updated at machine termination. This list is located by the recovery object list pointer. Only data spaces and data space indexes appear in the list. A recovery entry exists in the list for every object that is not fully updated at IPL as not having been handled. The object type identifies the type of the object and defines the format of the object status field. The object pointer is a system pointer to the object in question. Object type, object status, and the object pointer are repeated for each object in the list. The 15-byte object status field definitions include the following:

- Data space
  - Status — Char(1)
    - Damaged — Bit 0
      - 0 = Not damaged
      - 1 = Damaged
    - Indexes detached from data space — Bit 1
      - 0 = Indexes remain attached
      - 1 = All indexes detached from this data space
    - Reserved (binary 0) — Bits 2-7
  - Reserved (binary 0) — Char(10)
  - Ordinal entry number of last entry — Bin(4)

- Data space index
  - Status — Char(1)
    - Damaged — Bit 0
      - 0 = Not damaged
      - 1 = Damaged
    - Invalidated — Bit 1
      - 0 = Not invalidated
      - 1 = Invalidated
    - Reserved (binary 0) — Bits 2-7
  - Reserved (binary 0) — Char(14)

Data space – If object damage was detected during IPL, the object is marked as damaged, damage is indicated in the object status field, and no 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 no 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 recovery object list. Only damaged or invalidated data space indexes are included in the list.

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	
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	
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 materialized.

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)

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.

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 1	2	Other
06 Addressing			
01 Space addressing violation	X	X	
02 Boundary alignment	X	X	
03 Range	X	X	
0A Authorization			
08 Special authorization required			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	
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.

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	
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	
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
0621	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 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 and turn off the machine power supply.

    All other values are reserved; if any other values are specified, they cause an exception.
  - Reserved (binary 0) — Bits 4-7
  - 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.

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 checkstopped 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.

Authorization Required

- Privileged instruction

Events

0002 Authorization
    0201 Privileged instruction violation

0017 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	
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	
32 Scalar Specification			
01 Scalar type invalid	X		
02 Scalar attributes invalid	X		
03 Scalar value invalid	X		

Chapter 20. Exception Specifications

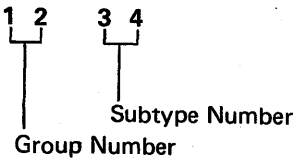
Exception generation is the only facility for synchronously communicating error conditions that are a direct result of System/38 instruction processing. Machine exceptions identify error conditions that require processing before the next sequential System/38 instruction is executed. Instructions that 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
- 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

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
- 08 Length conformance
- 0A Size
- 0B Zero divide

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
- 13 Invalid mapping 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 satisified
- 20 Copy data space entries termination
- 21 Unable to maintain a unique key data space index
- 22 Data space index with selection routine build termination
- 23 Data space index selection routine failure

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

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
- 05 Resume process invalid
- 06 Suspend process invalid
- 08 Mask or unmask process invalid
- 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

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

2E Resource Control Limit

- 01 User profile storage limit 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

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

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

Instruction 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) ≥ 1 and length (L) ≥ 1 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.

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.

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
- 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
- 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
- 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 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
- Divide
- Divide With Remainder
- Remainder
- Subtract Numeric
- Sum
- 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

0C08 Length Conformance

The operand lengths and/or resultant value length 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.

EDIT – The length of the resultant edited value must be equal to the length of the receiver operand.

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.

Instructions Causing Exception:

- Convert Character to Hex
- Convert Hex to Character
- Edit
- Search
- Signal Exception

0C0A Size

An operand was too small to contain a result.

Instructions Causing Exception:

- Add Numeric
- 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

0C0B Zero Divide

An attempt was made to divide by 0.

Instructions Causing Exception:

- Divide
- Divide With Remainder
- Remainder
- 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	Bin(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 interchange buffer.

The error type values are as follows:

- Hex 02 – Decimal Data: (1) Sign encoding is invalid for packed or zoned format, or (2) digit encoding is invalid for packed or zoned format.
- Hex 0A – Size: The destination field is too small to hold all significant digits of the source field.

These errors are the data base equivalents of exceptions numbered hex 0C02 and hex 0C0A 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)
Ordinal entry number (0 if mapping the input key on Set Cursor 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	Bin(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 interchange buffer.

The error type values are as follows:

- Hex 02 – Decimal Data: (1) Sign encoding is invalid for packed or zoned format, or (2) digit encoding is invalid for packed or zoned format.
- Hex 0A – Size: The destination field is too small to hold all significant digits of the source field.

These errors are the data base equivalents of exceptions numbered hex 0C02 and hex 0C0A 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

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 option)
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Update Data Space Entry
- 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).

Information Passed:

- Cursor — System pointer
- Data space — Bin(2)
- Ordinal entry number — Bin(4)
- Return code (bit significant) — Char(1)
  - Hex 00 = Locked to another process
  - Hex 01 = Locked to current process

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

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
- Data Base Maintenance (invalidate option)
- 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 — Bin(2)

Instructions Causing Exception:

- Copy Data Space Entries
- 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 — 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:

- 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 has been reached as the result of the Set Cursor instruction.

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

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. The template field number indicates the field in the template that has the error. A template field number of 0 indicates the number of byte fields is in error. A field of 1 indicates the input mapping type. The field for specification is considered to be one field for counting purposes. The possible errors are an invalid value, a value that exceeds the allowed range, a length that is invalid for a specified type, or a type that is inconsistent with the type specified for the field in the data space.

Information Passed:

- Data space number (position in list) — Bin(2)
- Template field number in error — Bin(2)

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 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.

Infomation Passed:

- Cursor — System pointer

Instructions Causing Exception:

- Retrieve Data Space Entry
- 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 — Char(1)
  - Hex 00 = Input mapping template
  - Hex 01 = Output mapping template

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

1220 Copy Data Space Entries Termination

The maximum number of exceptions has been reached for the Copy Data Space Entries instruction.

Information Passed:

- Cursor (receiver) — System pointer
- Cursor (source) — System pointer

Instructions Causing Exception:

- Copy Data Space Entries
- 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 — System pointer
- Data space — System pointer
- Data space index (invalidated) — System pointer

Instructions Causing Exception:

- Copy Data Space Entries
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Update Data Space Entry
- Signal Exception

1222 Data Space Index With Selection Routine Build Termination

While creating or rebuilding a data space index that contains a 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 — Char(1)
    - Hex 01 = Selection mapping error
    - Hex 02 = Selection routine failure
    - Hex 03 = Error in invoking selection routine
  - 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)

1223 Data Space Index 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 — 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

Instructions Causing Exception:

- Copy Data Space Entries
- Insert Data Space Entry
- Insert Sequential Data Space Entries
- Update Data Space Entry
- 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:

- A copy of the event monitor template being tested

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 resignal 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
- 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 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:

- Space pointer to the lock request template
- Failing request number (relative entry position) — Bin(2)

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
- 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 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.
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.
0005	The constants that are built into the encapsulated object from the program template exceed 64 K bytes minus 129 bytes. Constants defined in the program template and initial values for automatic storage are included in this area.
0006	The work space needed by the machine to support the machine interface invocation for this program exceeds 65 535 bytes (see note).

Error Code	*Meaning*
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 680. 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	64 K	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 currently held locks was exceeded.

No more than 57 344 locks, implicit locks, data base entry locks, and internal locks required for machine operation may exist at any one time.

Instructions Causing Exception:

- Lock Object
- Activate Cursor
- Create Cursor
- Create Data Space Index
- Data Base Maintenance
- Set Cursor
- Initiate Process
- Modify Process Attributes
- Dequeue
- Any instruction that acquires an implicit lock or an internal machine lock
- 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 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:

- Time stamp 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:

- Time stamp 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 for a load/dump operation.
- An attempt was made to replace a progarm 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 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.

Information Passed:

- System pointer to the object

Instructions Causing Exception:

- Activate Cursor
- Ensure Data Space Entries
- De-activate Cursor
- Create Duplicate Object
- Delete Object From Access Group
- Data Base Maintenance (all options)
- Materialize Cursor Attribute
- Suspend
- Request I/O
- Copy Data Space Entries
- 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.

Information Passed:

- System pointer to the object

Instructions Causing Exception:

- Activate Cursor
- Create Data Space Index
- Data Base Maintenance
- De-activate Cursor
- Delete Data Space Entry
- Ensure Data Space Entries
- Initiate Process
- Insert Data Space Entry
- Release Data Space Entries
- Retrieve Data Space Entry
- 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.

Information Passed:

- System pointer to the object.

Instructions Causing Exception:

- Destroy Access Group
- Destroy Data Space
- Destroy Data Space Index
- Destroy User Profile
- Signal Exception

24 Pointer Specification

2401 Pointer Does Not Exist

A pointer reference has been made to a storage location in a space that does not contain a pointer.

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 Process Attributes
- Modify Process Attributes
- Resume Process
- Suspend Process
- Terminate Process
- 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 Char(8) (bit significant)
- Modify attribute Char(1) (bit significant)

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 Bin(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 Bin(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.

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, 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.

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.

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.

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

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
- Transfer Control

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

2E Resource Control Limit

2E01 User Profile Storage Limit Exceeded

The user profile specified insufficient auxiliary storage to create or extend a permanent object.

Instructions Causing Exception:

- All create instructions creating a permanent object
- All instructions extending a permanent object
- 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

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.
- 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.
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.
1206	CRTCD	Invalid ND candidate (not switched, wrong ND type or wrong line discipline).
1301	CRTLUD	Forward object supplied is of the wrong source/sink object subtype.
2101	MODND	Status change attempted with no CDs or LUDs attached to this ND (nonswitched).
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).
2301	MODLUD	Status change attempted with no valid forward pointer.
3401	Request I/O	Request I/O response queue does not have proper attributes for the Request I/O instruction.

Instructions Causing Exception:

- Create Controller Description
- Create Logical Unit Description
- Create Network Description
- Modify Controller Description
- Modify Logical Unit Description
- 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 5251 remote 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 Char(1)
  (bit significant)
  - 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(6)
  - Reserved — Char(8)
- 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	

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
- 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 agreee with the hardware or support attributes currently configured on this system.)

Code (hex)	Instruction	Meaning
2102	Modify ND	Vary on failure
2103	Modify ND	Manual answer failure
2105	Modify ND	Enable 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
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
2311	Modify LUD	Resume failure
2312	Modify LUD	Suspend failure
2313	Modify LUD	Quiesce failure

The device-specific error codes are as follows:

Code (hex)	Meaning
0101	No MCR entry
0102	MCR entry not supported
0103	No MCR related OU
0104	Wrong related OU
0105	MCR device type or model number is different
0106	MCR power control mismatch

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 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
- Signal Exception

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

Chapter 21. 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(24)
- 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 ignored if the event signal is not related to a process action, such as a timer event.
- 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(*)

The following chart shows the status code definitions for the vary on failures. The chart also indicates if additional ID information is available (6-byte SSP-ID field).

Reason	Status	XID Data
Line failure	0001	No
Dial operation unsuccessful	0002	No
XID data or SSCP ID data does not match CD	0003	Yes
ND not in candidate list of this CD	0004	Yes
CD not varied on (CD represents a primary nonswitched station)	0005	Yes

0403 Loss of Contact

Compare Value: Allowed

Event-Related Data:
- System pointer to the controller description
- Data length (hex 000E or decimal 14

Status code	Char(2)
Hex 0001= disconnect received from SDLC primary	
Reserved	Char(12)

0501 Controller Description Failure (station inoperative)

0502 Controller Description Failure (protocol violation detected)

0503 Controller Description Failure (SSCP to PU session inactive)

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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* and *Chapter 24. Communications and Locally Attached Work Stations*)	Char(2)
Time stamp (if matching error log entry)	Char(8)
OU number	Char(2)
Optional data (see *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* and *Chapter 24. Communications and Locally Attached Work Stations*)	Char(2)
Optional system pointer	Char(16)

0601 Controller Description Manual Intervention

Compare Value: Allowed
- System pointer to the controller description

Event-Related Data:
- 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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* and *Chapter 24. Communications and Locally Attached Work Stations*)	Char(2)
Time stamp	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

0401 Data Space Entry Not Addressed By Data Space Index

Compare Value: Allowed
- System pointer to the data space index

Event-Related Data:
- System pointer to data space
- System pointer to data space index
- Data space index status code Char(1)
  - Hex 00 = Data space index is being created
  - Hex 01 = Data space index is invalid but is being rebuilt
  - Hex 02 = Data space index is invalid
- Reason code Char(1)
  - Hex 01 = Selection mapping error
  - Hex 02 = Selection routine failure
  - Hex 03 = Problem in invoking selection routine
- Data space number Bin(2)
- Ordinal entry number Bin(4)

The event is not signaled for entries omitted by the selection routine but rather when the selection routine encounters an error.

The data space index system pointer field contains binary 0's if the data space index has not completed creation.

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

0505 Logical Unit Description Unsolicited Incoming Messages Expedited (secondary)

0506 Logical Unit Description Unsolicited Incoming Messages Nonexpedited (secondary)

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 (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

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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* and *Chapter 24. Communications and Locally Attached Work Stations*)	Char(2)
Time stamp (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)

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 *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* and *Chapter 24. Communications and Locally Attached Work Stations*)	Char(2)
Time stamp of matching error log entry	Char(8)
OU number	Char(2)
Optional data (see *Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices* and *Chapter 24. 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)

Compare Value: 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 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 (source/sink request)
- Data length (hex 0000) — Bin(2)
- Variable data — None

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)

000D Machine Status

0101 Machine Check

Compare Value: None allowed

Event-Related Data:

The machine check event-related data is divided into three parts. The three parts are machine-related data, process-related data, and VLOG dump ID.

– 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.
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 time stamp	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)
- System pointer to the program
- Instruction number — Bin(2)
- VLOG dump ID — Char(8)
- Time stamp (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
- Auxillary 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)

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 80% 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

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(14)
- Error code (see *Chapter 24. Communications*) — Char(2)
- Time stamp of matching error log entry — Char(8)
- OU number — Char(2)
- Optional data (see *Chapter 24. Communications*) — Char(2)

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.

– 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

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 1024 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 from PASA (process automatic storage area)
- Invocation attribute from PASA 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 from the old PASA entry
- System pointer to the associated program pointer from the new PASA entry
- Invocation attribute from the old PASA entry Char(16)
- Invocation attribute from the new PASA entry 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 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

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 0006 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.

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 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)

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

Chapter 22. 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. 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.

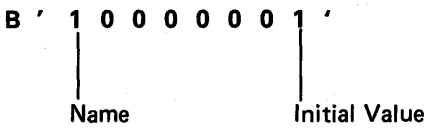
- OES appendages
  - A series of variable-length fields each containing a specific collection of information about the object.

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:



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.

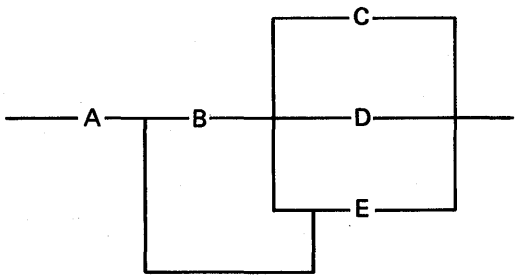
ODT ENTRIES IN DETAIL

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 presently used 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.

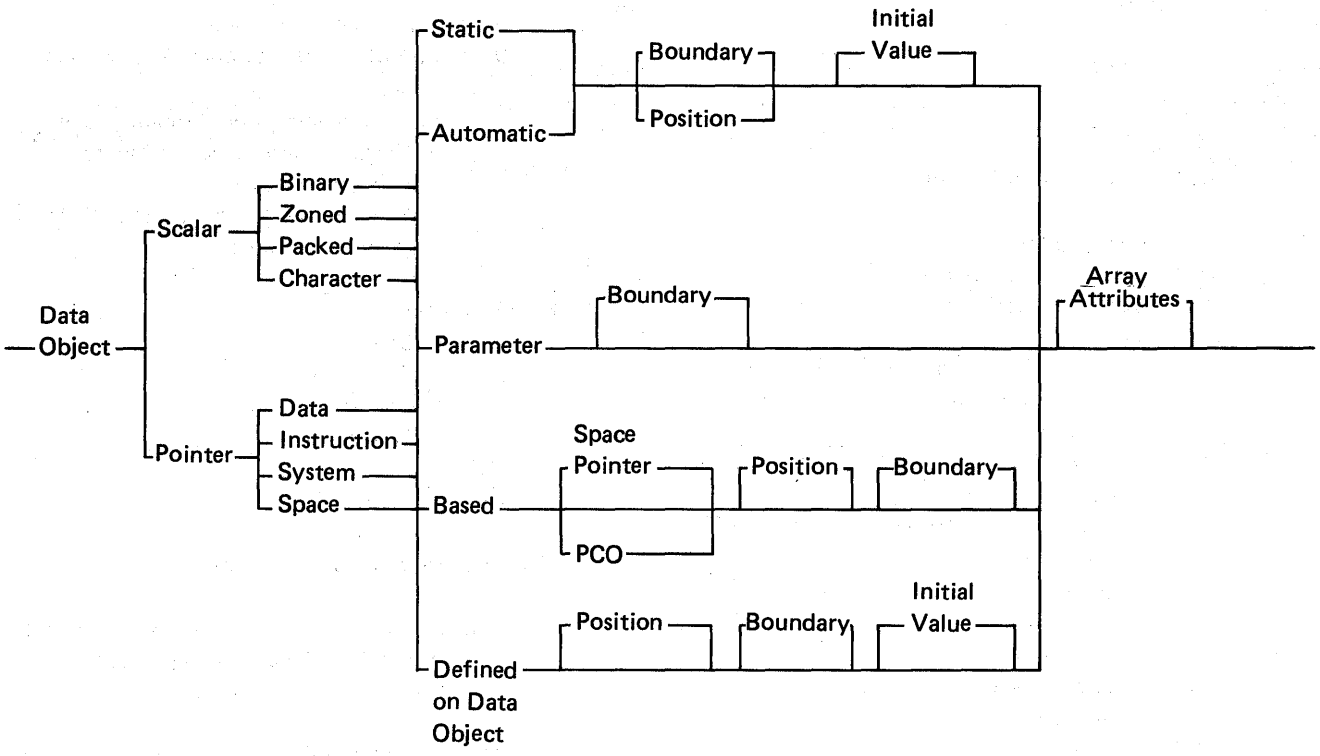
Data Object

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.

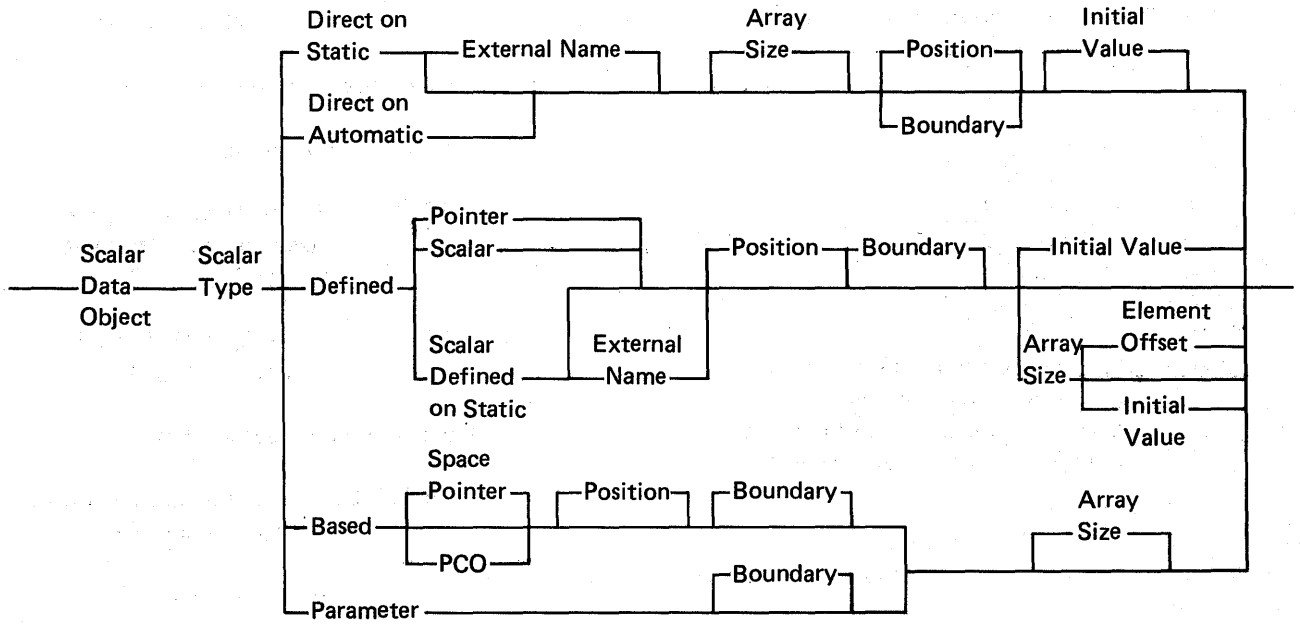
Pointer data objects provide operational characteristics for pointer data contained in a space.

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 Abnormal value attribute
0 = Do not refetch base addressability when base pointer is modified
1 = Refetch base addressability when base pointer is modified

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 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, or zone
- Blank character value (hex 40) for character strings

13-15 Scalar type
000 = Binary
001 = Reserved
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 zoned or packed decimal, then:

Bits	Meaning
16-23	Digits (D) to the right of assumed decimal point, where $0 \leq D \leq T$
24-31	Total digits (T) in field, where $1 \leq T \leq 31$

If character string scalar, then:

Bits 16-31: String length (L), where $1 \leq L \leq 32\,767$

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 $1 \leq L \leq 32$
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 $0 \leq D \leq T$
1	Total digits (T) in field, where $1 \leq T \leq 31$

If character string scalar, then:

Bytes 0-1: String length (L), where $1 \leq L \leq 32\ 767$

Array Appendage

Bytes	Meaning
0-3	Number (N) of elements in the array, where $1 \leq N \leq 16\ 777\ 215$
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 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:

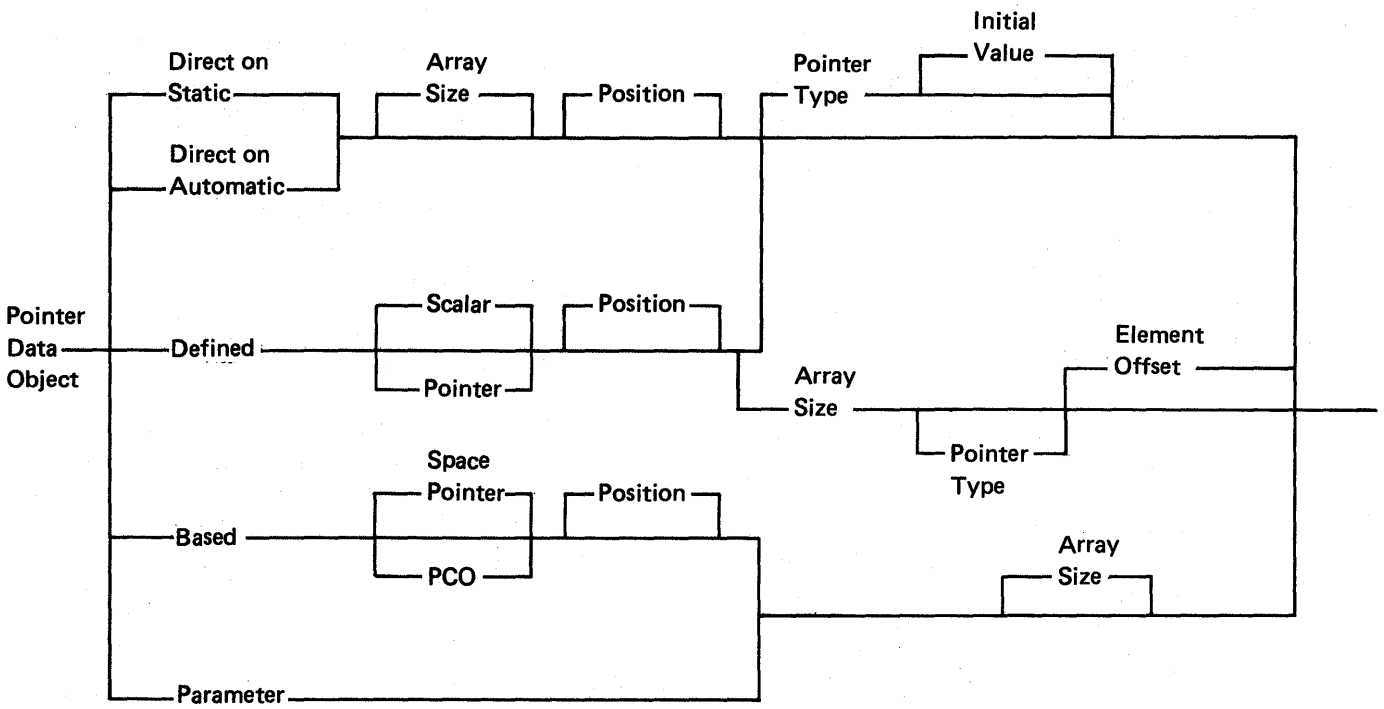
4 bytes: Length of initial value (less than or equal to the total number of bytes in the array)

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.

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
0-1	Reserved
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 $1 \leq N \leq 1\,000\,000$
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 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:

Bytes 0-1: ODT reference – Branch point or instruction number (Bit 0 is 1 to indicate immediate data; value is in bits 1-15.)

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 names (value of 1) Bin(2)
  - Scalar data object name length (N) Bin(2)
  - Scalar data object name string, where $1 \leq N \leq 32$ Char(N)

- 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 names (value of 2) Bin(2)
  - Program type (hex 02) Char(1)
  - Program subtype Char(1)
  - Program name length (M) Bin(2)
  - Program name string, where $1 \leq M \leq 30$ Char(M)
  - Scalar data object name length (P) Bin(2)
  - Scalar data object name string, where $1 \leq P \leq 32$ Char(P)

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 $1 \leq N \leq 30$ 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 $1 \leq M \leq 30$ 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 $1 \leq P \leq 30$ 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
08	User profile
0A	Queue
0B	Data space
0C	Data space index
0D	Cursor
0E	Index
10	Logical unit description
11	Network description
12	Controller description
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.

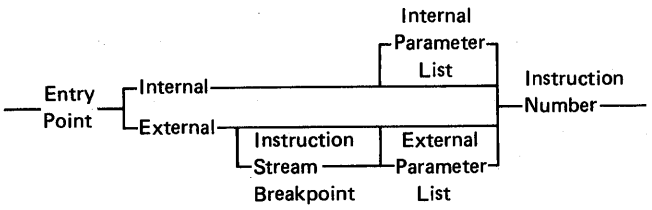
The size for the static and automatic spaces can also be defaulted if the size of static (automatic) storage entry in the program template is 0. The value used is such that the space is large enough to contain all data objects with defaulted and explicit direct mapping. An exception is signaled if a 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
0-2	Reserved (binary 0)
3	Parameter information present 0 = Parameter information not present 1 = Parameter information is present (required if there is an OES)
4	Instruction stream breakpoint 0 = Appendage not present 1 = Instruction stream breakpoint appendage present
5-6	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.

Branch Point

Attribute Combinations

— Branch Point ——— Instruction Number

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 ≤ N ≤ 32 767.

OES Format

No OES is needed for branch points.

Instruction Definition List

Attribute Combinations

— Instruction Definition List ——— Number of Elements (N) ——— N Instruction Reference

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
0-6	Reserved (binary 0)
7	Initial value present 1 = Initial value is required

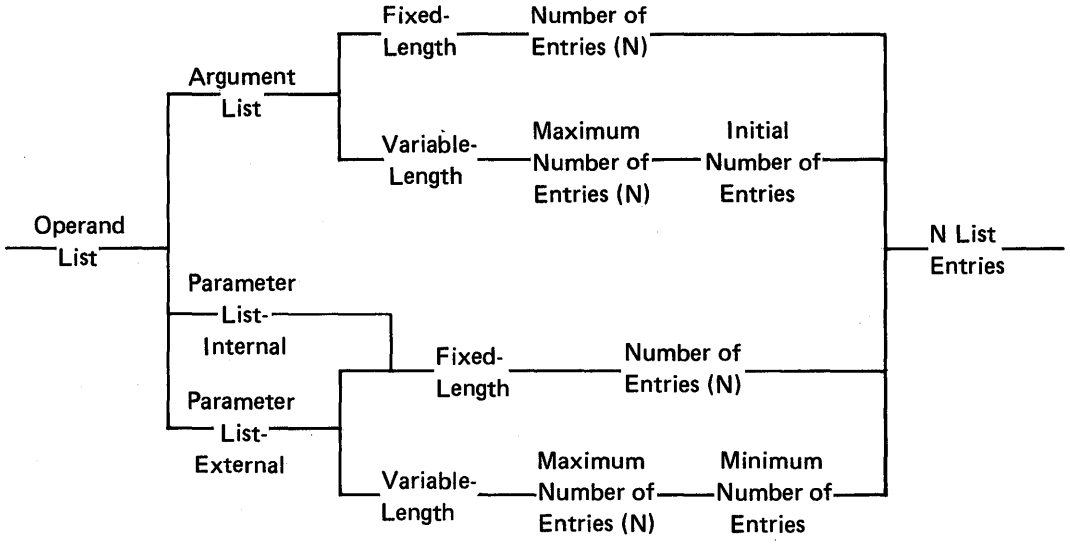
Initial Value Appendage

Bytes	Meaning
0-1	Number (N) of list elements, where 1 ≤ N ≤ 255
2-3	Instruction reference element 1
.	
.	
.	
X-X+1	Instruction reference element N

Note: An instruction reference element may be either an immediate instruction number (bit 0 is 1, bits 1-15 contain a binary value representing an instruction number) or an ODT reference to a branch point.

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 $1 \leq N \leq 255$

- For fixed-length lists (argument or parameter)

Bytes	Meaning
0-1	Number (N) of elements, where $1 \leq N \leq 255$
2-3	Reserved (binary 0)

- For variable-length lists

Bytes	Meaning
0-1	Maximum number (N) of elements that the list can contain, where $1 \leq N \leq 255$
2-3	For argument lists, the initial number (M) of elements to be passed on a Call External or Transfer Control instruction, where $0 \leq M \leq N$ For parameter lists, the minimum number (M) of elements to be received on entry, where $0 \leq M \leq N$

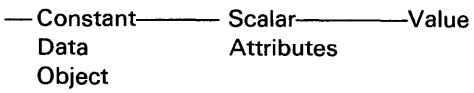
4-5	ODT reference 1
	.
	.
	.
X-X+1	ODT reference N N elements are required.

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



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, or zoned – 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)

ODV Format (Continued)

Bits	Meaning
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 = Reserved 010 = Zoned decimal 011 = Packed decimal 100 = Character 101-111 = Reserved

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 zoned or packed decimal, then:

Bits	Meaning
16-23	Digits (D) to the right of assumed decimal point, where $0 \leq D \leq T$
24-31	Total digits (T) in field, where $1 \leq T \leq 31$

If character string scalar, then:

Bits 16-31: String length (L), where $1 \leq L \leq 32\,767$

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 zoned or packed decimal, then:

Bytes	Meaning
0	Digits (D) to the right of assumed decimal point, where $0 \leq D \leq T$
1	Total digits (T) in field, where $1 \leq T \leq 31$

- If character string scalar, then:

  Bytes 0-1: String length
  0 = Length (L) beyond 2047, where $1 \leq L \leq 2047$

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

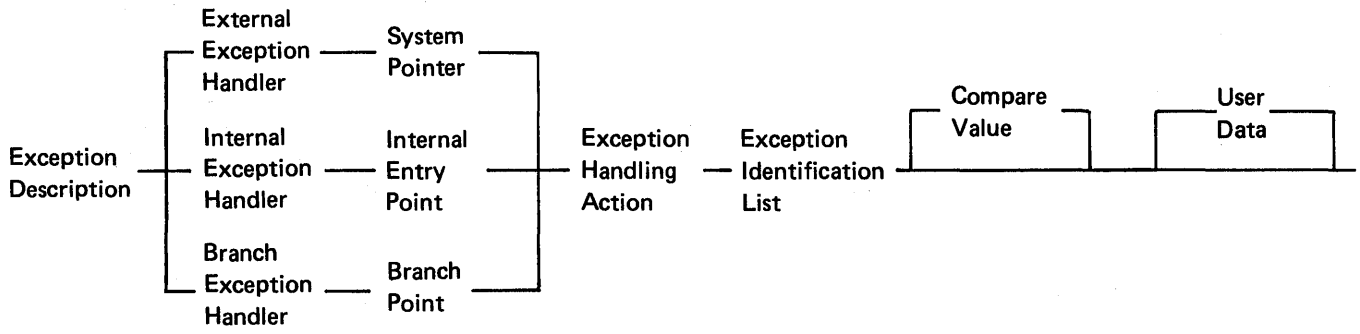
  L bytes: Associated value

  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. 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)
1-4	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

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.

User Data Appendage

Bytes 0-1	ODT reference to – Pointer data object – Scalar data object This data object must be either in the automatic or 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 target appendage for a branch point may be an immediate instruction number (if bit 0=1, bits 1-15 contain a binary value representing an instruction number) or an ODT reference to a branch point.
3. 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

Chapter 23. Source/Sink Specialization and Programming Considerations for Local Devices

Source/sink specialization involves the creation, modification, or destruction of the various source/sink objects. These objects are called LUDs (logical unit descriptions), CDs (controller descriptions), and NDs (network descriptions). LUDs, CDs, and NDs are necessary to support the physical input/output devices on each machine. Because the devices that are attached to each machine are variable and because each of these devices can perform many functions, the source/sink objects to be created must be tailored to match these various characteristics. Some source/sink objects have been created and shipped with the machine because the device characteristics are known at that time. Examples of these devices are the machine console, the machine printer, and the load/dump device. Other source/sink objects need to be created when the machine is installed because devices such as communications terminals are not necessarily known at the time of manufacture. Other source/sink objects need to be created subsequent to machine installation when new devices are added as the machine usage increases. Creation of the source/sink objects is a prerequisite to any use of the corresponding devices attached to the machine.

Before source/sink objects are created, certain actions must have taken place to ensure the uniqueness and integrity of the source/sink objects. The machine maintenance function is used to install the hardware (I/O device, control unit, or communications adapters) and to update the internal maintenance configuration records to reflect this hardware. Creation of the source/sink objects is not allowed to proceed if confirmation of this maintenance activity has not been completed. The extent of this confirmation is as follows:

- Maintenance configuration records must reflect the installation of all hardware before creation of any corresponding ND type 00, CD type 00, and LUD type 00 objects.
- No checking of physical hardware installation is done for any CD type 10, LUD type 10, or LUD type 30 objects.

This chapter describes the special programming considerations and the instructions needed to operate and control the various local devices.

The instruction formats and usage contained in this chapter are specific for the individual devices. A complete description of all fields in the instructions is given in *Chapter 17. Source/Sink Management Instructions*.

MACHINE CONSOLE PROGRAMMING CONSIDERATIONS

The machine console is a unique I/O device in that every machine has only one machine console device. It is the primary interface by which the operator monitors and operates the machine.

The basic object of control is the LUD (logical unit description). All references to a device are made with respect to the LUD. The Create LUD, Modify LUD, and Destroy LUD instructions control the environment in which the machine console operates. The Request I/O instruction controls the machine console and causes data to be passed to and from the machine console.

Before a Request I/O instruction can be accepted, several steps must occur.

1. An LUD must be created through the use of the Create LUD instruction.
2. The machine console must be varied on through the use of the Modify LUD instruction.
3. The LUD must be made active through the use of the Modify LUD instruction.

MACHINE CONSOLE CREATE LOGICAL UNIT DESCRIPTION (CRTLUD) TEMPLATE

The fields of the LUD template (see *Create Logical Unit Description (CRTLUD)* in Chapter 17) specifically needed for the machine console are as follows:

Field Name	Entry
Logical unit description type	Char 00
Device type	Char CONS
Model number	Hex 40404040
LUD operational unit number	Hex 0002
Power control	Hex 0000
Session definition data	Bin 0
Load/dump indicator	Bin 0
Specific characteristics length	Hex 0000
Retry value length	Hex 0000
Error threshold length	Hex 0000
Device-specific contents length	Hex 0000
Device-specific modify length	Hex 0000

MACHINE CONSOLE MODIFY LOGICAL UNIT DESCRIPTION (MODLUD)

The following is a list of functions that the machine console supports through the Modify Logical Unit Description instruction:

- Vary On
- Vary Off
- Activate
- De-activate
- Suspend
- Quiesce
- Reset
- Resume (activate after suspend, quiesce, or reset)

See the *Functional Concepts Manual* for the meaning and use of each function.

MACHINE CONSOLE REQUEST I/O INSTRUCTION (REQIO)

The Request I/O instruction is used to send I/O requests to the machine console. A source/sink request (SSR) is always associated with a Request I/O instruction. Source/sink data (SSD) is not always required with a Request I/O instruction. Therefore, the SSD is discussed as it applies to each type of console request. An FBR (feedback record) containing information about the execution of the Request I/O instruction is posted for each Request I/O instruction successfully issued.

Source/Sink Request (SSR)

The SSR for a Request I/O instruction to the machine console contains the following values:

Fields	Values
Source/sink object	Pointer to the LUD
Response queue	Pointer to the response queue
Source/sink data area (SSD)	Space pointer
Optional pointer	Reserved (binary 0)
Request priority	See *Request I/O (REQIO)* in Chapter 17.
Request identification field	See *Request I/O (REQIO)* in Chapter 17.
Function field	Hex 80 (any '8n' value is accepted)
Control field	Char N (Request I/O instruction) or Char C (Request I/O (continue instruction))
Key length	Bin(2)
Key offset	Bin(2)
RD (request descriptor) count	Hex 0000 = Request I/O (continue) instruction Hex 0001 = Request I/O instruction
RD (request descriptor) offset	Bin(2)

The RD count field specifies the number of 16-byte request descriptor fields associated with the Request I/O instruction. The machine console supports only one RD per Request I/O instruction.

The RD count field is directly related to the control field. If the control field data is N, then the RD count field data should be hex 0001. If the control field data is C, then the RD count field data should be hex 0000.

The RD offset field indicates the offset from the start of the SSR to the 16-byte RD. The source/sink data area (SSD) pointer points to the data being transferred to/from the machine console corresponding to the command in the RD.

The format of the 16-byte RD is as follows:

Byte 0 — Command — Char(1)
- Hex 01 – Write
- Hex 04 – Control
- Hex 05 – Write with Control
- Hex 12 – Read Modified
- Hex 22 – Read Buffer
- Hex 44 – Cancel

Bytes 1-6 — Command modifier — Char(6)

- Bytes 1-2 — Control value — Char(2)
  - Bits 0-4 — Reserved (binary 0)
  - Bit 5 — Erase screen
  - Bit 6 — Erase unprotected fields
  - Bit 7 — Reset modified data tags
  - Bits 8-9 — Reserved (binary 0)
  - Bits 10-11 — Keyboard
    - 00 – Unchanged
    - 10 – Lock
    - 01 – Unlock
    - 11 – Unlock
  - Bits 12-13 — Alarm
    - 00 – Unchanged
    - 10 – On
    - 01 – Off
    - 11 – Off
  - Bits 14-15 — Attention light
    - 00 – Unchanged
    - 10 – On
    - 01 – Off
    - 11 – Off

- Byte 3 — Precommand control — Char(1)
  - Bits 0-1 — Reserved (binary 0)
  - Bits 2-3 — Keyboard (precommand)
    - 00 – Unchanged
    - 10 – Lock
    - 01 – Unlock
    - 11 – Unlock
  - Bits 4-7 — Reserved (binary 0)

- Bytes 4-6 — Reserved (binary 0) — Char(3)

Bytes 7-9 — Reserved (binary 0) — Char(3)

Bytes 10-11 — Data length — Bin(2)

Bytes 12-15 — Reserved (binary 0) — Char(4)

The command byte is used to specify the exact function to be performed during the execution of the Request I/O instruction. Valid commands are Read, Write, Cancel, Control, and Write with Control. The last 3 bits of the command byte are used to specify the command as shown in Figure 23-1.

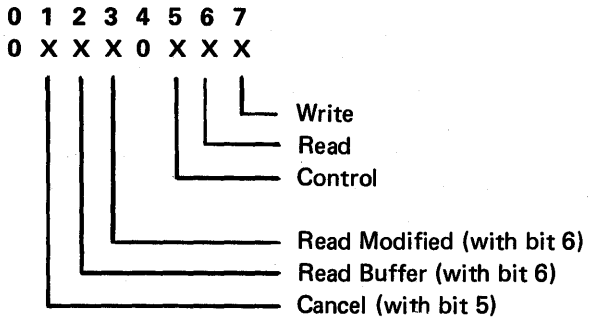


Figure 23-1. Command Byte 0

Because the machine console can perform two different read functions, bit 2 or 3 of the command byte is used along with bit 6 to identify which function is being requested. A special control function, to cancel an outstanding Read Modified command that has not completed, is specified with the combination of bits 1 and 5.

The command modifier bytes are used to control various characteristics of the machine console. The control value field is used only when a Control command is requested (such as the Control or Write with Control command) and must otherwise be binary 0.

When combined with a Write command, the control functions specified in the first byte of the control value field are performed before the write operation, and the control functions specified in the second byte are performed after the write operation.

The erase screen function signifies a request to erase the screen in preparation for writing a new format and data. Because this function causes the entire display buffer to be filled with blanks, all field attributes and data are lost. An error occurs if the display buffer is left containing no field attributes. Therefore, this erase function should be requested only on a Write with Control command that transmits the appropriate characters to reformat the screen.

The erase unprotected fields function causes all input fields (fields with the attribute byte specified as unprotected) to be filled with blank characters (hex 40). That is, any nonblank characters contained in these fields are effectively erased (replaced with blanks).

The reset modified data tags (MDT) function causes the modified data tag in all field attribute characters to be reset. This function is used to condition the input fields so that completion of a Read Modified command results in the transfer of those fields that were modified by the operator in the interim.

The bits of the second byte in the control value field are used in pairs to provide control over the hardware features of the machine console. The keyboard field is used to specify that the keyboard should be locked or unlocked. A locked keyboard means that all keys except the system request key are inoperative. The operator is unable to key data into the display buffer. This is visibly apparent since the cursor disappears when the keyboard is locked. The audible alarm can be turned on or off by setting the alarm field to the appropriate values. The attention indicator light on the console is controlled in a similar manner using the attention light field.

The precommand control field is valid only when a Read Modified command is specified. The field allows keyboard control before the read modified operation.

On a Write command, the data length field specifies the number of bytes of data to be transferred from the SSD to the machine console. On a Read command, the data length indicates the maximum number of bytes that can be placed in the SSD; that is, the data length field indicates the size of the user's input buffer.

Write Command

The Write command causes data to be transferred from the SSD to the machine console display buffer. The data stream can contain special buffer control information along with the field attribute characters and displayable data that is written into the display buffer. By properly constructing the data stream, a single Write command can be used to define formatting fields, display messages in scattered locations on the screen, and position the cursor. Data stream formats are discussed under *Source/Sink Data (SSD) Area* later in this chapter.

Read Buffer Command

The Read Buffer command causes the entire contents of the display buffer to be transferred to the SSD.

This command is normally used to save the entire buffer contents so that the same display format and displayable data can be restored later (in an error situation, for example).

Read Modified Command

Operation of the Read Modified command is somewhat different than the other commands discussed. A Request I/O instruction specifying a Read Modified command signifies that the program is ready to accept data from the machine console and a data area has been provided into which the data can be transferred. The actual transfer of data and completion of the request does not occur until the operator presses one of the function keys (the Enter, Roll ↑ (Roll Up), Roll ↓ (Roll Down), Help, Sys Req, or one of the CF (command function) keys). This command does not cause immediate data transfer and it does not necessarily complete in sequential order if followed by other Request I/O instructions.

The data stream that is transferred when the Read Modified command is executed includes the address of the cursor, identification of which function key was pressed, and the address and contents of all fields that have the modified data tag (MDT) on in the field attribute character. The first field following row 0 and column 0 that has an attribute byte with the MDT bit on is the first field in the data stream.

The machine console IOM always issues a Read Modified command within 2 seconds after a Request I/O instruction. The user can then interrupt the system by pressing the System Request key on the console which signals a supervisory service request event. The user can then obtain a system request menu by issuing a Request I/O instruction.

Depending on the values in the precommand control field, the keyboard is locked or unlocked upon receipt of a Read Modified command. Locking the keyboard before the Read Modified command causes the operator response to be limited to pressing the Sys Req key. Completion of a Read Modified command causes the keyboard to remain locked.

If the keyboard is unlocked and there is no outstanding Read Modified command, the hardware will accept any operator keying actions up to the pressing of one of the function keys. At that point, the keyboard locks. If the next command received by the machine console is a Read Modified, the command completes immediately with the buffered data. If any other command is received, the data remains in the display buffer with the MDT bits on and the new command is processed. A subsequent Control command may turn off the MDTs and a keyboard unlock causes the function key identity to be lost.

The only valid commands when a Read Modify command is outstanding is a Write or Cancel command. Any other commands result in a sequence error.

Cancel Command

The Cancel command is used to request the return of an incomplete Read Modified command and is only valid if a Read Modify command is outstanding. Depending on the relative timing of the Cancel command and the operator pressing the Enter key, the Read Modified command may complete with data being transferred after the Cancel command has been issued. The FBR (feedback record) of both the Read Modified and the Cancel commands indicates whether the read completed normally or was canceled. Completion of a Cancel command causes the keyboard to be locked. The command modifier and the data length fields are not used with a Cancel command and must be binary 0.

Write with Read Modified Command Outstanding

A Read Modified command does not need to be canceled to update displayed messages using a Write command. The user cancels a command only when the user wants to terminate or interrupt the interactive session with the console and must, therefore, retrieve the incomplete Read Modified command.

If a Write command is received while a Read Modified command is outstanding, the machine cancels the Read Modified command, issues the Write command to the machine console, and reinstates the Read Modified command. Care should be taken to ensure that the Write command data stream does not overlay the data entry area. The Write command must also put the keyboard lock back into the state needed for the Read Modified command.

Control Command

The Control command can be coded independently of a Write command to enable the user to control the various mode settings of the machine console. A Control command may also be used in conjunction with a Write command.

Sequencing Operations

The user can issue multiple commands to the console without having to dequeue the FBR between each command. Once the machine console has started working on a command, it will not receive another command until the first command has completed and the FBR is posted. The only exception to this is the Read Modified command. Because of the nature of the Read Modified command, it must be possible to cancel the Read Modified command or write data on the screen while the Read Modified command is pending and the operator is entering data. Therefore, it must be possible for the machine console to receive either a Cancel or Write command. If any other command is received, it is flagged with a sequencing error, and the FBR for that command is immediately returned to the user.

Source/Sink Data (SSD) Area

The SSD specified in an REQIO instruction is a user I/O buffer either containing RIUs (request information unit) to be transferred with a Write command or receiving the RIUs transferred with a Read command. (The console defines an RIU as 1 byte.) The I/O buffer areas should be doubleword aligned for performance considerations. If the buffer is not doubleword aligned for Write commands, the data is moved into an internal buffer before processing begins. For Read command operations, the data received from the device is always moved into an internal buffer during processing and then moved out to the user's I/O buffer. The data stream transferred or received for each command is discussed in this section.

Write Command Data Stream

The data stream for a Write command must contain buffer control commands along with the characters that are to be written into the display buffer. Each Write command data stream must begin with a set buffer address control order to specify where to begin writing data into the buffer. The set buffer address control order is a hex 11 immediately followed by a 2-byte row/column address of the desired position on the screen. The first address byte can assume values from 0 to 15 (hex 00 to hex 0F) to select one of the 16 lines where data can be displayed. The second address byte has a valid range of values from 0 to 63 (hex 00 to hex 3F) identifying one of the 64 columns (horizontal display positions). Thus, a data stream to start writing in the first buffer location (upper-left corner of the screen) would specify hex 110000 followed by the data to be written into the display buffer at that position. Multiple set buffer address control orders can be included in the data stream to write data at scattered locations in the buffer. If fields are overlapped, the last set buffer address control order will take precedence. The console supports a maximum Write command data stream length of 2048 bytes.

The user controls the position of the cursor by including an insert cursor control order in the data stream. The insert cursor control order is a hex 13 and causes the cursor to be inserted at the current buffer address. The insert cursor control order does not cause the buffer address to be incremented. The cursor can be inserted and the next character written into the same display buffer location.

A repeat through address control order causes a character to be propagated throughout an entire field. The control order is a hex 02 immediately followed by a 2-byte target address (row/column) and the hexadecimal value to be propagated from the current buffer location up to, and including, the target address location.

In addition to building a data stream containing buffer control orders and displayable data, the user is also responsible for formatting the display screen. A field in the display buffer is defined as an attribute character followed by all characters up to the next attribute character. These attribute characters are nondisplayable and unalterable characters that occupy one position in the buffer and appear as blanks on the screen. The attributes specified by the attribute character apply to all the following positions until the next attribute character is encountered. Because of automatic address wrap around at the end of the display buffer, one attribute character in any position defines the entire buffer as a single field.

The format of the attribute character and the individual bit assignments are shown in Figure 23-2. For example, the attribute character for a nonunderlined, unprotected, displayable field that has not been modified is hex 20.

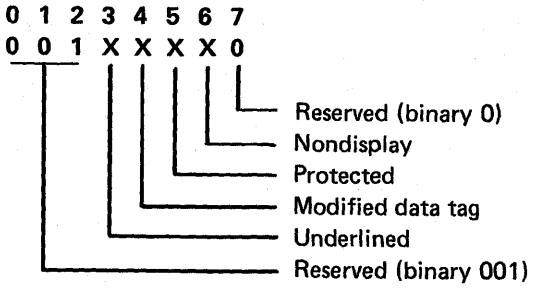


Figure 23-2. Attribute Character

All combinations of attributes are valid although some combinations would probably not be used. For example, if underline and nondisplay are specified, the characters are not displayed but the underline is. As another example, a protected field can be created with the MDT (modified data tag) bit on, in which case the operator cannot update the field, yet a Read command passes the field back to the user.

The user must ensure that at least one attribute character is always on the screen (one field defined).

If the Write command data stream contains any unprintable characters, they are shown as blanks on the screen but are read back with their original value.

The Write command data stream can be built in four ways, which cause an error. The four causes are:

- Failure to begin the Write command data stream with a set buffer address control order.
- Ending the data stream with a hex 11 (not enough bytes for all 3 bytes of the set buffer address control order).
- Ending the data stream with a hex 02 (not enough bytes for all 4 bytes of the repeat through address control order).
- A Write command data stream that results in a screen buffer with no attribute byte.

In each case, the console buffer is partially updated, and the first byte of the control value field has been processed. Therefore, for an invalid screen format error, the programming system should re-create the entire screen buffer before continuing.

Read Buffer Command Data Stream

The 1027-byte data stream that is transferred on a Read Buffer command has the following format:

Bytes 0-1	Cursor location	Char(2)
Byte 2	Keyboard status	Char(1)
• Bits 0-1	Reserved (binary 0)	
• Bits 2-3	Keyboard 10 – Locked 01 – Unlocked	
• Bits 4-5	Alarm 10 – On 01 – Off	
• Bits 6-7	Attention light 10 – On 01 – Off	
Bytes 3-1026	Display buffer contents	Char(1024)

The Read Buffer command data stream is truncated if the data length field in the SSR specifies less than 1027 bytes. The keyboard status byte can be used directly in the second byte of the control value field or the precommand control field of the RD to re-create the state of the machine console at the time the Read Buffer command was executed.

Read Modified Command Data Stream

Completion of the Read Modified command is controlled by the operator. When the operator presses an Enter/Rec Adv, CF, Sys Req, Help, Roll ↑ (Roll Up), or Roll ↓ (Roll Down) key, the following information is included in the data stream:

Byte 0	Function key identification	Char(1)
Bytes 1-2	Cursor location (row/column)	Char(2)
Bytes 3-5	Set buffer address (of first field with the MDT bit on)	Char(3)
	• Data from that field	Char(*)
	• Set buffer address (of next field with the MDT bit on)	Char(3)
	• Data from that field (and so on for all fields with the MDT bit on in the attribute byte)	Char(*)

The length of the data stream transferred is specified by the RIU segment count field in the FBR. The data stream may be truncated because of the entry in the data length field of the RD.

The possible values of the function key identification field are as follows:

Hex F1 – Enter	Hex F3 – Help
Hex F4 – Roll ↓ (roll down)	Hex F5 – Roll ↑ (roll up)
Hex FC – System Request	Hex B2 – CF14
Hex 31 – CF1	Hex B3 – CF15
Hex 32 – CF2	Hex B4 – CF16
Hex 33 – CF3	Hex B5 – CF17
Hex 34 – CF4	Hex B6 – CF18
Hex 35 – CF5	Hex B7 – CF19
Hex 36 – CF6	Hex B8 – CF20
Hex 37 – CF7	Hex B9 – CF21
Hex 38 – CF8	Hex BA – CF22
Hex 39 – CF9	Hex BB – CF23
Hex 3A – CF10	Hex BC – CF24
Hex 3B – CF11	
Hex 3C – CF12	

If the function key that has been pressed is either the Help key or the Sys Req key, then an RIU segment length of 3 is returned in the FBR. If any other function key was pressed, the fields with the MDT bit on follow in order.

If no fields have been modified, the data stream consists only of the function key identification and the cursor location. However, space should be provided in the specified data area for the maximum amount of data that could be transferred (the sum of the lengths of all unprotected fields plus the necessary control and address bytes). The maximum possible data stream that could be returned (every display position an attribute byte) is 4099 bytes.

Control Data Area

No data area need be reserved for use with control-only commands because the control functions involve no data transfer.

Cancel Data Area

No data area need be reserved for use with cancel commands because cancel functions involve no data transfer.

Feedback Record (FBR)

The FBR is the vehicle for signaling to the program that the functions requested in a Request I/O instruction have completed and for signaling whether they completed successfully or not.

The format of the FBR is as follows:

Bytes 0-15	SSR address	Space pointer
Bytes 16-17	Request identification	Bin(2)
Bytes 18-19	Error summary	Bin(2)
Bytes 20-21	RD number	Bin(2)
Bytes 22-23	RIU segment count	Bin(2)
Bytes 24-63	Device-dependent area	Char(40)
• Bytes 24-25	Device-dependent error code	Char(2)
• Bytes 26-27	Hardware error code	Char(2)
• Bytes 28-35	Time stamp	Char(8)
• Bytes 36-37	Operational unit number	Char(2)
• Bytes 38-63	Reserved (binary 0)	Char(26)

See *Request I/O (REQIO)* in Chapter 17 for descriptions of the source/sink request address and request identification fields.

The error summary field defines the status of the Request I/O instruction as defined under *Request I/O (REQIO)* in Chapter 17. The specific values possible for the machine console are listed in Figure 23-3.

The RD number is the index of the last RD processed or the RD in error if an error is indicated. For the console, the RD number is always hex 01.

The RIU segment count is the number of bytes transferred to the SSD by this request.

The device-dependent area is all binary 0's unless the presence of device-dependent data is indicated by the error summary value in the error summary field. (See *Description* under *Request I/O (REQIO)* in Chapter 17 for the error summary field definition.) If the device-dependent data is present, the fields have the values shown in Figure 23-3.

As shown in Figure 23-3, the device-dependent error code is a further categorization of the hardware error codes.

The hardware error code is the same as the data logged in the hardware error log and indicates the specific hardware error encountered. The possible values are shown in Figure 23-3.

The time stamp and operating unit number are the same values present in the hardware error log entry and in the LUD event related data (see *Chapter 21. Event Specifications).* These values are used to correlate the FBR, the LUD event-related data, and the error log entry for maintenance purposes.

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
0000	N/A	N/A	Normal completion	N/A
0008	N/A	N/A	REQIO (continue) response	N/A
4009	N/A	N/A	REQIO partially complete–MODLUD (reset)	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (reset) instruction followed by an MODLUD (de-activate) instruction must be issued to destroy the session.
400A	N/A	N/A	REQIO not complete MODLUD (reset)	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (reset) instruction followed by an MODLUD (de-activate) instruction must be issued to destroy the session.
4014	N/A	N/A	Read Modified command canceled	N/A

Figure 23-3 (Part 1 of 3). Machine Console Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
5014	N/A	N/A	Cancel command incomplete because Read Modified command completed first	N/A
C010	N/A	N/A	Operational unit task failure	LUD must be varied off.
C044	N/A	N/A	Read data truncated because of insufficient buffer space	If MDTs were not reset, correct problem and reissue command.
C045	N/A	N/A	SSD byte space too large on Write command	Correct RD, reissue REQIO instruction, and issue REQIO (continue) instruction.
C086	N/A	N/A	Invalid RD count	Correct problem, reissue REQIO instruction, and issue REQIO (continue) instruction.
C087	N/A	N/A	Invalid RD command or command modifier or reserved fields set incorrectly	Correct RD, reissue REQIO instruction, and issue REQIO (continue) instruction.
D080	N/A	N/A	Command out of sequence, Read command only	Issue REQIO (continue) and correct problem.

Figure 23-3 (Part 2 of 3). Machine Console Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E015	0001		I/O Error	Call your service representative. The LUD failure event is signaled for these cases.
		0101	– Channel error	
		0201	– CRT cable disconnected	
		0202	– Keyboard cable disconnected	
		0203	– Screen buffer parity	
		0204	– Screen buffer and DBI parity	
		0205	– Keyboard overrun	
		0206	– Invalid keyboard scan code	
		0301	– Post event	
		0302	– Invalid disconnect	
		0401	– Invalid BSTAT/DSTAT data	
		0402	– FOB time-out	
E015	0002		I/O Error	Call your service representative.
		0105	– Command reject	
		0403	– Operating program error	
E015	0003	blank	I/O Error	Correct problem, rewrite entire screen, reissue REQIO instruction and issue REQIO (continue) instruction.
		0104	– Invalid screen format	

Figure 23-3 (Part 3 of 3). Machine Console Error Summary Values

Machine Console Errors

The user of the machine console must know the state of the machine console after an error occurs.

Cancel Command Error: If the user is attempting to cancel a Read Modified command, a device error can occur on either or both of the commands. If the console detects a device error on the Read Modified command, no retry is done and the value in the error summary field of the Read command indicates that the command canceled, and the error summary field in the Cancel command indicates that the command completed. If the device error condition still exists during ensuing commands, normal error recovery takes place at that time.

Write with Read Modified Command Error: The Write with Read Modified command involves the console canceling the outstanding Read Modified command, executing the Write command, and then reissuing the Read Modified command. If the Write command FBR is returned with an error summary field indicating an error, the Read Modified command has not been canceled and is still outstanding. The machine console command has to receive a Request I/O (continue) instruction before the Read Modified command is reexecuted. It is possible that the Read Modified command can complete before it can be canceled. If the Read Modified command completes with a device error, the Write command does not execute until the machine console receives a Request I/O (continue) instruction.

General Device Errors: Whenever the FBR error summary field is returned with a value greater than hex 8000, the error conditions must be cleared by issuing a Request I/O (continue) instruction, or by issuing a Modify LUD instruction to reset the LUD.

If the FBR error summary field is returned with a value of hex C010, the machine console is in a failure mode and before it can be functional again, it must be varied off and then varied on again. If the FBR error summary field is returned with values of hex E015 (device-dependent codes of hex 0001 and hex 0003), the state of the machine console is unknown and if the user wishes to continue operations, the entire screen and control states must be rewritten. If any of the other summary error codes greater than hex 8000 are returned, the state of the machine console is restored to the state it was in before the error occurred. Only a Request I/O (continue) instruction is needed to resume operations.

If a Request I/O (continue) instruction is needed and the user issues a normal Request I/O instruction, a time-out most likely occurs when an attempt is made to dequeue the FBR for the normal Request I/O instruction.

Events

In addition to the events specified under *Request I/O (REQIO)* in Chapter 17, the following events are also signaled. For a complete description of the following events, see Chapter 21.

- Supervisory service request event (hex 000B 04 01)

  This event is signaled by the IOM when a Read Modified command completes with a System Request key.

- LUD contact event (hex 000B 06 01, 02)

  This event is signaled when the LUD vary on processing is completed by the MSCP. Subtype hex 01 is signaled upon successful contact; however, subtype hex 02 is never signaled for the machine console. In this case the Modify LUD (Vary on) instruction signals an exception.

- LUD failure event (hex 000B 08 01)

  The LUD failure event is signaled if the device should not be used again until the problem is corrected. The event is signaled if the recovery action for an error requires that the user call his service representative.

  The event-related data for this event consists of:

Bytes 0-1	Hardware error log code
Bytes 2-9	Error log time stamp
Bytes 10-11	Operational unit number (hex 0002)
Bytes 12-13	Optional data (not used)

- Request I/O complete event (hex 000B 09 01)

  The request I/O complete event is signaled if the user requests it. See *Request I/O (REQIO)* in Chapter 17 for details.

- Request I/O response queue destroyed event (hex 000B 0A 01)

  The response queue destroyed event is signaled if the user truncates or destroys the request I/O response queue while the machine is processing Request I/O instructions.

Exceptions

The following table gives the cases where the source/sink resource not available exception (hex 3404) is signaled when a Modify LUD instruction for the machine console LUD is executed.

Command	Defect Code (hex)	Device-Specific Return Code (hex)	Meaning
Suspend Session	2312	1201	Suspend session rejected because a Read Modified command is outstanding.
Quiesce Session	2313	1301	Quiesce rejected because a Read Modify command is outstanding.
		1302	Quiesce rejected because a terminating error condition exists.

Note: See *Chapter 20. Exception Specifications* for detailed descriptions of these exceptions. The source/sink resource not available exception is not signaled for the machine console vary on, activate, or resume operations. If any hardware failures have occurred, an LUD contact event is signaled. These hardware failures cause additional Request I/O instructions to fail; however, the state of the LUD is not changed.

5424 PROGRAMMING CONSIDERATIONS

The basic object of control for the 5424 is the LUD (logical unit description). All references to a device are made relative to the LUD. The Create, Modify, and Destroy LUD instructions establish and control the environment in which the 5424 operates. The Request I/O instruction controls the device and causes it to read, feed, punch, and print cards.

Following power on and before requests for I/O operations can be accepted, the following steps must occur.

1. An LUD must be created through the use of the Create LUD instruction.

2. The 5424 must be varied on through the use of the Modify LUD instruction.

3. The 5424 must be made active through the use of the Modify LUD instruction.

Create Logical Unit Description (CRTLUD) Instruction

For the 5424, the following fields in the LUD instruction template (see *Create Logical Unit Description (CRTLUD)* in Chapter 17) must be initialized as shown:

Field Name	Entry
LUD type	Char 00
Device type	Char 5424
Model number	Choose one: Char ƀƀA1, Char ƀƀK1, Char ƀƀK2, Char ƀƀA2, or Char ƀƀK3
LUD operational unit number	Hex 0019
Power control	Hex 0100
Session definition data	All binary 0
Load/dump indicator	All binary 0
Specific characteristics length	Hex 0000
Retry value length	Hex 0000
Error threshold length	Hex 0000
Device-specific contents length	Hex 0000
Device-specific modifiable length	Hex 0000

Modify Logical Unit Description (MODLUD) Instruction

The following is a list of functions the 5424 supports through the Modify LUD instruction.

- Power on
- Power off
- Vary on
- Vary off
- Activate
- De-activate
- Suspend
- Quiesce
- Reset
- Resume (activate after suspend quiesce, or reset)

Note: If a Modify LUD (reset) instruction is issued while request I/O operations are pending in the machine, the reset operation takes approximately 12 seconds to complete. This ensures that the device can be halted without leaving cards in the transport. If no request I/O operations are pending, the reset operation completes normally.

See Chapter 17 for the meaning of each function.

Request I/O (REQIO) Instruction

The Request I/O instruction is used to send I/O requests to the device.

The SSR (see *Create Logical Unit Description (CRTLUD)* in Chapter 17) for a Request I/O instruction to the 5424 contains the following values:

Field	Value
Source/sink object	Pointer to the 5424 LUD
Response queue	Pointer to the response queue
Source/sink data area	Space pointer
Optional pointer	Reserved (binary 0)
Request priority	See *Request I/O (REQIO)* in Chapter 17.
Request identification	See *Request I/O (REQIO)* in Chapter 17.
Function field	Hex 80
Control field	N or C
Key length	Bin(2)
Key offset	Bin(2)
RD count	Bin(2)
RD offset	Bin(2)

The 5424 supports multiple RDs in a Request I/O SSR. The RD offset field indicates the offset from the start of the SSR to the first 16-byte RD. The number of RDs is specified by the RD count field. An RD represents a command to the 5424 to process one or more cards. The data representing one card is called a segment, and the segment count field in the RD indicates the number of cards to be processed by the RD.

The SSD space pointer identifies a contiguous data area comprising, in the order processed, the segments within an RD for the RDs corresponding to the SSR. The size of the area needed is the sum over each RD of: (number of bytes per segment) X (number of RIU segments specified in the RD). The SSD space has no boundary alignment requirements because all data is placed in an internal buffer by the machine.

The format of the RD is as follows:

Byte 0 — Command — Char(1)
- Hex 14 – Feed
- Hex 02 – Read
- Hex 11 – Punch
- Hex 21 – Print 3 lines
- Hex 61 – Print 4 lines
- Hex 13 – Read and punch
- Hex 23 – Read and print 3 lines
- Hex 63 – Read and print 4 lines
- Hex 31 – Punch and print 3 lines
- Hex 71 – Punch and print 4 lines
- Hex 33 – Read, punch, and print 3 lines
- Hex 73 – Read, punch, and print 4 lines

Bytes 1-6 — Command modifier — Char(6)

- Byte 1 — Char(1)
  - Bit 0 — Select feed path
    - 0 = Primary feed path
    - 1 = Secondary feed path
  - Bit 1 — Stacker select
    - 0 = Default stacker (chosen based upon the operation to be performed)
    - 1 = Stacker number specified in following 2 bits
  - Bits 2-3 — Stacker number
    - 00 = Stacker 4
    - 01 = Stacker 1
    - 10 = Stacker 2
    - 11 = Stacker 3
  - Bits 4-7 — Reserved (binary 0)
- Byte 2 — Multi-use data (MUD) field — Char(1)
  - 0 = Each function within the command uses a separate data field in the data area
  - 3 = Fields 2 and 3 are the same field
- Bytes 3-6 — Reserved (binary 0) — Char(4)

Byte 7 — RIU segment type — Char(1)
Hex 00 = Contiguous segments

Bytes 8-9 — Segment count — Bin(2)

Bytes 10-11 — Segment length — Bin(2)

Bytes 12-15 — Reserved (binary 0) — Char(4)

The command field indicates the type of operation to be performed. The four basic commands supported by the 5424 are the Feed, Read, Punch, and Print commands.

The Feed command moves a card from the specified hopper to the corresponding wait station. The card is not read during this command.

An error in feeding a card causes a summary status of unrecoverable I/O error to be returned in the FBR, and the appropriate device-dependent bits are turned on.

The Read command moves a card from the specified hopper to the corresponding wait station. The data contained in all 96 columns of the card is transferred to the address specified by the data area. The data read is checked to ensure that it is read correctly.

An error during a read operation causes a summary status of unrecoverable error to be returned in the FBR, and the appropriate device-dependent status bits are turned on.

The Punch command moves a card from one of the wait stations, through the punch station, cornering station, and print station and to the stackers. As the card passes through the punch station, data is punched in the cards. The punching is checked to ensure that the correct data is punched in the card. No printing is done during the punch commands.

An error during a punch operation causes a summary status of unrecoverable error to be returned in the FBR, and the appropriate device-dependent status bits are turned on.

The Print command moves a card from the selected wait station, through the punch and cornering stations, and into the print station where three or four lines of 32 characters each are printed on the card. After printing is complete, the card is moved into the selected stacker.

The 96-byte or the 128-byte print data area is printed on the card in the following manner:

Line 1 – Leftmost address to byte 32

Line 2 – Bytes 33 through 64

Line 3 – Bytes 65 through 96

Line 4 – Bytes 97 through 128
(if the fourth line of print is specified)

The 5424 prints any of the 64 characters in the card code. Any of the characters in the 256-character EBCDIC set, not included in the card code, print as blanks without signaling the program.

An error during a print operation causes a summary status of unrecoverable error to be returned in the FBR, and the appropriate device-dependent status bits are turned on.

These commands can be used separately or in combination. Request I/O instructions specifying combined operations proceed in the same manner described for individual operations except that one card is fed from the wait station and punched and/or printed before stacking. The next card is fed from the specified hopper into the wait station during punching. If the Read command is specified, the data in the card is read into storage.

The command modifier field (byte 1) is used to control the feed path and stacker selection. The MUD field specifies the use of the data fields within the SSD. Field 1 is the read field, field 2 is the punch field, and field 3 is the print field. If the command includes 4 print lines, the specified MUD is ignored, and it defaults to MUD equals 0. Command modifier bytes 3, 4, 5, 6, and 7 are reserved.

The RIU segment type field must be hex 00 for the 5424. This indicates that the RIU segments are contiguous in the SSD and that there are no gaps or control characters between segments in the SSD.

The segment count field contains the number of cards to be processed. This entry can be 1 through 32 767. If 0 is specified, one card is processed. A negative number entered in the segment count field causes an error to be signaled.

The segment length field is the total number of bytes to be read, and/or punched, and/or printed on a single card. When a read or punch operation is specified in the Request I/O instruction, a segment length of 96 bytes is assumed. The print record is either 96 or 128 bytes long depending on whether 3 or 4 lines of print are requested. When the 5424 punches and prints the same data, the segment length field value for the punch/print operation must be 96 bytes. The punch record must always be 96 bytes long.

The following example shows the entries in the RD and SSD fields to read five records of 96 bytes each.

Request Descriptor

CMD	MUD	Reserved	Segment Count	Segment Length	Reserved
02	0		5	96	

Source/Sink Data Area

Read Data Field 1	Read Data Field 1	Read Data Field 1	Read Data Field 1	Read Data Field 1
96 bytes	96 bytes	96 bytes	96 bytes	96 bytes

The following example shows the entries in the RD and SSD fields to punch and print three cards with the same data.

Request Descriptor

CMD	MUD	Reserved	Segment Count	Segment Length	Reserved
31	3		3	96	

Source/Sink Data Area

Punch/Print Data Field 2/3	Punch/Print Data Field 2/3	Punch/Print Data Field 2/3
96 bytes	96 bytes	96 bytes

The following example shows the entries in the RD and SSD fields to punch and print different data into one card and read the next card.

Request Descriptor

CMD	MUD	Reserved	Segment Count	Segment Length	Reserved
33	0		1	288	

Source/Sink Data Area

Read Data Field 1	Punch Data Field 2	Print Data Field 3
96 bytes	96 bytes	96 bytes

Note: The punch data (Field 2) and the print data (Field 3) will be printed on the card that was at the wait station when the command was issued.

FEEDBACK RECORD AND ERROR RECOVERY PROCEDURE

The format of the feedback record is as follows:

Bytes 0-15	SSR address	Space pointer
Bytes 16-17	Request identification	Bin(2)
Bytes 18-19	Error summary	Char(2)
Bytes 20-21	RD number	Bin(2)
Bytes 22-23	RIU segment count	Bin(2)
Bytes 24-63	Device-dependent area	Char(40)
• Bytes 24-25	Device-dependent error code	Char(2)
• Bytes 26-27	Hardware error code	Char(2)
• Bytes 28-35	Time stamp	Char(8)
• Bytes 36-37	Operating unit number	Char(2)
• Bytes 38-39	BSTAT	Char(2)
• Bytes 40-47	DSTAT	Char(8)
• Bytes 48-63	Reserved (binary 0)	Char(16)

See *Request I/O (REQIO)* in Chapter 17 for descriptions of the request address and request ID fields.

The error summary field defines the status of the Request I/O instruction as defined under *Request I/O (REQIO)* in Chapter 17. The specific values possible for the 5424 are shown in Figure 23-4.

The RD number is the index of the last RD processed or the RD in error if an error is indicated.

The device-dependent area is all binary 0 unless the presence of device-dependent data is indicated by the error summary value in the error summary field. (See *Description* under *Request I/O (REQIO)* in Chapter 17 for the error summary field definition.) If device-dependent data is present, the field has the values shown in Figure 23-4.

As shown in Figure 23-4, the device-dependent error code is a further categorization of the hardware error codes.

The hardware error code is the same value as that logged in the hardware error log and indicates the specific hardware error encountered. The possible values are shown in Figure 23-4.

The time stamp and operating unit number are the same values present in the hardware error log entry and are used to correlate the FBR and the error log entry for maintenance purposes.

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Codes (hex)	Meaning	Recovery Action
0000	N/A	N/A	Normal completion	N/A
0008	N/A	N/A	REQIO (continue) response	N/A
C009	N/A	N/A	Partially processed request terminated because of reset session	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (reset) instruction followed by an MODLUD (de-activate) instruction must be issued to destroy the session.
C00A	N/A	N/A	Unprocessed request because of reset session	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (reset) instruction followed by an MODLUD (de-activate) instruction must be issued to destroy the session.
C016	N/A	N/A	End of file	An REQIO (continue) instruction must be issued to restart processing.
C044	N/A	N/A	SSD too small	Correct SSD, reissue REQIO instruction, and issue an REQIO (continue) instruction.
C084	N/A	N/A	Invalid SSD pointer	Correct SSD pointer, reissue REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C085	N/A	N/A	Invalid function	Correct REQIO instruction, reissue REQIO instruction, and issue an REQIO (continue) instruction.
C087	N/A	N/A	Invalid RD	Correct RD, reissue REQIO instruction, and issue an REQIO (continue) instruction.

Figure 23-4 (Part 1 of 3). 5424 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Codes (hex)	Meaning	Recovery Action
E010	N/A		Retryable hardware errors	Check the BSTAT and DSTAT fields (defined later in this chapter) for error bits and determine appropriate recovery. An REQIO (continue) instruction must be issued to restart processing.
		1916	Read check	
		1917	Punch invalid	
		1918	Punch check	
		191A	Emitter check	
		191F	Command reject; command was issued for a path with an empty wait station (primary or secondary) and the command was not a Feed or a Read command	
		1980	Hopper check	
E010	0001	1920	IOC parity error from DBI	Call your service representative. An LUD failure event is also signaled.
		1921	IOC internal parity error	
		1922	CSA parity error	
E010	0002	1901	Feed check 1	Check the BSTAT and DSTAT fields (defined later in this chapter) for error bits and determine appropriate recovery. An REQIO (continue) instruction must be issued to restart processing.
		1902	Feed check 2	
		1903	Feed check 3	
		1904	Feed check 4	
		1905	Feed check 5	
		1906	Feed check 6	
		1907	Feed check 7	
		1908	Feed check 8	
		1909	Feed check 9	
		190A	Feed check 10	
		190B	Feed check 11	
		190C	Feed check 12	
		190D	Feed check 13	

Figure 23-4 (Part 2 of 3). 5424 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Codes (hex)	Meaning	Recovery Action
E010	0002	190E	Feed check 14	
		190F	Feed check 15	
		1910	Feed check 16	
		1911	Feed check 17	
		1912	Feed check 18	
		1913	Feed check 19	
		1914	Feed check 20	
		191E	Power fault	
E010	0003	191B	Print sync check	Check the BSTAT and DSTAT fields (defined later in this chapter) for error bits and determine appropriate recovery. An REQIO (continue) instruction must be issued to restart processing.
		191C	Print home check	
		191D	Print clutch check	
E010	0004		Permanent hardware errors	Call your service representative. An LUD failure event is also signaled.
		1915	Read cell defective	
		1919	Device address out of sequence	
		191F	Command reject	
		1923	Invalid OU	
		1924	Invalid disconnect	
		1925	Channel error	
		1926	Function operation blocked time-out	
		1928	Operation program error	
		1929	Invalid BSTAT/DSTAT data	
		192A	Read sense/data store	
E010	0005	N/A	Cover open error	Check the DSTAT field (defined later in this chapter) for error bits and determine appropriate recovery. An REQIO (continue) instruction must be issued to restart processing.
E010	0006	N/A	Chip box full	Correct the condition, reissue REQIO instruction, and issue an REQIO (continue) instruction.
E087	0001	N/A	Reserved field in RD is not 0.	Correct the RD, reissue REQIO instruction, and issue an REQIO (continue) instruction.

Figure 23-4 (Part 3 of 3). 5424 Error Summary Values

BSTAT Field Definitions

Byte	Bit	Description
0	0	Reserved for the channel
	1	Operation program error
	2	Halt
	3	Channel error
	4	I/O exception
	5	Command reject
	6	I/O error
	7	Command complete
1	0	Power fault
	1	Power on
	2	Reserved
	3	Hopper 1 empty
	4	Hopper 2 empty
	5	Stacker full
	6	End of data delimiter
	7	Reserved

DSTAT Field Definitions

Byte	Bit	Description
0	0-7	Same as BSTAT byte 0 bits 0-7 of previous command
1	0-7	Same as BSTAT byte 1 bits 0-7 of previous command
2	0	Hopper check
	1	Reserved
	2	Cover open error
	3-7	Feed check identifiers (hex values 01 through 14 correspond to feed checks 1 to 20)
3	0	Cells defective
	1	Read check
	2	Reserved
	3	NPRO 1 required
	4	NPRO 2 required
	5	Punch invalid
	6	Punch check
	7	Reserved
4	0	Halt (same as DSTAT Byte 0 Bit 2)
	1	Disconnect
	2	Channel parity error
	3	Device address recognition out of sequence
	4	Fire emitter check
	5	Print sync check
	6	Print home check
	7	Print clutch check
5	0	Stacker full
	1	Chip box full switch
	2	Cover switch
	3-4	Indicates which stacker was selected for last card to leave the transport:

Bits 3-4	Stacker
01	1
10	2
11	3
00	4

	5	Card in wait 2 (secondary path)
	6	Card in wait 1 (primary path)
	7	Motor ready
6	0	Reserved
	1	Hopper cycle 0 = Hopper cycle complete 1 = Hopper cycle not complete
	2-4	Reserved (binary 0)
	5-7	Count of cards in the transport
7	0-7	Reserved

Events

In addition to the events specified under *Request I/O (REQIO)* in Chapter 17, the following events are signaled. For a complete description of the following events, see Chapter 21.

- LUD contact event (hex 000B 06 01, 02)

  This event is signaled when the LUD vary on processing is completed by the MSCP. Subtype hex 01 is signaled upon successful contact; however, subtype hex 02 is never signaled for the 5424. In this case the Modify LUD (vary on) instruction signals an exception.

- LUD failure event (hex 000B 08 01)

  The LUD failure event is signaled if the device should not be used again until the problem is corrected.

  This event is signaled whenever the device has an error in which the recovery action requires the user to call his service representative. The MFCU LUD should be varied off before varying it back on and attempting further operations.

  The event-related data for this event consists of:

Bytes 0-1	Hardware error log code
Bytes 2-9	Error log time stamp or 0's
Bytes 10-11	Operational unit number (hex 0019)
Bytes 12-13	Optional data (not used)

- Operator intervention required event (hex 000B 07 01)

  This event is signaled after the machine detects that an operator action is needed for the device.

  The event-related data for this event consists of:

Bytes 0-1	Hardware error log code
Bytes 2-9	Error log time stamp
Bytes 10-11	Operational unit number (hex 0019)
Bytes 12-13	Optional data has the following meaning:

Condition	Description and Operator Action
Byte 12, bit 0	Hopper 1 is empty. Make feed path 1 ready. After correcting the problem, press the start key.
Byte 12, bit 1	Hopper 2 is empty. Make feed path 2 ready. After correcting the problem, press the start key.
Byte 12, bit 2	A stacker is full or the device is not ready. To recover, correct the problem and press the start key.

  The remaining bits of byte 12 and all of byte 13 are not used and are 0.

- Request I/O complete event (hex 000B 09 01)

  This event is signaled if the user requests it. See *Request I/O (REQIO)* in Chapter 17 for details.

- Request I/O response queue destroyed event (hex 000B 0A 01)

  This event is signaled if the user truncates or destroys the request I/O response queue while the machine is processing Request I/O instructions.

Exceptions

The following table gives the cases where the source/sink resource not available exceptions (hex 3404) are signaled when a Modify LUD instruction for the machine console LUD is executed.

Command	Defect Code (hex)	Device-Specific Return Code (hex)	Meaning
Vary On	2302	0001	I/O failure has occurred. LUD failure event has been signaled.
Power On	2306	0601	Power failure. LUD failure event has been signaled.
Power Off	2307	0601	Power failure. LUD failure event has been signaled.
Suspend Session	2312	1203	Suspend session rejected because an operator intervention condition exists.
Quiesce Session	2313	1302	Quiesce session rejected because a terminating error condition exists.
		1303	Quiesce session rejected because an operator intervention condition exists.

3262/5211 PRINTER PROGRAMMING CONSIDERATIONS

The basic object of control for the 3262/5211 printers is the Logical Unit Description (LUD). All references to a device are made with respect to the LUD. The Create, Modify, and Destroy LUD instructions establish and control the environment in which the printer operates. The Request I/O instruction controls the device and causes it to print data.

Before Request I/O operations can be accepted, several steps must occur.

1. An LUD must be created through use of the Create LUD instruction.

2. The printer must be varied on through use of the Modify LUD instruction.

3. The LUD must be made active through use of the Modify LUD (activate) instruction.

4. The device-specific area contents must be initialized through use of the Modify LUD (device-specific area) before any print operations can be performed. If the device-specific area contents are not initialized, a command reject feedback record will be returned for all requests to print.

3262/5211 PRINTER CREATE LOGICAL UNIT DESCRIPTION (CRTLUD) TEMPLATE

The following fields of the logical unit description template must be initialized as indicated:

Field Name	Entry
LUD type	Char 00
Device type	Char 5211 for the 5211 Char 3262 for the 3262
Model number	Char 0002 for the 5211 Char ƀƀA1 for the attached 3262 Char ƀƀB1 for the stand-alone 3262
LUD operational unit number	Hex 0018 for the first printer Hex 0058 for the second printer
Power control	Hex 0100 for all models
Session definition data	Bin 0
Load/dump indicator	Bin 0
Specific characteristics length	Hex 0000
Retry value length	Hex 0000
Error threshold length	Hex 0000
Device-specific contents length	485
Device-specific modifiable length	485

Device-specific area		Char(485)
Byte 0	Control flags	Char(1)
• Bits 0-1	Reserved (binary 0)	
• Bit 2	Write control 0 = No data translation 1 = Translate the data	
• Bits 3-7	Reserved (binary 0)	
Byte 1	Lines per inch	Char(1)
Byte 2	Lines per form	Char(1)
Bytes 3-4	Character set length	Bin(16)
Bytes 5-292	Character set	Char(288)
Bytes 293-484	Translate table	Char(192)

The device-specific parameters may or may not be supplied in the device-specific area of a create template. If they are not supplied at create time, the create template must still contain the 485-byte area and this area should be set to 0. The LUD will be created to contain whatever is in the template, without validating any of these parameters. These parameters are not used until after the LUD is in the active session state.

3262/5211 PRINTER MODIFY LOGICAL UNIT DESCRIPTION (MODLUD)

The following is a list of functions the 3262/5211 supports through the MODLUD instruction:

- Power on
- Power off
- Vary on
- Vary off
- Activate
- De-activate
- Suspend
- Quiesce
- Reset
- Resume (activate after suspend, quiesce, or reset)
- Modify device-specific area

See Chapter 17 for the meaning of each function.

LUD Device-Specific Area

The device-specific area contains the information used by the printer attachment to control the line spacing, form size, belt image, and translate table used by the printer. The device-specific area can be altered by a Modify LUD instruction. If any part of the device-specific area is changed, then all the device-specific area parameters are written to the printer. Changes to the device-specific area are reflected when the machine processes the next Request I/O instruction. The control flags field controls the translate mode in the machine.

If translate data is specified, the data is translated (based on the translate table) prior to printing.

For the 3262 and the 5211, the lines per inch must be either 6 or 8. When 6 lines per inch is specified, the number of lines per form can be from a minimum of 18 up to a maximum of 84. When 8 lines per inch is specified, the number of lines per form can be from a minimum of 24 to a maximum of 112.

A Modify LUD (device-specific area) instruction must be issued any time after the LUD is in the varied on state and before any printing operations are started. This causes the printer to be initialized to the desired parameters for line spacing, form size, belt image, and translate table. This set of device-specific area parameters is then used until the next Modify LUD instruction is issued to change the device-specific area parameters or until the LUD is varied off. If the Modify LUD (device-specific area) occurs while Request I/O instructions are outstanding, the new values take effect before the next Request I/O instruction is processed. The printer is not initialized in any other way.

If a printer cannot be initialized for a Modify LUD (device-specific area) instruction due to a device hardware failure, the subsequent Request I/O instruction that requests a printing operation causes a feedback record that indicates the hardware error condition encountered.

If the device-specific area parameters supplied for a Modify LUD instruction are invalid, a template value invalid exception may be signaled. If the template value invalid exception was not signaled, or if the device-specific area has not been modified since the LUD was varied on, a terminating error feedback record is signaled for the next Request I/O instruction.

3262/5211 PRINTER REQUEST I/O (REQIO) INSTRUCTION

The Request I/O instruction is used to request the 3262/5211 printer to perform its various I/O functions.

The SSR for a Request I/O instruction to the 3262/5211 printer contains the following values:

Field	Value
Source/sink object	System pointer to LUD
Response queue	System pointer to response queue
Source/sink data area	Space pointer
Optional pointer	Reserved (binary 0)
Request priority	See *Request I/O (REQIO)* in Chapter 17
Request identification	See *Request I/O (REQIO)* in Chapter 17
Function field	Hex 80
Control field	N or C
Key length	Bin(2)
Key offset	Bin(2)
RD count	0 for Request I/O (continue) instruction 1 for Request I/O instruction
RD offset	Bin(2)

The 3262/5211 printer supports only one RD for each Request I/O instruction. The RD count field must be one for Request I/O (normal) operations; however, for Request I/O (continue) operations, this field is ignored. The RD offset field indicates the offset from the start of the SSR to the 16-byte RD. The source/sink data area (SSD) contains the data to be printed corresponding to the command in the RD.

The format of the RD is as follows:

Byte 0	Commands Hex 41 = Print SCS data Hex 42 = Continue printing after error	Char(1)
Bytes 1-3	Command modifiers	Char(3)
Bytes 4-15	Reserved (binary 0)	Char(12)

Print SCS Data Command (hex 41)

The Print SCS Data command causes the data in the SSD to be printed in the format specified by the SCS command embedded in the SSD. The command modifier bytes (bytes 1-3 of the RD) contain additional information and have the following format:

Byte 1 Char(1)

- Bits 0-4 Reserved (binary 0)
- Bit 5 Unused
- Bit 6 Unprintable character detection
  0 = Signal unprintable character detected error
  1 = Do not signal an error to the user
- Bit 7 Continue
  0 = Start printing at beginning of the SSD
  1 = Retain any data saved from previous Print SCS Data command and continue printing where printing stopped

Bytes 2-3 Data block group count Bin(2)

Normally, the continue bit should be on to ensure that all data is printed.

The data block group count field is the number of 8-byte groups in the SSD. The data block count must be from 1 to 8192; otherwise, an error occurs.

The print data area must be loaded before the Print SCS Data command is issued.

Continue Printing After Error (hex 42)

The Continue Printing After Error command (hex 42) can be used to recover from an error that occurred on a print Request I/O instruction that used a Print SCS Data RD command (hex 41). When an error occurs, a Print Request I/O instruction with a Continue Printing After Error (hex 42) RD command can be issued to continue printing as if the previous error had not occurred. The SCS data is saved from the previous SCS print, and printing is continued from where the error occurred. The command modifier bytes (bytes 1-3 of the RD) contain additional information and have the following format:

Byte 1 Char(1)

- Bits 0-5 Reserved (binary 0)
- Bit 6 Unprintable character detection
  0 = Signal unprintable character detected error
  1 = Do not signal an error to the user
- Bit 7 Reserved

Bytes 2-3 Reserved

The SSD pointer value is ignored by this command.

This command should be used for only hardware errors that do not require a response. If this command is used for errors that require a response, the results are unpredictable. The user of this command should not change the print data area that was used by the Request I/O instruction that encountered the error. An invalid RD command error is signaled if the preceding Request I/O instruction did not have a valid hardware error.

A Request I/O (continue) must be issued to continue processing.

STANDARD CHARACTER STREAM (SCS)

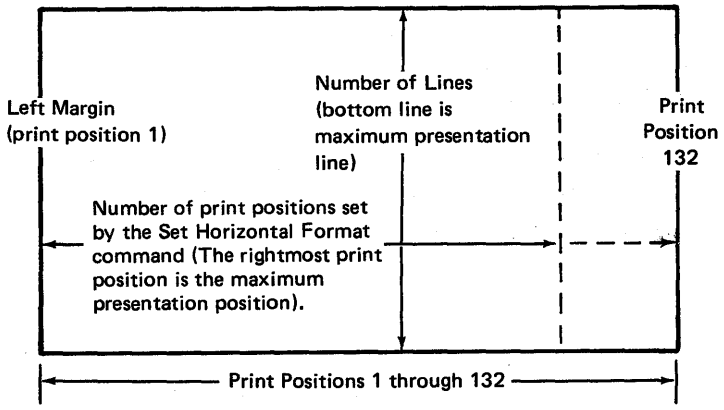
The SSD contains the SCS. The SCS is used by the 3262/5211 printer for transferring print data and SCS commands from the system to the printer. With SCS, Print Data and SCS commands are sent to the attachment in free-form; that is, SCS commands can appear anywhere within the print data stream. The SCS commands have values of hex 00 through hex 3F, and hex FF.

Print data characters have values from hex 40 through hex FE. Any character not recognized as a printable character prints as a blank and an unprintable character condition is returned in the feedback record. If the translate option is used (write control bit is on in the LUD), characters are handled as defined by the user.

SCS COMMANDS

The SCS commands control carriage operations. For a better understanding of the SCS command descriptions, see the following illustration, which shows the format of a printer form. View the form as a presentation surface on which a presentation position (the print position after an executed command) can be moved.

The contents of the print buffer will not be printed until a command that moves the carriage or that causes the horizontal print position to move left is received.



Null Command (hex 00)

This command provides the no-op functions; that is, no character is printed and no function is performed.

Interchange Record Separator Command (hex 1E)

This command is the same as the new line command described later in this section.

Line Feed Command (hex 25)

This command moves the print position to the same print position of the next line. If the print position is on the last line of a page, it is moved to the same print position of the first line on the next page.

Form Feed Command (hex 0C)

This command moves the presentation position to the top line and left margin of the next page as specified by the maximum print line parameter, which is set by the vertical format operation described under *Format Command* (hex 2B) later in this section. If the print line parameter is not specified, the maximum print line is assumed to be 1, and the presentation position moves the left margin of the next line.

Programming Note: Use the Form Feed command to move the presentation position to a new form because this command is used for page numbering by spool intercept. Do not use the New Line command or the absolute vertical parameter when moving the presentation position from one form to the next.

Carriage Return Command (hex 0D)

This command moves the presentation position to the left margin of the same line. If the current presentation position equals the left margin, the carriage is not moved.

New Line Command (hex 15)

This command moves the presentation position to the left margin of the next line. If a sequence of print characters attempts to cause the presentation position to go beyond position 132 (the maximum presentation position), an automatic new line is generated.

Format Command (hex 2B)

This command specifies the start of a formatting data stream. It is used with one of the function bytes (hex C1, C2, C7, and C8) described below. The function byte is followed by a count byte that specifies the number of bytes (including the count byte) remaining in the formatting data stream. All printer formats must be assembled before any print or carriage operation is performed.

The function bytes used with the Format command and their descriptions are described in the following text.

Set Horizontal Format Command (2BC1)

This command specifies the horizontal format (maximum presentation position) for the forms width. The format code of 2BC1 is followed by a count byte of either hex 01 or hex 02. If the count byte is hex 02, a forms width byte follows the count byte. If the count byte is hex 01, the forms width byte has a default value of 132.

Set Vertical Format Command (2BC2)

This command is a no-op to the 3262/5211 attachment.

Set Chain Image Command (2BC7)

This command is a no-op to the 3262/5211 attachment.

Set Graphic Error Action Command (2BC8)

This command is a no-op to the 3262/5211 attachment.

Bell Command (hex 2F)

This command is a no-op to the 3262/5211 attachment.

Presentation Position (PP) Command (hex 34)

This command is used with four different function parameters (hex 4C, C0, C4, and C8) to move the presentation position. Each function parameter follows the command in the data stream. A value parameter, a 1-byte number describing a position or line number, follows the function parameter in the data stream.

The function parameters used with the Presentation Position command are described in the following text.

Relative Horizontal PP Command (34C8)

This command moves the presentation position to the right. The number of positions moved is contained in the value parameter. If this causes the presentation position to be moved past the maximum presentation position, an invalid SCS error is signaled.

Absolute Horizontal PP Command (34C0)

An Absolute Horizontal PP command moves the presentation position to the position given in the value parameter. If the value parameter is less than the current presentation position, a carriage return is executed before the presentation position is moved to the designated location. If the value parameter is greater than the maximum presentation position, an invalid SCS error is signaled.

Relative Vertical PP Command (344C)

A Relative Vertical PP command causes the printer to space or skip the number of lines contained in the value parameter. The horizontal presentation position is not changed. If the value parameter is greater than the maximum presentation position, an invalid SCS error is signaled.

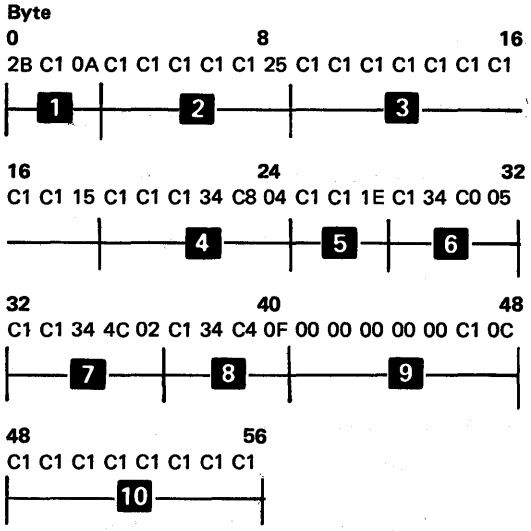
Absolute Vertical PP Command (34C4)

An Absolute Vertical PP command causes the printer to skip to the line number contained in the value parameter. The horizontal presentation position is not changed. If the value parameter is 0 or is the current line number, no operation is performed. If the value parameter is less than the current line number, the printer skips to the designated line on the next form. If the value parameter is greater than the current line number and less than or equal to the forms length, the printer skips to the designated line on the present form. If the value parameter is greater than the forms length, an invalid SCS error is signaled.

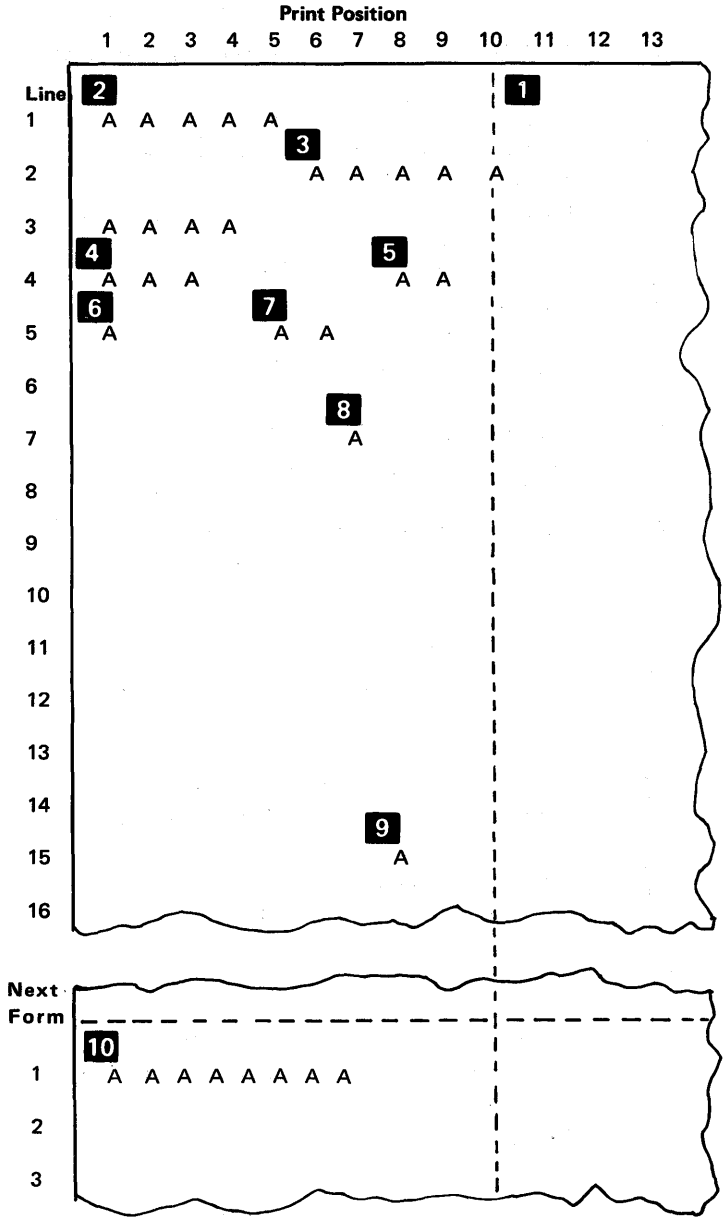
SCS EXAMPLE

This example shows commands and data embedded in a standard character stream along with the resulting printed output.

Standard Character Stream



Printed Output



1. Set the horizontal format (2BC1) to print position 10 (0A).

2. Print five A's (C1's); then execute a Line Feed command (25).

3. Print nine A's (new line is automatically generated after print position 10 because the horizontal format was set to 10); then execute a New Line command (15).

4. Print three A's; then execute a Relative Horizontal Print Position command (34C8) for four print positions.

5. Print two A's; then execute an Interchange Record Separator command (1E) (new line).

6. Print one A; then execute an Absolute Horizontal Print Position command (34C0) to print position 5 (05).

7. Print two A's; then execute a Relative Vertical Print Position command (344C) for two lines (02).

8. Print one A; then execute an Absolute Vertical Print Position command (34C4) to line 15 (0F).

9. Execute five null commands and print one A; then execute a Form Feed command (0C).

10. Print eight A's.

Note: The last eight A's will not be printed until a command is received that causes the carriage to move or that causes the horizontal print position to move to the left.

3262/5211 FEEDBACK RECORD AND ERROR RECOVERY PROCEDURE

The format of the feedback record is as follows:

Bytes 0-15	SSR address	Space pointer
Bytes 16-17	Request identification	Bin(2)
Bytes 18-19	Error summary	Bin(2)
Bytes 20-21	RD number	Bin(2)
Bytes 22-23	RIU segment count	Bin(2)
Bytes 24-63	Device-dependent area	Char(40)
• Bytes 24-25	Device-dependent error code	Char(2)
• Bytes 26-27	Hardware error code	Char(2)
• Bytes 28-35	Time stamp	Char(8)
• Bytes 36-37	Operating unit number	Char(2)
• Bytes 38-63	Reserved (binary 0)	Char(26)

See *Request I/O (REQIO)* in Chapter 17 for descriptions of the request address and request ID fields.

The error summary field defines the status of the Request I/O instruction as defined under *Request I/O (REQIO)* in Chapter 17. The specific values possible for the 3262 and 5211 printers are shown in Figure 23-5.

The RD number is the index of the last RD processed or the RD in error if an error is indicated. For the 3262 and 5211 printers, the RD number is 1.

The RIU segment count is the number of forms completed by the printer before an error occurred. If no error occurred, this field is not defined.

The device-dependent area is all binary 0 unless the presence of device-dependent data is indicated by the error summary value in the error summary field. (See *Description* under *Request I/O (REQIO)* in Chapter 17 for the error summary field definition.) If device-dependent data is present, the field has the values shown in Figure 23-5.

As shown in Figure 23-5, the device-dependent error code is a further categorization of the hardware error codes.

The hardware error code is the same value as that logged in the hardware error log and indicates the specific hardware error encountered. The possible values are shown in Figure 23-5.

The time stamp and operating unit number are the same values present in the hardware error log entry and are used to correlate the FBR and the error log entry for maintenance purposes.

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
0000	N/A	N/A	Normal completion.	N/A
0008	N/A	N/A	REQIO (continue) instruction response.	N/A
C009	N/A	N/A	Partially processed request terminated because of reset session.	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (de-activate) instruction must be issued to destroy the session.
C010	N/A	N/A	Request I/O instruction rejected because the device is disabled.	The MODLUD (reset, de-activate, vary off, vary on, and activate) instructions must be issued to restart processing.
C00A	N/A	N/A	Unprocessed request because of reset session.	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (de-activate) instruction must be issued to destroy the session.
C043	N/A	N/A	Invalid SSD boundary alignment.	Correct the boundary alignment, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C044	N/A	N/A	SSD too small.	Correct the SSD, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C084	N/A	N/A	Invalid pointer to SSD.	Replace SSD pointer with a valid pointer, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C085	N/A	N/A	Invalid function field.	Correct the function field value, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C086	N/A	N/A	Invalid RD count.	Remove the extra RDs, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.

Figure 23-5 (Part 1 of 4). 3262/5211 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	0002	4450	Invalid SCS command	Check SCS codes, correct command in error, and reprint the form. Issue an REQIO (continue) instruction to restart processing.
E010	0003	5437	Forms jam	Correct the forms jam and reprint the form. Issue an REQIO (continue) instruction to restart processing.
E010	0004	N/A	CE switch in wrong position	Check CE switch setting and retry command. Issue an REQIO (continue) instruction to restart processing.
E010	0005	3203	Unrecoverable I/O error (OU task failure)	The MODLUD (reset, de-activate, and vary off) instructions are required to clear this condition.
E010	0006	N/A	Unprintable character detected	The line has printed with blank substitution. If blank substitution is acceptable, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing. If blank substitution is not acceptable, correct the print data, correct the translate table, or correct the train image, and reprint the file. Issue an REQIO (continue) instruction to restart processing.
E010	0009	N/A	Interlock open	Close interlock on printer and reprint form. An REQIO (continue) instruction must be issued to restart processing.
E010	000B	3215	Actual belt image length does not equal specified belt length, or hex 00 was detected in image data	Put correct belt on printer or change hex 00 in image data. Issue an REQIO (continue) instruction to restart processing.

Figure 23-5 (Part 2 of 4). 3262/5211 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	000C	4008 4020 4040 4080 4447 5425 5426 5427 5430 5432 5440 5441 5443	Printer or hardware adapter failures	Operator intervention is required. Disengage paper and restore to the top of the form. Issue an REQIO (continue) instruction to restart processing. Reprint the form. If the error persists, call your service representative.
E010	000D	3203 3205 4034 4038 4452 9000	System error	Call the service representative.
E010	000E	5434 5435 5451	Unrecoverable I/O error	Call the service representative.
E010	000F	5431 5433	Belt speed check	Inspect belt and align forms. Issue an REQIO (continue) instruction to restart processing.
E010	0010	5420 5423 5424 5442	Forms alignment	Align forms. Issue an REQIO (continue) instruction to restart processing.
E010	0011	5436	Ribbon check	Inspect the ribbon. Issue an REQIO (continue) instruction to restart processing.
E010	0012	5421	Manual carriage check	Inspect belt and align forms. Issue an REQIO (continue) instruction to restart processing.
E010	0014	5444	Power fault	The MODLUD (reset, de-activate, vary off, vary on, and activate) must be issued to restart processing.

Figure 23-5 (Part 3 of 4). 3262/5211 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	0015	5422	Cable interlock	The MODLUD (reset, de-activate, vary off, vary on, and activate) must be issued to restart processing.
E010	0016	N/A	End of forms	Add paper, make the printer ready, reissue REQIO, and issue REQIO (continue) instruction.
E087	0007	N/A	Invalid RD command detected or a hex 42 RD command was issued that could not be accepted	Correct RD command byte, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	0008	N/A	Zero data block field in the RD or more than 8192 bytes of data	Correct the data block field, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	0017	N/A	RD invalid, reserved field violated	Correct the RD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
F075	N/A	4104 4102 4105 4106 4107 4108	Error during error recovery	Call the service representative.

Figure 23-5 (Part 4 of 4). 3262/5211 Error Summary Values

Events

In addition to the events specified under *Request I/O (REQIO)* in Chapter 17, the following events are signaled. For a complete description of the following events, see Chapter 21.

- LUD contact event (hex 000B 06 01)

  This event is signaled when the MSCP when vary on processing is completed for this device. Only subtype hex 01 is signaled upon successful contact; however, subtype hex 02 is never signaled since the MSCP processing is synchronous to the Modify LUD (vary on) instruction and an exception is signaled instead.

- LUD failure event (hex 000B 08 01)

  The LUD failure event is signaled if the printer has a problem that requires the service representative to be called. As part of the recovery action for this event, the LUD should be varied off before attempting further operations.

  The event-related data for this event consists of:

Bytes 0-1	Hardware error code
Bytes 2-9	Error log time stamp or 0's
Bytes 10-11	Operational unit number (hex 0018 or hex 0058)
Bytes 12-13	Optional data (not used)

  This event is signaled only for those failures that have hardware error codes that correspond to the error summary codes hex F075 and hex E010 000E.

- Operator intervention required event (hex 000B 07 01)

  The operator intervention required event is signaled when the device requires the operator to take some action, but no error has occurred.

  The 14-byte variable data returned with this event is formatted as follows:

Bytes 0-1	Status – code that was included in the error log message if there was a corresponding error log message; otherwise, the status bytes contain a 0.
Bytes 2-9	Time stamp – contains the time stamp of the corresponding error log message; otherwise, the time stamp is 0.
Bytes 10-11	Operational unit number – hex 0018 or hex 0058
Bytes 12-13	Optional data – format is as follows:

Condition	Indication	Description
Byte 12 bit (0)	Ready light off	SCS command was received, and the printer was not ready. Printer is stopped and is not ready as a result of a normal stop condition, and no errors exist.

  Recovery Procedures

  Press the Start key.

  The remaining bits of bytes 12 and 13 are 0.

- Request I/O complete event (hex 000B 09 01)

  This event is signaled if the user requests it. See *Request I/O (REQIO)* in Chapter 17 for details.

- Request I/O response queue destroyed event (hex 000B 0A 01)

  This event is signaled if the user truncates or destroys the request I/O response queue while the machine is processing Request I/O instructions.

Exceptions

The following table gives the cases where the source/sink resource not available exceptions (hex 3404) are signaled when a Modify LUD instruction for the machine console LUD is executed.

Command	Defect Code (hex)	Device-Specific Return Code (hex)	Meaning
Vary On	2302	0001	I/O device failure has occurred. LUD failure event has been signaled.
Power On	2306	0601	I/O device power failure has occurred. LUD failure event has been signaled.
Power Off	2307	0601	I/O device power failure has occurred. LUD failure event has been signaled.
Suspend Session	2312	1203	Suspend session rejected because an operator intervention condition exists.
Quiesce Session	2313	1302	Quiesce failure because a terminating error condition exists.
		1303	Quiesce session rejected because an operator intervention condition exists

DISKETTE MAGAZINE DRIVE PROGRAMMING CONSIDERATIONS

The diskette magazine drive may be used for either load/dump operations or data interchange operations.

The logical unit description (LUD) for the diskette magazine drive is created and initialized by the Create Logical Unit Description instruction. There must be enough space reserved in the LUD for the device-specific parameters needed by the diskette magazine drive. The device-specific parameters are used during session processing and are initially set to 0's by the Create Logical Unit Description instruction.

Before a Request I/O instruction can be accepted by the diskette magazine drive, certain steps must occur.

1. The LUD must be created through the use of the Create LUD instruction.

2. The diskette magazine drive must be varied on through the use of the Modify LUD instruction.

3. The LUD must be made active through the use of the Modify LUD instruction.

An active session allows creation of a file(s) or access to a file(s) on the diskette. The Request I/O instruction causes blocks of data to be transferred to the diskette magazine drive or from the diskette magazine drive if the device is in an active session for data interchange. When the device is in an active session for load/dump operations, objects are transferred to/from the diskette magazine drive.

DISKETTE MAGAZINE DRIVE CREATE LOGICAL UNIT DESCRIPTION (CRTLUD) TEMPLATE

The following fields of the logical unit description template must be initialized as indicated for the diskette magazine drive:

Field	Entry
LUD type	Char 00
Device type	Char 72MD
Model number	Char 1001
LUD operational number	Hex 0012
Power control	Hex 0000
Session definition data	Bin 0
Load/dump device	Hex 01
Operating mode	Hex 00 (data interchange mode) Hex 01 (load mode) Hex 02 (dump mode)
Specific characteristics length	Hex 0000
Retry value length	Hex 0006 (one retry value)
Retry values (see *Retry Values* later in this chapter)	Char(6)
• Error type	Char(2)
• Error retry value	Bin(2)
• Reserved (binary 0)	Bin(2)
Error threshold length	Hex 0008 (one error threshold)
Error threshold values (see *Error Threshold Values* later in this chapter)	Char(8)
• Error type	Char(2)
• Threshold value	Bin(2)
• Reserved (binary 0)	Char(4)
Device-specific contents length	Hex 0210 (528)
Modifiable length	Hex 0105 (261)
Device-specific contents (see *Device-Specific Contents* later in this chapter)	Char(528)

Retry Values

Error Type	Error Description	Error Retry Value
Hex 0001	Data or ID CRC error on Read Data operation	Value (40-80) Suggested nominal value = 40

Error Threshold Values

Error Type	Error Description	Threshold Value
Hex 0001	Data or ID CRC error on Read Data operation	Value (1-100) Suggested nominal value = 50

Device-Specific Contents: Char(528)

Bytes 0-4 Active session flags Char(5)

- Byte 0 Char(1)
  - Bit 0 Diskette encoding 0 = EBCDIC 1 = ASCII
  - Bits 1-7 Reserved (binary 0)
- Bytes 1-4 Reserved (binary 0) Char(4)

Bytes 5-132 Current volume label Char(128)

- Bytes 5-75 Unused Char(71)
- Byte 76 Type Char(1)
  Hex 40 = 1-sided, FM mode
  Char 2 = 2-sided, FM mode
  Char M = 2-sided, MFM mode
- Bytes 77-79 Unused Char(3)
- Byte 80 Sector size Char(1)
  Hex 40 = 128-byte sectors
  Char 1 = 256-byte sectors
  Char 2 = 512-byte sectors
  Char 3 = 1024-byte sectors
- Bytes 81-132 Unused Char(52)

Bytes 133-260 File header Char(128)

- Bytes 133-166 Unused Char(34)
- Bytes 167-171 End of extent (CCHRR) Char(5)
- Bytes 172-176 Unused Char(5)
- Byte 177 Multivolume indicator Char(1)
  Hex 40 = Data set complete on volume
  Char C = Data set continued on another volume
  Char L = Last volume of the data set
- Bytes 178-179 Volume sequence number Char(2)
  Hex 4040 = Noncontinued
  Char 01-Char 99 = Continued
- Bytes 180-206 Unused Char(27)
- Bytes 107-211 End of data (CCHRR) Char(5)
- Bytes 212-260 Unused Char(49)

Bytes 261-266 Status flags Char(6)

- Byte 261 Reserved (binary 0) Char(1)
- Byte 262 Current slot number (binary 0) Char(1)
- Bytes 263-265 Bad cylinders (binary 0) Char(3)
- Byte 266 Reserved (binary 0) Char(1)

Bytes 267-394	L/D volume label (binary 0)	Char(128)
Bytes 395-522	L/D header label (binary 0)	Char(128)
Bytes 523-527	L/D interrupt location (binary 0)	Char(5)
• Byte 523	Interrupted slot number	Char(1)
• Byte 524	Interrupted cylinder	Char(1)
• Byte 525	Interrupted head	Char(1)
• Byte 526	Interrupted sector	Char(1)
• Byte 527	Interrupted sector size	Char(1)

The current volume label and file header fields are ignored during a Create LUD instruction and would generally contain binary 0's or blanks. The fields other than the unused fields could be specified, if known, but they are usually set after a Request I/O instruction for a read VTOC operation has been executed. The normal startup procedure is as follows:

```
MODLUD  – Vary on
MODLUD  – Activate session
REQIO   – Increment diskette address
          (loads diskette)
MODLUD  – Set encoding bit (if needed)
REQIO   – Read VTOC
MODLUD  – Modify the LUD device specific
          contents (sets current volume label and
          file header fields from read VTOC data)
REQIO   – Seek to address
REQIO   – Read/write
  .
  .
  .
```

The current volume label and file header fields are formatted as they are on the diskette so that they can be set directly from the information read from the VTOC. The values in the unused fields are ignored. The end-of-extent and end-of-data fields have a 5-character CCHRR format in which:

CC = Cylinder Char 01-Char 74 for end of extent
Char 011-Char 75 for end of data

H = Head Char 0 or Char 1

RR = Sector Char 01 (see *Seek to Address Command* later in this chapter)

If a MODLUD (activate) instruction is issued, the type, sector size, and the end of extent/end of data sector number in the LUD device-specific contents area are checked for valid values. If invalid values area specified, the following defaults are used internally for I/O processing.

Type:	1-sided, FM mode
Sector size:	128-byte sectors
End of extent/end of data sector number:	If 0, default to 1

These defaults are used until a MODLUD instruction is issued to modify the device specific area of the LUD. If any of these values are found invalid at that time, the MODLUD instruction is rejected and the template value invalid exception is signaled.

The current slot number and bad cylinders fields are feedback areas for the Materialize LUD instruction and should be binary 0 for a Create LUD instruction. The current slot number field is set whenever the Increment Diskette Address command in the Request I/O instruction is used; this field can have values of hex 01-17. The bad cylinders field is set only when a diskette is formatted by the Request I/O instruction and is initialized to hex FFFFFF. For bad cylinders, each byte (for up to three bad cylinders) can have values of hex 00-4C.

The L/D volume label, L/D header label, and L/D interrupt location are used only with L/D operations. See Chapter 25 for additional information about L/D operations. The values are hex values and have the following ranges:

Interrupted slot number:	Hex 01-Hex 17
Interrupted cylinder number:	Hex 01-Hex 4C
Interrupted head:	Hex 00 or Hex 01
Interrupted sector:	Hex 01-Hex 08
Interrupted sector size:	Hex 03 for a 1024-byte sector

Note: The only valid sector size for L/D operations is 1024.

DISKETTE MAGAZINE DRIVE MODIFY LOGICAL UNIT DESCRIPTION (MODLUD) INSTRUCTION

The following is a list of functions the diskette magazine drive supports through the Modify Logical Unit Description (MODLUD) instruction:

- Vary on
- Vary off
- Activate
- De-activate
- Suspend
- Quiesce
- Reset
- Resume (activate after suspend, quiesce, or reset)
- Modify device-specific contents
- Modify operating mode (load, dump, data interchange)
- Modify retry values
- Modify error threshold values

See Chapter 17 for the meaning and use of each function.

The following are special considerations taken by the machine for the MODLUD functions listed previously:

- The vary on and de-activate functions eject a diskette, if one is loaded, and cause the magazine to go to the home position.

- The reset function causes the machine to cease processing as soon as possible. Continuing the session after a Reset function yields unpredictable results.

- The first 261 (hex 105) bytes of device-specific contents area of the LUD (the active session flags, current volume label, and file header fields) may be altered at any time through the use of the MODLUD instruction. Note that any changes to the device-specific area are reflected in the I/O commands issued to the device. Generally, the area is altered to set the volume label and file header before file processing begins, not during the processing of a file. If the user does alter the area during file processing, the results are unpredictable and may cause I/O errors. The status flags field, load/dump volume label, header label, and interrupt location fields are feedback areas meaningful only to the MATLUD instruction and are not modifiable. The other modifiable fields in the LUD template are described under the *Modify Logical Unit Description (MODLUD)* instruction in Chapter 17.

For normal data interchange, the load/dump definition data in the LUD should be set as follows:

- Load/dump device (set on the Create LUD instruction) — Hex 01
- Operating mode — Hex 00
- Load/dump pending — Bin 0
- Corresponding primary address — Bin 0
- Load/dump exchange status — Bin 0

For load or dump data interchange, the operating mode field can be set to hex 01 (load mode primary device) or to hex 02 (dump mode primary device). All other fields should be set the same as in normal mode.

DISKETTE MAGAZINE DRIVE REQUEST I/O (REQIO) INSTRUCTION

The SSR (see *Create Logical Unit Description (CRTLUD)* in Chapter 17) for a Request I/O instruction to the diskette magazine drive contains the following values:

Field	Value
Source/sink object	Pointer to the diskette magazine drive LUD
Response queue	Pointer to the response queue
Source/sink data area	Space pointer
Optional pointer	Reserved (binary 0)
Request priority	See *Request I/O (REQIO)* in Chapter 17
Request identification	See *Request I/O (REQIO)* in Chapter 17
Function field	Hex 80
Control field	N or C
Key length	Bin(2)
Key offset	Bin(2)
RD count	Bin(2)
RD offset	Bin(2)

The diskette magazine drive does not check the label information except on a VTOC operation. On a write VTOC operation, the first record in the SSD (volume label) must begin with VOL1 and any header labels to be written must begin with HDR1. On a read VTOC operation, the first record read from the diskette must begin with VOL1 and the only subsequent records placed in the SSD are those that begin with HDR1.

If the diskette encoding bit in the device-specific area of the LUD specifies ASCII and it is a read operation, the machine translates the ASCII data that is read into EBCDIC. If ASCII is specified and it is a write operation, the machine takes the EBCDIC data from the SSD and translates it to ASCII before the write operation takes place. If invalid EBCDIC characters are found during an EBCDIC to ASCII conversion, an error is returned. Bytes 3-6 of the write RD contain an offset specifying the byte in the SSD that is invalid. If EBCDIC is specified, no translation takes place.

Note: If an error recovery action is to reissue the REQIO instruction in error, the request descriptors and the SSR should not be altered; otherwise, unpredictable results could occur.

Request Descriptor

A Request I/O instruction (other than continue) to the diskette magazine drive I/O manager must contain one or more RDs. Only one write or read RD is allowed for each Request I/O instruction. A Request I/O instruction with one read RD and one write RD results in an error. Any number of Seek, Address, Format, Halt, or Increment commands may exist in a Request I/O instruction, with or without a single read or write RD.

The format of the RD is:

Byte 0	Command Hex 01 – Write Hex 02 – Read Hex 04 – Seek to Address Hex 05 – Format Diskette Hex 0C – Increment Diskette Address Hex 18 – Address Hex 1C – Halt	Char(1)
Bytes 1-6	Command modifier	Char(6)
• Byte 1 – Bit 0	Control Command control[1] 0 = Perform command 1 = Ignore command	Char(1)
– Bits 1-7	Command specific (see *Command* in this chapter)	
• Bytes 2-6	Command specific (see *Command* in this chapter)	Char(5)
Byte 7	Reserved (binary 0)	Char(1)
Bytes 8-9	Segment count	Bin(2)
Bytes 10-15	Reserved (binary 0)	Char(6)

[1]Fields modified by the execution of a Request I/O instruction

The command field indicates the type of operation to be performed. The specific operations are described with the individual commands.

The control field indicates whether the RD is to be ignored and generally is a binary 0 when the instruction is issued. This field is set by the machine to a binary 1 when the RD is successfully completed. This information is useful when the user restarts an operation after an error. The Request I/O instruction can be reissued; all RDs that were completed are ignored, and the RD that failed is retried. After normal completion, the control field must be reset before the SSR can be reused.

The segment count field identifies the number of consecutive sectors to be transferred by the operation. This field must be 0 for a read VTOC function or a read VTOC IPL function.

Each segment corresponds to the sector size on the diskette and is accessed sequentially. All RD commands use the parameters specified in the device-specific area of the LUD. All command modifier bytes must be reinitialized unless the RD command reissued is a partially complete Request I/O instruction.

Source Sink Data (SSD) Conventions

Sectors are placed in the SSD consecutively with no control bytes between sectors. A reissued Request I/O instruction begins at the position indicated by the segment count, sectors transferred count, sectors skipped count, and the sector size in the LUD. The SSD area for any Read or Write command must be page aligned.

Commands

The following are the commands supported by the Request I/O instruction.

Read Command: The format of the RD for a Read command is as follows:

Byte 0	Command (Hex 02)	Char(1)
Bytes 1-6	Command modifier	Char(6)
• Byte 1	Control	Char(1)
– Bit 0	Command control[1]	
– Bits 1-2	Read control Bin 00 = Normal read Bin 10 = Read VTOC Bin 01 = Read VTOC IPL	
– Bits 3-7	Reserved (binary 0)	
• Byte 2	Reserved (binary 0)	Char(1)
• Bytes 3-4	Sectors transferred count[1]	Bin(2)
• Bytes 5-6	Sectors skipped count[1]	Bin(2)
Byte 7	Reserved (binary 0)	Char(1)
Bytes 8-9	Segment count	Bin(2)
Bytes 10-15	Reserved (binary 0)	Char(6)

[1]Fields modified by the execution of a Request I/O instruction

The three types of Read commands are normal, VTOC, and VTOC IPL. The normal Read command is used to read one or more sectors from any cylinder other than 0. The device must have been previously positioned to the first sector to be read. The first sector read is placed in the SSD at the offset indicated by the sectors transferred count multiplied by the sector size in the LUD.

The number of sectors to be read must be greater than 0, or an error is signaled. The number of sectors to be read is the segment count minus the sectors transferred count and the sectors skipped count. Normally, when an RD is first issued, the sectors transferred count and sectors skipped count fields are 0. The sectors transferred count in the RD is incremented for every sector transferred to the SSD. The sectors skipped count in the RD is incremented every time an error is signaled due to the occurrence of a sector marked as deleted or relocated. When the RD is marked as successfully completed, the sum of the sectors transferred count and the sectors skipped count equals the segment count, and the SSD contains the number of sectors indicated by the sectors read count. When a sector marked as deleted or relocated is encountered the count is increased, an error FBR is returned, nothing from that sector is placed in the SSD, and the device is positioned after the sector in question. The user only has to issue a Request I/O (continue) instruction to clear the error and reissue the Request I/O to continue reading subsequent sectors. A bad sector error works exactly the same way except that the sectors skipped count is not incremented. This allows the user to either quit reading or to note the sector that is missed and to continue reading the rest of the file.

Each sector on the diskette has a sequential sequence number preceding it. The sequence numbers are checked by the machine when reading a diskette. Some diskettes may not have sequential sector sequence numbers. When a sector sequence number error is encountered, the device is left positioned at the sector in error and an error FBR is returned. The user has to issue only a Request I/O (continue) and reissue the Request I/O instruction to read the sector (since it is now the starting sector of a new read command). If the following sectors sequence number is not one larger than the previously read one, another sector sequence error is signaled.

If an end of track is encountered while reading, the read/write head is automatically moved to the next sequential sector, to the other side of a two-sided diskette, or to the next good cylinder.

End-of-volume and end-of-data conditions are checked at all times, by using the volume and file header information stored in the LUD. On end-of-volume, the user, after issuing a Request I/O (continue) instruction, causes a new diskette to be loaded, reads the VTOC, modifies the LUD with the volume and file data, seeks to the correct address, and then reissues the Request I/O instruction to continue reading.

There are special requirements and limitations for a normal read command when sectors are read from cylinder 0 because cylinder 0 is unique. The segment count field must be set to 1 which allows only one sector to be read at a time.

The following fields in the device-specific area of the LUD must be set to reflect the characteristics of the particular sector to be read from cylinder 0 rather than the characteristics of the data sectors on cylinders 1 through 74:

- Diskette encoding – EBCDIC or ASCII
- Type – 1-sided FM mode, 2-sided FM mode, or MFM mode
- Sector size – 128 or 256

The end of extent and the end of data must be set beyond the sector to be read.

Sectors 1 through 3 of cylinder 0 are always in EBCDIC. The remaining sectors may be either EBCDIC or ASCII.

Sectors on the front side of cylinder 0 are always 128 bytes (FM mode). If the diskette is two-sided, the sectors on the back of cylinder 0 may be 128 bytes (FM mode) or 256 bytes (MFM mode).

The Read VTOC command is used to read cylinder 0 for the volume label (those having VOL1 in the first 4 characters) and all undeleted valid header records (those having HDR1 as the first 4 characters). Only the volume label and the undeleted valid header records are placed in the SSD. The SSD must be large enough to hold the maximum number of records (40 for a one-sided diskette, 46 or 72 for a two-sided diskette). Note that VTOC records are always 128 bytes long. The sectors transferred count is set to indicate the number of records including the VOL that are placed in the SSD. The maximum number of records possible on the front side of a diskette (including the VOL) is 20. On the back side of a diskette, the maximum number of headers possible in FM mode is 26 or 52 if in MFM mode.

The segment count and sectors skipped count are not used with a Read VTOC command and must be binary 0.

When a Read VTOC command is executed, the machine does an implicit seek to the correct position on the disk. Following the Read VTOC command, a seek must be done to position for a normal Read or Write command.

Normal completion of a Read VTOC command results in an error FBR and requires a Request I/O (continue) instruction to continue operations. This is done to facilitate end-of-volume processing. For example, the end-of-volume sequence of events could be as follows:

1. End-of-volume error on a Request I/O instruction.
2. Issue a Request I/O instruction to load the new diskette. The new diskette should have a lower key value to bypass Request I/O instructions already on the queue.
3. Issue a Read VTOC command.
4. Issue a Request I/O (continue) instruction to release the above Request I/O instructions.
5. Read VTOC command complete error. Analyze header information.
6. Issue a Modify LUD instruction to modify VOL and HDR fields.
7. Issue a Request I/O instruction to seek to the first sector of the new file.
8. Reissue a Request I/O instruction to read VTOC.
9. Issue a Request I/O (continue) instruction to release the above Request I/O instructions.

Encountering a bad sector causes the read to stop. Sectors previously read are in the SSD. Subsequent sectors are inaccessible.

The Read VTOC IPL command is used at IPL time and reads the first 3 sectors of the VTOC (384 bytes) from cylinder 0.

At normal completion of a Read VTOC IPL command, an error is signaled to allow the user to analyze the data. To recover from this condition, the user need only issue a Request I/O (continue) to continue processing. When this command is to be executed, the machine does an implicit seek to the correct position on the disk and the former position is lost.

If the data read is to be converted from ASCII to EBCDIC, any characters read that cannot be converted to a corresponding EBCDIC character are replaced by an asterisk (hex 5C) in the user buffer. This is done without any error condition being returned to the user. The exception to this is a read VTOC operation. If the first 4 bytes read from sector seven are not VOL1 in ASCII format, nothing is transferred to the SSD area. This is also true if EBCDIC mode is specified and the first 4 bytes are not VOL1 in EBCDIC format. Conversion does not occur on a Read VTOC IPL RD command because the data is always written and read in EBCDIC format.

Encountering a bad sector causes the operation to stop, and subsequent sectors are inaccessible. The sectors transferred count, sectors skipped count, and segment count fields are not used with a Read VTOC IPL command and must be binary 0.

Write Command: The format of the RD for a Write command is as follows:

Byte 0	Command (Hex 01)	Char(1)
Bytes 1-6	Command modifier	Char(6)
• Byte 1	Control	Char(1)
– Bit 0	Command control[1]	
– Bits 1-2	Write control Bin 00 = Normal write Bin 10 = Read VTOC Bin 01 = Read VTOC IPL	
– Bits 3-7	Reserved (binary 0)	
• Byte 2	Reserved (binary 0)	Char(1)
• Bytes 3-4	Sectors transferred count[1]	Bin(2)
• Bytes 5-6	Reserved (binary 0)	Bin(2)
Byte 7	Reserved (binary 0)	Char(1)
Bytes 8-9	Segment count	Bin(2)
Bytes 10-15	Reserved (binary 0)	Char(6)

[1]Fields modified by the execution of a Request I/O instruction

A normal Write command is used to write one or more sectors to any cylinder other than cylinder 0. The device must have been previously positioned to the first sector to be written.

The number of sectors to be written must be greater than 0 or an error is signaled. The number of sectors to be written is the segment count minus the sectors transferred count. Normally, when an RD is first issued, the sectors transferred count field is 0. The sectors transferred count in the RD is incremented for every sector written. The device does not write any deleted or relocated sectors, but does write over deleted or relocated sectors. When the RD is marked as successfully completed, the sectors transferred count equals the segment count.

If a sector media write error FBR is returned, the read/write head is positioned after the sector that could not be written.

If an end of track is encountered while writing, the read/write head is automatically moved to the next sequential sector, to the other side of a two-sided diskette, or to the next good cylinder. End-of-volume is indicated when the sector written corresponds to the end of extent field in the file header record located in the device-specific area of the LUD. If the end-of-extent value is wrong (greater than the diskette capacity), an I/O error occurs. In an end-of-volume condition, the user must handle the situation and reissue the Request I/O instruction being processed when the error occurred.

The Write VTOC command is used to write all of cylinder 0 (starting with sector 7) with a VOL label, header labels, and deleted sectors.

If the first 128-byte record in the SSD does not begin with VOL1, an error is returned. All of the subsequent 128-byte records in the SSD starting with HDR1 are also written. Records not starting with HDR1 are ignored in the SSD. The segment count indicates the number of 128-byte records in the SSD and must be greater than 0. The maximum number of records possible on the front side of a diskette (including VOL) is 20. On the back side of the diskette, the maximum number of records possible in FM mode is 26, and the maximum number of records possible in MFM mode is 52.

When the SSD is exhausted, the rest of cylinder 0 (both sides if two-sided diskette) is written with deleted records (records starting with DDR1). If the diskette encoding bit in the LUD specifies ASCII, the VOL header and deleted records are written in ASCII format. When a Write VTOC is executed, the machine does an implicit seek to the VTOC area. Following the Write VTOC command, a seek must be done to position for a normal read/write since the former position is not retained. It is not possible to write more than cylinder 0 with this command. The number of headers written (including the VOL and excluding deleted headers) is returned in the sectors transferred count field. Normal completion of the Write VTOC command results in an error FBR and requires a Request I/O (continue) instruction to continue.

The Write VTOC IPL command is used to write the first three sectors of cylinder 0 with 384 bytes of data from the SSD. At the normal completion of a Write VTOC IPL command, an error FBR is returned and the user need only issue a Request I/O (continue) to continue processing.

When a Write VTOC IPL command is executed, the machine does an implicit seek to the VTOC area, and the former position of the read/write head is not retained. It is not possible to write more or less than 384 bytes with this command. The sectors transferred count and segment count fields are unused and must be binary 0. An error in writing a sector causes the operation to stop and a device error to be returned in the FBR.

If the data is to be converted from EBCDIC to ASCII format, the data is copied from the SSD area to an internal buffer and then converted. If an EBCDIC character is encountered that cannot be converted to ASCII, the offset to the invalid data field in the RD (bytes 3-6) is set to the offset of the byte in error in the SSD area. For example, if the first byte in the SSD contains an invalid character, the offset field in the RD is set to 0. Conversion is done only on a cylinder boundary. If the data to be written requires more sectors than are available on the cylinder to be written on, any data successfully converted, on a cylinder basis, is written and the position of the read/write heads will increment to the next cylinder(s). If the user recovery action is to correct the character in error (substitute a valid EBCDIC character) and reissue the Request I/O command, the offset to invalid data in the RD must be reset (set to 0). Depending on how much data was successfully converted and written, it may be necessary to issue a Request I/O command to position the read/write heads at the address that they were initially at before reissuing the Write Request I/O command. Conversion will not occur on a Write VTOC IPL RD command because the data is always written in EBCDIC format.

Seek to Address Command: This command allows the user to specify an address, and the machine positions the read head at that address. If the address specified is invalid, an error is indicated in the status field of the feedback record, and the former position of the read/write head is retained.

The format of the RD for the Seek to Address command is as follows:

Byte 0	Command (Hex 04)	Char(1)
Bytes 1-6	Command modifier	Char(6)
• Byte 1	Control	Char(1)
– Bit 0	Command control[1]	
– Bits 1-7	Reserved (binary 0)	
• Byte 2	Cylinder Hex 01-Hex 4A (1-74)	Char(1)
• Byte 3	Head Hex 00-Hex 01	Char(1)
• Byte 4	Sector Hex 01 (dependent on sector size; see below)	Char(1)
• Bytes 5-6	Reserved (binary 0)	Char(2)
Bytes 7-15	Reserved (binary 0)	Char(9)

[1]Fields modified by the execution of a Request I/O instruction

If the type field in the device-specific area of the LUD specifies frequency modulation (hex 40 or char 2), the valid sector sizes and ranges are:

Size	Maximum Number Sectors/Track
128	26
256	15
512	8

If the type field in the device-specific area of the LUD specifies multiple frequency modulation (char M), the valid sizes and ranges are:

Size	Maximum Number Sectors/Track
256	26
512	15
1024	8

Note: These sizes and ranges apply only to cylinders other than cylinder 0.

Format Diskette Command: The machine reformats the diskette upon receiving this command.

When diskette formatting is requested, the entire diskette is formatted by writing valid identifier fields throughout all the sectors on the diskette. If any sector is bad, the cylinder is marked defective. Formatting is done by writing cylinder X head 0, then cylinder X head 1. For a 33FD diskette, formatting head 1 is ignored.

The cylinder initialization pass follows the cylinder formatting. The initialization pass is done by writing worst case data sectors and verifying the write was successful. This sequence is repeated cylinder by cylinder for the entire diskette. Upon completion of these Write commands, the volume header, along with the error recovery map, is written in the VTOC area. The volume header is taken from the device-specific area. The error recovery map is constructed. If a sector is found to be defective during the write operation, the cylinder is marked bad, and the next cylinder is used. If more than two cylinders are bad or if cylinder 0 is bad or if none of the hex FF control records written on a bad track can be read, the diskette is bad. For each defective cylinder found, the device-specific area of the logical unit description is updated with the bad cylinder number.

The format of the RD for the Format Diskette command is as follows:

Byte 0	Command (hex 05)	Char(1)
Bytes 1-6	Command modifier	Char(6)
• Byte 1	Control	Char(1)
– Bit 0	Command control[1]	
– Bits 1-7	Reserved (binary 0)	
• Byte 2	Slot number Hex 01-Hex 17 (1-23)	Char(1)
• Bytes 3-6	Reserved (binary 0)	Char(4)
Bytes 7-15	Reserved (binary 0)	Char(9)

[1]Fields modified by the execution of a Request I/O instruction

Increment Diskette Address Command: This command causes the device to replace the diskette currently in the machine with one specified by the user. The valid request range is 1-3 for manual slots, 4-13 for magazine 1 slots 1-10, and 14-23 for magazine 2 slots 1-10. Any other request results in an error. If 0 is specified for the slot, the magazine goes to the home position. This command always results in a seek to cylinder 0. If the current slot number is the one requested, no magazine movement occurs.

The format of the RD for the Increment Diskette Address command is as follows:

Byte 0	Command (hex 0C)	Char(1)
Bytes 1-6	Command modifier	Char(6)
• Byte 1	Control	Char(1)
– Bit 0	Command control[1]	
– Bits 1-7	Reserved (binary 0)	
• Byte 2	Slot number Hex 00-Hex 17 (0-23)	Char(1)
• Bytes 3-6	Reserved (binary 0)	Char(4)
Bytes 7-15	Reserved (binary 0)	Char(9)

[1]Fields modified by the execution of a Request I/O instruction

Address Command: This command indicates to the user the current position of the read/write head on the diskette. This information is returned in the request descriptor command modifier byte. This address is the internal position of the diskette read/write head; therefore, there is no interface to the I/O devices.

The possible values returned in the slot number field are 0-23 (hex 00-hex 17) and represent the slot number of the diskette loaded. If no diskette is loaded, the drive is in the home position (slot number 0) and the cylinder, head, sector, and sector size fields will be set to 0.

The other fields can have the following values:

Cylinder:	0-74 (hex 00-hex 4A)
Head:	0-1 (hex 00-hex 01)
Sector:	1 (dependent on sector size; see the *Seek to Address Command*)
Sector size:	Hex 00 = 128-byte sectors Hex 01 = 256-byte sectors Hex 02 = 512-byte sectors Hex 03 = 1024-byte sectors

For example, if the user started reading on diskette 13 (last diskette in magazine 1), written with 128-byte sectors, on cylinder 5, the primary side, starting with sector 1 for 8 sectors and then this command was issued, the values returned in bytes 2-6 would be 0D05000900. If the user issued a Seek to Address command and followed by an Address command, the address returned would be the same address as that for the Seek command.

The format of the RD for the Address command is as follows:

Byte 0	Command (hex 18)	Char(1)
Bytes 1-6	Command modifier	Char(6)
• Byte 1	Control	Char(1)
– Bit 0	Command control[1]	
– Bits 1-7	Reserved (binary 0)	
• Byte 2	Slot number[1]	Char(1)
• Byte 3	Cylinder number[1]	Char(1)
• Byte 4	Head[1]	Char(1)
• Byte 5	Sector number[1]	Char(1)
• Byte 6	Sector size[1]	Char(1)
Bytes 7-15	Reserved (binary 0)	Char(9)

[1]Fields modified by the execution of a Request I/O instruction

Halt Command: This command halts further Request I/O instructions from being processed. Processing is terminated, and a feedback record is returned. Issuing a Request I/O (continue) instruction restarts the process.

The format of the RD for the Halt command is as follows:

Byte 0	Command (hex 1C)	Char(1)
Bytes 1-6	Command modifier	Char(6)
• Byte 1	Control	Char(1)
– Bit 0	Command control[1]	
– Bits 1-7	Reserved (binary 0)	
• Bytes 2-6	Reserved (binary 0)	Char(5)
Bytes 7-15	Reserved (binary 0)	Char(8)

[1]Fields modified by the execution of a Request I/O instruction

Feedback Record and Error Recovery Procedures

The format of the feedback record is as follows:

Bytes 0-15	Source/sink request address	Space pointer
Bytes 16-17	Request identification	Bin(2)
Bytes 18-19	Error summary	Bin(2)
Bytes 20-21	RD number	Bin(2)
Bytes 22-23	RIU segment count	Bin(2)
Bytes 24-63	Device-dependent area	Char(40)
• Bytes 24-25	Device-dependent error code	Char(2)
• Bytes 26-27	Hardware error code	Char(2)
• Bytes 28-35	Time stamp	Char(8)
• Bytes 35-36	Operating unit number	Char(2)
• Bytes 38-63	Reserved (binary 0)	Char(26)

Descriptions of the request address and request ID fields are given under *Request I/O (REQIO)* in Chapter 17.

The error summary field defines the status of the Request I/O instruction defined in the Request I/O instruction description. The specific values possible for the diskette magazine drive are listed in Figure 23-6.

The RD number is the index of the last RD processed or the RD in error if an error is indicated.

The RIU segment count field is not used by the diskette magazine drive and is set to binary 0.

The device-dependent area is all binary 0's unless the presence of device-dependent data is indicated by the error summary value. If device-dependent data is present, the fields have the following definitions.

The device-dependent error code is a further categorization of the hardware error codes shown in Figure 23-6.

The hardware error code is logged in the hardware error log and indicates the specific hardware error encountered. The possible values are shown in Figure 23-6.

The time stamp and operating unit number are the same values present in the hardware error log entry and are used to correlate the FBR and the error log entry for maintenance purposes.

Figure 23-6 lists the error summary, device-dependent error codes, the hardware error log codes that caused the error, and the recommended recovery action. If the error is the result of a load/dump operation, only the last byte of the error summary field is shown. This is concatenated with the load/dump code, resulting in the error summary returned with the load/dump feedback record and itemized in the load/dump description.

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
0000	N/A	N/A	No error condition	N/A
0008	N/A	N/A	REQIO (continue) instruction complete	N/A
4089	N/A	N/A	REQIO (continue) instruction rejected because of a normal REQIO instruction currently being processed	N/A
C009	N/A	N/A	Partially processed request terminated because of reset session	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (de-activate) instruction must be issued to destroy the session.
C00A	N/A	N/A	Unprocessed request because of reset session	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (de-activate) instruction must be issued to destroy the session.
C010	N/A	N/A	An REQIO instruction rejected because the device operational unit task failed	The MODLUD (reset, de-activate, vary off, vary on, and activate) functions must be issued to restart processing.
C016	N/A	N/A	End of file	An REQIO (continue) instruction is needed to start processing.
C017	N/A	N/A	End of volume	An REQIO (continue) instruction is needed to start processing.
C042	N/A	N/A	Data not valid (EBCDIC to ASCII translation error)	Change the encode bit in the device-specific area, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
C043	N/A	N/A	Invalid buffer alignment	Allocate the SSD to be page aligned, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
C044	N/A	N/A	SSD area is too small	Make a smaller request or make the SSD larger, reissue the REQIO instruction, and issue an REQIO (continue) instruction.

Figure 23-6 (Part 1 of 6). Diskette Magazine Drive Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
C084	N/A	N/A	Invalid SSD pointer	Correct SSD pointer with a valid pointer, reissue the REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C085	N/A	N/A	Invalid function field	Correct the SSR, reissue the REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C088	N/A	N/A	Invalid RD sequence	User may have only one Read or Write command per REQIO instruction. Correct and reissue the REQIO instruction and issue an REQIO (continue) instruction to restart processing.
D0CF	N/A	N/A	Halt RD encountered in RD sequence	Issue an REQIO (continue) instruction to restart processing.
D0EF	N/A	N/A	VTOC command has been executed	Issue an REQIO (continue) instruction to restart.
F0DF	N/A	0101	Unrecoverable error because of a channel error	User should reset and terminate the session. The job may be restarted. If the error persists, call the service representative.
		0504	Channel error on read sense during channel error recovery procedure	
E010	0001	0107	One-sided media in drive	Request is invalid for a one-sided diskette. Check request to see whether logic is correct. An REQIO (continue) instruction is needed to restart processing.
E010	0003	N/A	Device cover open	Close cover, reissue the REQIO instruction, and issue an REQIO (continue) instruction to restart processing.

Figure 23-6 (Part 2 of 6). Diskette Magazine Drive Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	0007	N/A	Diskette defective	Replace diskette. Issue an REQIO (continue) instruction to restart.
		0403	Read data CRC error	
		0404	Write/verify data CRC error	
		0405	Read ID CRC error	
		0406	Write/verify ID CRC error	
		0407	Not oriented 1 error	
		0408	An incorrect address mark sequence (not oriented 2)	
E010	0009		I/O error media	Issue an REQIO (continue) instruction to restart. The machine is positioned at the first sector beyond the failing sector.
		0403	Read data CRC error	
		0404	Write/verify data CRC error	
		0405	Read ID CRC error	
		0406	Write/verify ID CRC error	
		0408	An incorrect address mark sequence (not oriented 2)	
E010	000A	0407	Nonsequential sector sequence detected (not oriented 1)	Operation will continue if the REQIO instruction in error is reissued and an REQIO (continue) instruction is issued to restart processing. Note that this sequence may have to be repeated for each sector specified to be processed.
E010	000B	0303	Failed to pick a diskette (autoload motion check)	A diskette was not at the slot number requested, or the hardware is malfunctioning. If a diskette is located at the slot requested, call the service representative.

Figure 23-6 (Part 3 of 6). Diskette Magazine Drive Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	000F	0108 or N/A	CTRL address mark with sequence sector relocation was found, or a deleted sector was found. Note that the error log code is not applicable if a deleted sector is found because an error log entry is not made for this condition.	The machine will be positioned at the first sector beyond the failing sector. Reissue the REQIO instruction in error and issue an REQIO (continue) instruction.
E010	0010		I/O hardware error	Call the service representative. The current position of the diskette is indeterminate.
		0100	Operation program error	
		0101	Operational unit task failure	
		0103	Overrun	
		0104	Disconnect	
		0105	Parity error 1	
		0106	Parity error 2	
		0107	Command reject	
		0108	Control address mark	
		0201	Wrap error	
		0202	Autoload parity error	
		0203	Invalid autoload function code	
		0204	Function operation block time-out during an Autoload command	
		0205	Erase current error	
		0206	Autoload command reject modifier D (write/erase error)	
		0207	Write gate error	
		0208	Function operation block time-out of non-autoload command	
		0301	Autoload motion check	
		0302	Autoload motion check	
		0303	Autoload motion check	
		0304	Autoload motion check	
		0306	Autoload command reject (not oriented)	
		0307	Autoload command reject (out of sequence)	

Figure 23-6 (Part 4 of 6). Diskette Magazine Drive Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	0010	0405	ID CRC error on read data	
		0504	Read sense error on channel error	
		0505	Invalid BSTAT or DSTAT data	
		0506	Unexpected BSTAT on Read sense	
		0507	Autoload parity error on read sense	
		0508	Retry stack limit exceeded	
		050C	Failed to read hex FF field on bad cylinder during format operation	
E010	0011	0108	Control address mark with nonsequence sector relocation found	The machine is positioned at the first sector beyond the failing sector. Issue an REQIO (continue) instruction to restart.
E010	0012	0308	Speed check	Check diskette. If it is in backward, insert it correctly, reissue the failing REQIO instruction, and issue an REQIO (continue) instruction to restart. If the diskette is inserted correctly, the hardware is failing. Call the service representative.
E010	0013	0309	Cylinder/head/record mismatch or the RD does not match the diskette.	
		040A	No record found. The diskette mode (FM or MFM) is different than specified by user.	Ensure proper diskette is inserted. Reissue the failing REQIO instruction, and issue an REQIO (continue) instruction to restart.
E010	0014	N/A	Device cover open during error recovery of a read or write operation	Close cover and issue a message to operator to restart the job.
E014	0004	N/A	No diskette loaded	Issue an Increment command, the REQIO instruction that caused the error, and an REQIO (continue) instruction to restart processing.

Figure 23-6 (Part 5 of 6). Diskette Magazine Drive Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E087	0002	N/A	Read/write of VTOC area attempted with control bits not set	Command modifier bits must be on to do a read or write on the VTOC. Correct, reissue the REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
E087	0005	N/A	Seek address invalid	Correct the RD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	0006	N/A	Diskette slot number invalid	Correct the RD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	0008	N/A	Invalid command	Replace the RD command with the proper command, reissue the REQIO instruction, issue an REQIO (continue) instruction to restart processing.
E087	000C	N/A	No sectors specified, or number of sectors specified is equal to RD bytes 3-4	Correct the RD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	000D	N/A	Too many sectors or no sectors to write with VTOC command	Correct the RD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	000E	N/A	VTOC write buffer did not start with VOL1 or VTOC data read did not contain a VOL1 in either EBCDIC or ASCII.	Use one of these recovery actions: 1. Correct the buffer area, reissue the REQIO instruction, and issue an REQIO (continue) instruction. 2. Issue a message stating that the diskette is improperly formatted.
E087	0015	N/A	Reserved field in the RD is not 0	Correct the RD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.

Figure 23-6 (Part 6 of 6). Diskette Magazine Drive Error Summary Values

Diskette Magazine Drive End-of-Volume Handling for Data Interchange

In processing a multivolume file, the following sequence should be followed.

1. The Request I/O instruction with the feedback record marked as end of volume is returned to the user. This is a terminating feedback record.

2. Processing stops as with any terminating condition.

3. The user constructs the Request I/O instruction necessary to address the next diskette. This Request I/O instruction has a request priority less than that of any other Request I/O instruction waiting to be processed.

4. The user issues the Request I/O instruction.

5. The user issues a Request I/O (continue) to start processing.

6. The Request I/O instruction constructed in step 3 is the first one processed because of its lower key value. Because it reads cylinder 0, this Request I/O instruction is special and results in a feedback record marked as terminating. This again stops processing of requests.

7. Repeat steps 3 through 6 until the proper diskette is retrieved.

8. The device-specific area (the VOL and HDR information) is updated if necessary to reflect the attributes of the new diskette.

9. Any Request I/O instructions necessary to seek to a starting point are constructed with a lower key than the one in step 8. These Request I/O instructions are then issued.

10. The Request I/O instruction that caused the end-of-volume condition is reissued with a key lower than any outstanding requests. No modification to this Request I/O instruction is necessary. Processing continues where it left off in step 1.

11. The user should issue a Request I/O (continue) instruction to continue processing.

The Diskette Magazine Drive End-of-Volume Handling for Load/Dump

Load/dump end-of-volume processing is handled like that for data interchange except for steps 8 and 10. (See *End-of-Volume Handling for Data Interchange* earlier in this section.) These steps must have the load/dump bit set on in the function field of the Request I/O instruction involved.

Events

In addition to the events described under *Request I/O (REQIO)* in Chapter 17, the following events are signaled. For a complete description of the following events, see *Chapter 21. Event Specifications*.

- LUD contact event (hex 000B 06 01)

  This event is signaled by the MSCP when vary on processing is completed for this device. Only subtype hex 01 is signaled upon successful contact; subtype hex 02 is never signaled since the MSCP processing is synchronous to the Modify LUD (vary on) instruction and an exception is signaled instead.

- LUD failure (hex 000B 08 01)

  The LUD failure event is signaled when the diskette magazine drive has a problem that requires the service representative to be called.

  The 14-byte variable data returned with this event is formatted as follows:

  Bytes 0-1 Hardware error code – code that was included in the error log message if there was a corresponding error log message; otherwise, the status bytes contain a 0.

  Bytes 2-9 Time stamp – contains the time stamp of the corresponding error log message; otherwise, the time stamp is 0.

  Bytes 10-11 Operational unit number – Hex 0012

  Bytes 12-13 Optional data – reserved (binary 0)

- Request I/O complete event (hex 000B 09 01)

  The Request I/O complete event is signaled when the request has been processed if the user has requested this event to be signaled.

- Response queue destroyed event (hex 000B 0A 01)

  The response queue destroyed event is signaled if the user truncates or destroys the Request I/O instruction response queue while the machine is processing Request I/O instructions.

Exceptions

The following table gives the cases where the source/sink resource not available exceptions (hex 3404) are signaled when a Modify LUD instruction for the diskette magazine drive is executed.

Command	Defect Code (hex)	Device Specific Return Code (hex)	Meaning
Vary On	2302	0001	I/O device failure has occurred. LUD failure event has been signaled.
		0204	A correctable condition such as cover open at the device is preventing device initialization for vary on processing.
Resume Session	2311	0001	I/O device failure has occurred. LUD failure event has been signaled.
Quiesce Session	2313	1302	Quiesce failure because a teminating error condition exists.

3410/3411 PROGRAMMING CONSIDERATIONS

The 3410/3411 tape device may be used for either load/dump operations or data interchange operations.

Each tape drive requires an LUD (logical unit description) (type 10) object to be created for its support. A CD (controller description) (type 00) object must be created for the tape subsystem. These system objects must be created and initialized by using the create logical unit description and the create controller description instructions. There must be enough space reserved in each LUD for the device-specific parameters needed by the machine. The device-specific parameters control the processing performed during an active session.

A Request I/O instruction must be preceded by the activate session function of the Modify Logical Unit Description instruction. The Request I/O instruction causes data to be transferred to or from the tape media.

3410/3411 CREATE CONTROLLER DESCRIPTION (CRTCD) TEMPLATE

The fields of the CD template must be initialized for the 3411 tape controller as follows:

Field Name	Entry
CD type	Char 00
Unit type	Char 3411
Model number	Char 0001, Char 0002, or Char 0003
Physical address (CD OU number)	Hex 0015
Power control	Hex 0100
Station control information	Bin 0
Selected mode data	Bin 0
Activate physical unit information	Bin 0
Dial digits	Bin 0
Specific characteristics length	Hex 0000
XID information length	Hex 0000
Unit-specific contents length	Hex 0000

3410/3411 CREATE LOGICAL UNIT DESCRIPTION (CRTLUD) TEMPLATE

The following fields of the LUD must be initialized as indicated for the 3410/3411 tape devices.

Field Name	Entry
LUD type	Char 10
Device type	Char 3410
Model number	Char 0001, Char 0002, or Char 0003
Forward object pointer	Pointer to CD
Physical address	
• LU address	Hex 0000, Hex 0001, Hex 0002, or Hex 0003
• CD OU number	Hex 0015
Power control	Hex 0000
Session definition data	Bin 0
Load/dump definition data	
• Load/dump device	Hex 00 = Not used as a load/dump device Hex 01 = Used as a noninterruptible and nonexchangeable load/dump device Hex 21 = Used as an exchangeable load/dump device
Operating mode	
– Normal	Hex 00 = Normal
– Load mode	Hex 01 = Load mode
– Dump mode	Hex 02 = Dump mode
Specific characteristics length	Hex 0000
Retry value length	Hex 000C (two retry values)
Retry values (see *Retry Values* later in this chapter)	Char(6)
• Error type	Char(2)
• Error retry value	Bin(2)
• Reserved (binary 0)	Bin(2)
Error threshold length	Hex 0010 (two error threshold values)

Field Name	Entry
Error threshold values (see *Error Threshold Values* later in this chapter)	Char(8)
• Error type	Char(2)
• Threshold value	Bin(2)
• Reserved (binary 0)	Char(4)
Device-specific contents length	Hex 005A (90)
Modifiable length	Hex 0056 (86)
Device-specific contents (see *Device-Specific Contents* later in this chapter)	Char(90)

Retry Values

Error Type	Error Description	Error Retry Value
Hex 0001	I/O error, 3411 I/O check, or data check on Read command	Value (10-20) Suggested normal value = 10
Hex 0002	I/O error, 3411 I/O check, or data check on Write command	Value (15-30) Suggested normal value = 15

Error Threshold Values

Error Type	Error Description	Threshold Value
Hex 0001	I/O error, 3411 I/O check, or data check on Read command	Value (1-10) Suggested normal value = 5
Hex 0002	I/O error, 3411 I/O check, or data check on Write command	Value (1-64) Suggested normal value = 32

Device-Specific Contents: Char(90)

Bytes 0-5	Active session flags	Char(6)
• Byte 0		Char(1)
– Bit 0	Tape encoding 0 = EBCDIC 1 = ASCII	
– Bit 1	Tape density 0 = 1600 BPI 1 = 800 BPI	
– Bits 2-7	Reserved (binary 0)	
• Bytes 1-2	Block length Hex 0000 for variable-length	Char(2)
• Byte 3	Reserved (binary 0)	Char(1)
• Bytes 4-5	Load/dump block counter updated by the number of blocks transferred	Char(2)
Bytes 6-85	Value label	Char(80)
Bytes 86-89	Reserved (binary 0)	Char(4)

The first 86 bytes of the device-specific parameters (the active session flags and volume label fields) are modifiable through the Modify Logical Unit Description instruction. The block length, for non-load/dump operations is used only as an attribute of the tape volume when logging hardware errors or volume SDRs (statistical data records). When dumping objects under load/dump, the block length is ignored. A block length of 16 384 is always used when dumping objects.

For proper logging of volume SDRs, each time that a different tape is mounted there must be a device specific area change message issued through the Modify Logical Unit Description instruction. This would normally be required to update the block length and volume label fields.

3410/3411 MODIFY CONTROLLER DESCRIPTION (MODCD) INSTRUCTION

The following is a list of functions the 3410/3411 tape controller supports through the MODCD instruction:

- Power on
- Power off
- Vary on
- Vary off

See Chapter 17 for the meaning and use of each function.

3410/3411 MODIFY LOGICAL UNIT DESCRIPTION (MODLUD) INSTRUCTION

The following is a list of functions the 3410/3411 tape device supports through the MODLUD instruction.

- Vary on
- Vary off
- Activate
- De-activate
- Suspend
- Quiesce
- Reset
- Resume (activate after suspend, quiesce or reset)
- Error threshold sets
- Retry value sets
- Load/dump definition data
- Device-specific contents

The device-specific area of the LUD (that portion that is modifiable) may be altered at any time. It should be noted that any changes to the device-specific area are reflected in the machine processing. For this reason, the area should be altered prior to beginning the processing of the tape media and not during processing. Changes in tape density will be effective only if the tape is positioned at load point.

The machine maintains a copy of the load/dump block counter. The value in the device-specific area is copied when the LUD is varied on and when there is a device-specific area change and is updated whenever a load/dump message is returned. This requires the user to initialize the counter and avoid making any device-specific area changes while a load/dump message is outstanding.

For the 3410 tape device to support the rewind or rewind and unload commands with immediate response (3410/3411 Request I/O instruction), the IOM may require up to 4 minutes to complete a Modify LUD (vary off) operation so that the physical rewind can complete before the device is varied off. The user of a Modify LUD (vary off), where rewind operations with immediate response may still be outstanding, must ensure that the Modify LUD time-out value is sufficient to allow the rewind to complete or a partial damage (time-out) exception condition will result.

The load/dump definition data field is used to specify the load/dump operations performed by the 3410 tape devices. The operations that are allowed for the 3410 tape devices are controlled by the value that was specified in the load/dump device field at Create LUD time. The following is a list of the load/dump device entries followed by the operations that are allowed for each entry.

Field Name	Entry	Load/Dump Operation
• Load/dump device The remaining load/dump definition data fields are not used.	Hex 00	Not used for load/dump
• Load/dump device	Hex 01	Noninterruptible and nonexchangeable device
Operating mode (use one of the values)	Hex 00	Normal mode (data interchange)
	Hex 01	Load mode
	Hex 02	Dump mode
The remaining load/dump definition data fields are not used.		

Field Name	Entry	Load/Dump Operation
• Load/dump device	Hex 21	Exchangeable
Operating mode (use one of the values)	Hex 00	Normal mode (data interchange)
	Hex 01	Load Mode (primary device for exchanges)
	Hex 02	Dump mode (primary device for exchanges)
	Hex 21	Load mode (alternate device for exchanges)
	Hex 22	Dump mode (alternate device for exchanges)
Load/dump pending	Hex 0000	Not used for exchange operations
Corresponding primary address (use one of the values)	Hex 0000	When the operating mode is hex 00, hex 01, or hex 02
	Hex 0001 Hex 0002 Hex 0003	When the operating mode is hex 21 or hex 22 use the logical unit address of the primary mode device.
Load/dump exchange status	Hex 000000	Normal mode (no modification requested)
	Hex 010000 Hex 010000 Hex 010001 Hex 010002 Hex 010003	The last 2 bytes of this field (hex 0000, hex 0001, hex 0002, or hex 0003) indicate the logical unit address of the LUD that this Modify instruction causes to become current. The other LUD is modified to not current.
Load/dump exchange status (indications for a Materialize LUD instruction)	Hex 000000	This device is not current.
	Hex 010000	This device is current.

3410/3411 REQUEST I/O INSTRUCTION (REQIO) INSTRUCTION

The SSR for a Request I/O instruction to one of the tape drives contains the following values:

Fields	Values
Source/sink object	Pointer to the corresponding tape LUD
Response queue	Pointer to response queue
Source/sink data area	Space pointer
Optional pointer	Reserved (binary 0)
Request priority	See *Request I/O (REQIO)* in Chapter 17
Request identification	See *Request I/O (REQIO)* in Chapter 17
Function field	Hex 80
Control field	N or C
Key length	Bin(2)
Key offset	Bin(2)
RD count	See below
RD offset	Bin(2)

At least one RD (request descriptor) must exist with each Request I/O instruction (except continue) or a template value invalid exception is returned. A Request I/O (continue) instruction must have an RD count of 0. Only one read or write RD may exist with each Request I/O instruction. Any number of the other RDs may exist with or without a read or write RD.

All label processing must be done by the user. The machine will not check for this. An example is the sequence of trailer labels which must be issued to the machine in accordance with the tape standards specification.

The device-specific area in the LUD must specify ASCII encoding when the tape is encoded (or is to be encoded) in ASCII. The machine will convert the data from ASCII to EBCDIC on a read operation or from EBCDIC to ASCII on a write operation.

Request Descriptor

The format of the RD is:

Byte 0	Command	Char(1)
	Hex 02 – Read block	
	Hex 01 – Write block	
	Hex 12 – Read block backwards	
	Hex 14 – Forward space block	
	Hex 1C – Backward space block	
	Hex 24 – Forward space file	
	Hex 2C – Backward space file	
	Hex 34 – Rewind with delayed response	
	Hex 35 – Rewind with immediate response	
	Hex B4 – Rewind and unload with delayed response	
	Hex B5 – Rewind and unload with immediate response	
	Hex 54 – Write tape mark	
	Hex 4C – Tape clear	
	Hex 0A – Check tape	

Byte 1	Control[1]	Char(1)
• Bit 0	0 = Perform command 1 = Ignore command	
• Bits 1-7	Reserved (binary 0)	
Bytes 2-3	Operations performed[1]	Bin(2)
Bytes 4-5	Block size detected[1]	Bin(2)
Byte 6	Reserved (binary 0)	Char(1)
Byte 7	Reserved (binary 0)	Char(1)
Byte 8	Number of block in error[1]	Char(1)
Byte 9	Number of operations to perform	Char(1)
Bytes 10-11	Block size 18-32 768 bytes	Bin(2)
Bytes 12-13	Number of byte in error[1]	Bin(2)
Bytes 14-15	Reserved (binary 0)	Bin(2)

[1]Fields are modified by an REQIO execution.

The *command* field indicates the type of operation to be performed. The specific operations are described with the individual commands.

The *control* field indicates whether the RD is to be ignored or not and is generally a binary 0 when the instruction is issued. This field is set by the machine to a binary 1 when the RD is successfully completed. The command control field can be used in conjunction with the current RIU segment field in restarting a request that ended with an error. The already completed RDs will be ignored and the RD that had the error will be reexecuted. After normal completion, this field must be reset before the SSR can be reused.

The *operations performed* field indicates the number of operations which were completed satisfactorily before the command ended. The field is normally set to 0 before issuing the REQIO instruction. The field is then incremented as operations are completed successfully. This field in conjunction with the *control field* of previously processed RDs can be used to reissue a REQIO instruction after an error.

The *block size detected* field indicates the block size read if the block size is less than the block size specified. If the block size read is greater than the block size specified, the block size detected will be 0. This allows different tapes to be read because the maximum block size can always be specified and this field will return the actual size of the block read.

The *number of block in error* field contains the number of the block which has invalid data (associated with the data not valid error hex C042).

The *number of operations to perform* field contains the number of blocks, files, tape marks to read, write, or space operations to be performed. This must be greater than 0.

The *block size* field indicates the size of the block or blocks to read or write. This must be greater than or equal to 18 and less than or equal to 32 768.

The *number of byte in error* field contains the number of the byte within the block indicated in the *number of block in error* field which has invalid data (associated with the data not valid error hex C042).

Commands

The following are the commands supported by the Request I/O instruction.

Read Block: With this command the user may specify from 1 to 255 blocks to read. If the number of blocks specified is 0, an error will be returned. The number of blocks specified must not be greater than the source/sink data area buffer size (block count times block size with the block size rounded up to a multiple of 8) or an error will occur. The data area used must be doubleword aligned unless the block size to be read is greater than 32 256 in which case the buffer must be page aligned. Page alignment is recommended even for block sizes less than 32 256 to increase performance. For normal and error completions, the operations performed field indicates the number of blocks read. If an I/O error occurs, the machine retrieves as much data from the block as possible (this does not include attempts to read the block in both directions). If a tape mark is encountered, an error is returned.

If a retry to read an invalid block completes successfully, the machine continues processing. If the error cannot be corrected, the operation stops, the machine returns as much data as possible, and an error is indicated. The tape is positioned so the read head is positioned after the block that caused the error.

If the block size read is different from the block size specified, the machine will stop reading after this block and the block size read will be returned. When a block less than the specified size is read, the unused area is left unchanged. If a block is read and the block size is not a multiple of 8, the data will be transferred and the buffer will be padded with hexadecimal 0's until the block size is a multiple of 8. Padding is based on the specified block size, not the actual block size. For example, an 18-byte block will result in 18 bytes of data and 6 bytes of 0's.

If variable sized blocks are to be read and the size of the blocks is unknown, the maximum block size should be specified. The block along with an indication of its actual size will be returned to the user.

Tape density need not be specified for reading. Tape encoding must be specified to ensure proper operation.

If the data read is to be converted from ASCII to EBCDIC, any characters read that cannot be converted to a corresponding EBCDIC character will be replaced by an asterisk (hex 5C) in the user buffer. This is done without an error indication being returned to the user. This also applies to the Read Block Backward command.

Write Block: With this command, the user may specify from 1 to 255 blocks to write. If the number of blocks specified is 0, an error will be returned. The number of blocks specified must not be greater than the source/sink data area buffer size (block count times block size with the block size rounded up to the next multiple of 8) or an error occurs. The data area must be page aligned if the block size to be written is greater than 32 256; otherwise, the data area need only be doubleword aligned. For any condition, the user is notified of the number of blocks written. If the end of tape marker is sensed when attempting to write a block, the block is written and the user notified that this occurred.

After the end of tape has been sensed, subsequent write commands that are issued with the tape positioned past the end of tape marker continues to have this error indicated and only one block will be written per request. There is a remote possibility that the above will not occur if a recoverable error occurs in the vicinity of the end of tape marker. If this happens, the end of tape may be indicated only once. Subsequent write commands could cause the tape to run off the end of the reel.

If a write command results in an error and the machine, after retrying the command, successfully completes the operation, the machine will continue processing. If the error is unrecoverable, the operation stops and an error is indicated. The tape is moved until the write head is over the area just after the block that caused the error.

If a block of data is written and the block size is not a multiple of 8, the data bytes greater than the block size but less than the next multiple of eight are ignored. Tape density and tape encoding must be specified in the LUD to ensure proper operation.

If the data is to be converted from EBCDIC to ASCII format, the data is copied from the SSD area to an internal buffer and then converted in the internal buffer. If an EBCDIC character is encountered that cannot be converted to ASCII, the number of block in error and number of byte in error fields in the RD specifies which byte in the SSD is invalid.

Read Block Backward: With this command, the user may specify from 1 up to 255 blocks to read. If the number of blocks specified is 0, an error will be returned. The number of blocks specified must not be greater than the source/sink data area buffer size (block count times block size with the block size rounded up to a multiple of 8) or an error is indicated. The user supplies the address of the low-order end of the buffer, as with the Read Block command. The data area used must be page aligned if reading a block greater than 32 256; otherwise, the data area need only be doubleword aligned. Page alignment is recommended even for block sizes less than 32 256 to increase performance. If beginning of tape or a tape mark is encountered, an error will be returned. For any condition, the user is notified of the number of blocks read. In the case of reading a block and an I/O error occurs, the machine retrieves as much data as possible and the tape is positioned after (in the direction of travel) the block in error. The data is stored in the buffer from the high-order address to the low-order address. For reading blocks less than the specified size, the operation transfers data from the high-order address toward the low-order address leaving unchanged that low-order area not used. If the block size is not correct, only one block is read backward. If a block is read and the number of bytes it contains is not a multiple of 8, it is padded with hexadecimal 0's to the next multiple of 8. Padding occurs at the high-order address end of the buffer.

If variable sized blocks are to be read and the size of the blocks is unknown, the maximum block size should be specified. The block along with an indication of its actual size will be returned to the user. If the size of the block read is not the size specified, the operation ends with an error indication.

If the error 'tape moving backward at beginning of tape' is returned, 8 bytes of 0's will be transferred to the user's buffer.

Tape density need not be be specified in the device-specific area of the LUD. Tape encoding must be specified to ensure proper operation.

Space Block and File Commands: With this group of commands, the user may position the tape at a particular block or file relative to the current tape position. From 1 to 255 blocks or files can be spaced. A space or skip of 0 blocks or files results in an error response. When spacing the tape forward or backward, the tape will stop beyond the block or file requested in the direction of travel. If, when spacing the tape forward, more blocks are requested than exist in the file and a tape mark is detected, an error is returned. If the tape is positioned in the inter-block gap before a tape mark and it is desired to have the tape positioned after the tape mark, a Skip File command must be issued. If the same conditions exist but no tape mark is detected, an error is returned and the tape runs off the reel. If, while spacing the tape forward, more files are requested than exist on the tape, the tape will run off the reel and an error is returned. An example of a Forward Space Block command is if the current position is after block 1 and the request is to move two blocks, the resulting position after processing will be after block 3.

The Forward Space File command moves the tape forward and stops it immediately after the specified number of tape marks are sensed. For example, if the request is made to move forward two files, the machine will search for two tape marks and stop the tape when the second tape mark is sensed.

The Backward Space File command operates in the same manner as the Forward Space File command except that the tape stops just after (in the direction of tape travel) the specified number of tape marks have been sensed or at the beginning of the tape, whichever comes first. An example of a Backward Space File command is, if the current position is in any file and the request is to move backwards one file, the resulting position after processing will be in the gap after the tape mark encountered in the direction of travel or at the beginning of the tape whichever occurs first.

Rewind/Rewind and Unload (hex 34, hex 35, hex B4, or hex B5): These commands are used to rewind or rewind and unload the tape. These commands position the tape at its beginning or put it in a state to be dismounted. If the command specifies delayed response, the request is not returned to the user until the operation completes. If the command specifies immediate response, the request is returned to the user when the operation begins unless other RDs follow this command in the Request I/O.

Write Tape Mark: With this command, a special block is written on the tape that indicates the end of a tape file. The user can specify from 1 to 255 tape marks.

Tape Clear: This command erases the tape from its current position to the end-of-tape marker. Any data past the end-of-tape marker is not erased. Any data past the end-of-tape marker can be erased by repeatedly issuing this command. Each time this command is issued with the tape positioned past the end of tape an additional 3.6 inches of tape is erased.

Check Tape: This command checks the status of the specified tape drive and causes a mode set command to be issued to the tape drive which sets the density indicated in the device-specific area of the LUD. The following tape drive status conditions may be indicated in the feedback record as a result of this command.

- Tape drive ready and at load point (normal response)
- Tape drive ready but not at load point (error response)
- Tape drive busy searching for load point (error response)
- Tape not mounted on drive (error response)
- Start button not pressed (error response)

3410/3411 FEEDBACK RECORD AND ERROR RECOVERY PROCEDURES

The format of the feedback record is as follows:

Bytes 0-15	Source/sink request address	Space pointer
Bytes 16-17	Request identification	Bin(2)
Bytes 18-19	Error summary	Bin(2)
Bytes 20-21	RD number	Bin(2)
Bytes 22-23	RIU segment count	Bin(2)
Bytes 24-63	Device-dependent area	Char(40)
• Bytes 24-25	Device-dependent error code	Char(2)
• Bytes 26-27	Hardware error code	Char(2)
• Bytes 28-35	Time stamp	Char(8)
• Bytes 36-37	Operating unit number	Char(2)
• Bytes 38-63	Reserved (binary 0)	Char(26)

Descriptions of the request address and request ID fields are given under *Request I/O (REQIO)* in Chapter 17.

The *error summary* field defines the status of the Request I/O instruction defined in the Request I/O instruction description. The specific values possible for the tape subsystem are itemized in Figure 23-7 which follows.

The *RD number* is the index of the last RD processed or the RD in error if an error is indicated.

The *RIU segment count* is the number of blocks, tape marks, or files transferred to or from the SSD by this request.

The *device-dependent* area is all binary 0 unless the presence of device-dependent data is indicated by the error summary value. If the device-dependent data is present, the fields have the following definitions.

The *device-dependent error code* is a further categorization of the hardware error codes shown in Figure 23-7.

The *hardware error code* is logged in the hardware error log and indicates the specific hardware error encountered. The possible values are shown in Figure 23-7. This code is also provided as event-related data for the CD failure event or the LUD failure event when these are also signaled.

The *time stamp* and *operating unit number* are the same values present in the hardware error log entry and the failure event data and are the values used to correlate the FBR, the event, and the error log entry for maintenance purposes.

Figure 23-7 lists the error summary, device-dependent error codes, the hardware error log codes that caused the error, and the recommended recovery action. If more than one error occurs, the error status returned will be returned on a worst condition basis. Example: For invalid block size detected and a tape media failure, the tape media failure condition is returned as the status.

When the recovery action indicates that the tape job must be restarted, this must include a de-activate session.

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
0000	N/A	N/A	No error condition	N/A
0008	N/A	N/A	Request I/O (continue) instruction complete	N/A
4000	N/A	N/A	I/O error occurred but was corrected	N/A
C009	N/A	N/A	Partially processed request, terminated by reset	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (de-activate)instruction must be issued to destroy the session.
C00A	N/A	N/A	Unprocessed request because of reset session on error	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (de-activate) instruction must be issued to destroy the session.
C016	N/A	N/A	Tape mark encountered	Issue REQIO (continue) to resume processing.
C017	N/A	N/A	End of tape sensed	End of tape sensed during an I/O operation, issue REQIO (continue) instruction to restart drive. Continued use may cause tape to run off reel.
C042	N/A	N/A	Data not valid. EBCDIC to ASCII conversion error	Correct character in error, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
C043	N/A	N/A	Invalid buffer alignment	Allocate SSD to the proper alignment, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
C044	N/A	N/A	SSD area not large enough	Make smaller blocks or larger SSD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
C084	N/A	N/A	Invalid pointer to SSD	Correct pointer, reissue the REQIO instruction, and issue an REQIO (continue) instruction.

Figure 23-7 (Part 1 of 6). 3410/3411 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardward Error Code (hex)	Meaning	Recovery Action
C085	N/A	N/A	Invalid function field	Correct the REQIO, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
C087	N/A	N/A	Invalid RD	Correct the RD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
C088	N/A	N/A	Invalid RD sequence	User may have only one Read or Write command for each request.
E010	0001	0000	This problem was caused by an error that was indicated in a previous feedback record for this device or another device in this subsystem.	Correct the problem indicated by the previous feedback record.
		1300 3100 3300 3400 6300 8800 8A00 E110 E120 E130 E160 E210 1200 2300 E140 E150 E170 E290 C200 B200	Subsystem failure	Not correctable, call the service representative. Tape position cannot be determined. Any attempt should include vary off and vary on CD. The CD failure event is also signaled for these cases.

Figure 23-7 (Part 2 of 6). 3410/3411 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	0002	0000	This problem was caused by an error that was indicated in a previous feedback record for this device or another device in this subsystem.	Correct the problem indicated by the previous feedback record.
		B1XX C1XX 74XX 77XX B600 8C00 B0XX B300 7100 7300 7600 8900 XX = DSTAT 15	Tape drive failure	A permanent tape drive failure has occurred. Rerun the job using a different tape drive. The LUD failure event is also signaled for these cases. Recovery should include vary off and vary on LUD.
E010	0003	0000	This problem was caused by an error that was indicated in a previous feedback record for this device or another device in this subsystem.	Correct the problem indicated by the previous feedback record.
		B400 B700 B800 B900 BB00 BC00 BD00 BE00 C600 BF00 C300 C400 C500 C700 C800 C900 CA00 CB00 CC00 B500 7500 7800 7900	Possible tape media failure	Try a different tape or clean heads and columns. If error persists call the service representative.

Figure 23-7 (Part 3 of 6). 3410/3411 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	0004	8500 8501	Enable/disable switch in disable position	Abort all tape jobs, move switch to enable position and issue a vary off CD followed by a vary on CD.
E010	0005	8300 8301	Subsystem power off	Vary off the tape subsystem and then issue a power off command for it. Power the subsystem back on and vary it on again. Rerun all jobs. If this error occurs again, a permanent failure has occured.
E010	0006	8600	Write ring is missing	Correct problem, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E010	0007	8B00	NRZI tape on PE only drive	Rerun the job on a phase encoding device that has the dual density feature installed.
E010	0010	N/A	Invalid RD command	Correct the RD and reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E010	0012	7200	Equipment check	If all tape has run off the user's reel, check tape to be sure that an end-of-tape marker is located about 25 feet from the end of tape. If none is present, attach one and rerun job. If EOT marker is present, and error C017 has previously occurred, a program error is indicated. If all tape has not run off of user's reel, a permanent tape drive failure has occurred. Rerun the job using a different tape drive.
E010	0014	BA00	Phase encoding ID burst error	Move beginning-of-tape marker 2 cm and restart job.
E014	0008	8100	At load point but not ready	Make the drive ready and restart the job.
E014	0012	8200	Tape searching for load point	Wait, reissue command and issue an REQIO (continue) instruction.

Figure 23-7 (Part 4 of 6). 3410/3411 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E014	0013	8100	Tape not mounted on drive, or tape mounted on drive but not at load point and Start key not pressed.	If tape is not mounted, mount the tape, make the drive ready and restart the job. If tape is mounted, make the drive ready and restart the job.
E018	000A	8700	Tape moving backward at beginning-of-tape	Check program to verify correct coding and issue an REQIO (continue) instruction.
E018	000B	N/A	Wrong length record detected by device	Check command modifier for correct block size, issue an REQIO (continue) instruction.
E018	000C	N/A	Tape mounted and ready but not at load point	Issue rewind command and an REQIO (continue) instruction.
E087	000D	N/A	Invalid block size specified block size < 18 or block size > 32768	Correct the block size, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	000E	N/A	Block count to read/write not given (0) or block count given equals RD bytes 3-4	Correct the block count, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	000F	N/A	Invalid number of blocks, files, or tape marks; number must be from 1 to 255	Insert the valid number of blocks, files, or tape marks, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	0015	N/A	Reserved fields in RD not 0. No processing on the REQIO has occurred and the RD number field in the FBR contains 0.	Correct the RD, reissue the REQIO instruction and issue an REQIO (continue) instruction.

Figure 23-7 (Part 5 of 6). 3410/3411 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
FODF	N/A	4100 1500 1100 2100 4200 4300 4400 5100 6100 1400 5200 5300 5400 5500 5501 5600 5700 62XX 8400 XX = DSTAT 15	I/O error	All tape jobs must be restarted. Recovery should include vary off and vary on CD. If error persists, call the service representative. The CD failure event is also signaled for these error cases.

Figure 23-7 (Part 6 of 6). 3410/3411 Error Summary Values

Events

In addition to the events described under *Request I/O (REQIO)* in Chapter 17, the following events are signaled. For a complete description of the following events, see *Chapter 21. Event Specifications.*

- CD contact event (hex 0004 04 01,02)

  This event is signaled for the MSCP component when the vary on processing is completed for this CD. This processing is done synchronous to the execution of the Modify CD (vary on) instruction so only subtype 01 (successful contact) is signaled. Subtype 02 (unsuccessful contact) is not signaled on behalf of tape controllers since the Modify CD is terminated and an exception is signaled instead.

- CD failure event (hex 0004 05 01)

  The CD failure event is signaled and an error log entry is made whenever the entire subsystem receives a nonrecoverable error. As part of the recovery action for this event, the CD should be varied off and then varied back on before attempting further operations. The variable data provided for this event is formatted as follows:

  Bytes 0-1 Hardware error code: code that further defines the specific error encountered

  Bytes 2-9 Time stamp: contains the time stamp of the corresponding error log message

  Bytes 10-11 Operational unit number (hex 0015)

  Bytes 12-13 Optional data: the 2-byte optional data field returned with the event has the following meaning:

Condition	Description and Operator Action
Byte 12, bit 0	Subsystem failure due to I/O error. This case may be retryable.
Byte 12, bit 1	Subsystem failure. This case is not correctable. Seek hardware assistance.

  Remaining bits of byte 12 and all of byte 13 are unused and are 0.

- LUD contact event (hex 000B 06 01,02)

  This event is signaled when the LUD vary on processing is completed by the MSCP. Subtype 01 is signaled upon successful contact, however, subtype 02 (unsuccessful contact) is never signaled for Tape devices. In these cases, the Modify LUD (vary on) instruction signals an exception instead. This exception case can occur for the same reasons that the CD failure event is signaled.

- LUD failure event (hex 000B 08 01)

  The device failure event is signaled and an error log entry is made whenever a tape drive has an uncorrectable problem (that is, where the recovery action is to call the service representative). As part of the recovery action for this event, the LUD should be varied off and then varied back on before attempting further operations. The variable data provided for this event is formatted as follows:

  Bytes 0-1 Hardware error code: code that further defines the specific error encountered

  Bytes 2-9 Time stamp: contains the time stamp of the corresponding error log message

  Bytes 10-11 Operational unit number (hex 0015)

  Bytes 12-13 Optional data: the 2-byte optional data field returned with the event has the following meaning:

Condition	Description and Operator Action
Byte 12, bit 0	Device failure due to I/O error.
Byte 12, bit 1	Device not attached. The status bytes will contain hex A200 in this case.
Byte 13, bits 0-1	Drive number of device in error.

  Remaining bits of bytes 12 and 13 are unused and are 0.

- Request I/O instruction complete event (hex 000B 09 01)

  The Request I/O instruction complete event is signaled if the user requests it in the Request I/O instruction.

- Response queue destroyed event (hex 000B 0A 01)

  The response queue destroyed event is signaled if the user truncates or destroys the Request I/O instruction response queue while the machine is processing Request I/O instructions.

Exceptions

The following table gives the cases where the source/sink resource not available exceptions (hex 3404) are signaled when a Modify CD instruction for the 3410/3411 tape device CD is executed.

Command	Defect Code (hex)	Device-Specific Return Code (hex)	Meaning
Power On	2206	0602	Power failure occurred. The CD failure event has been signaled.
Power Off	2207	0602	Same as above.
Vary On	2202	0001	I/O controller failure. CD failure event has been signaled.

The following table gives the cases where the source/sink resource not available exceptions (hex 3404) are signaled when a Modify LUD instruction for the 3410/3411 tape device LUD is executed.

Command	Defect Code (hex)	Device-Specific Return Code (hex)	Meaning
Quiesce Session	2313	1302	Quiesce rejected due to terminating error condition.

3203-5 PRINTER PROGRAMMING CONSIDERATIONS

The basic object of control for the 3203-5 Printer is the logical unit description (LUD). All references to a device are made with respect to the LUD. The Create, Modify, and Destroy LUD instructions establish and control the environment in which the printer operates. The Request I/O instruction controls the device and causes it to print data.

Before request I/O operations can be accepted, several steps must occur.

1. An LUD must be created through use of the Create LUD instruction.

2. The printer must be varied on through use of the Modify LUD instruction.

3. The LUD must be made active through use of the Modify LUD (activate) instruction.

3203-5 PRINTER CREATE LOGICAL UNIT DESCRIPTION (CRTLUD) TEMPLATE

The following fields of the logical unit description template must be initialized as indicated:

Field Name	Entry
LUD type	Char 00
Device type	Char 3203
Model number (must be stand-alone)	Char ƀƀƀ5
LUD operational unit number	Hex 0040 for the first printer Hex 0041 for the second printer
Power control	Hex 0100
Session definition data	Bin 0
Load/dump indicator	Bin 0
Specific characteristics length	Hex 0000
Retry value length	Hex 0000
Error threshold length	Hex 0000

Field Name	Entry	
Device-specific contents length	501	
Device-specific modifiable length	501	
Device-specific area		Char(501)
Byte 0	Control flags	Char(1)
• Bits 0-1	Reserved (binary 0)	Bit(2)
• Bit 2	Write control 0 = No data translation 1 = Translate the data	Bit(1)
• Bits 3-7	Reserved (binary 0)	Bits(4)
Byte 1	Lines per inch	Char(1)
Byte 2	Lines per form	Char(1)
Bytes 3-4	Character set length	Bin(16)
Bytes 5-308	Universal character set buffer (USCB)	Char(304)
• Bytes 5-244	Train image	Char(240)
• Bytes 245-308	Dualing and uncomparable character table (DUCT)	Char(64)
Bytes 309-500	Translate table	Char(192)

The device-specific parameters may or may not be supplied in the device-specific area of a create template. If they are not supplied at create time, the create template must still contain the 501-byte area and this area should be set to 0. The LUD will be created to contain whatever is in the template, without validating any of these parameters. These parameters are not used until after the LUD is in the active session state.

3203-5 PRINTER MODIFY LOGICAL UNIT DESCRIPTION (MODLUD)

The following is a list of functions the 3203 supports through the MODLUD instruction:

- Power on
- Power off
- Vary on
- Vary off
- Activate
- De-activate
- Suspend
- Quiesce
- Reset
- Resume (activate after suspend, quiesce, or reset)
- Modify device-specific area

See Chapter 17 for the meaning of each function.

LUD Device-Specific Area

The device-specific area contains the information that is passed to the printer attachment to control the line spacing, form size, universal character set buffer, train image, dualing and uncomparable character table, and translate table used by the printer. This area may be altered by Modify LUD instruction at any time. Changing any portion of the device-specific area causes all device-specific area parameters to be written to the printer. If the number of lines per inch or the number of lines per form is redefined, the line counter is set to the top of the form and it may be necessary for the operator to reposition the form to line one on the printer. Changes to the device-specific area are reflected when the machine processes the next Request I/O instruction.

3203-5 PRINTER REQUEST I/O (REQIO) INSTRUCTION

The Request I/O instruction is used to request the 3203-5 Printer to perform its various I/O functions.

The SSR for a Request I/O instruction to the 3203-5 Printer contains the following values:

Field	Value
Source/sink object	System pointer to LUD
Response queue	System pointer to response queue
Source/sink data area	Space pointer
Optional pointer	Reserved (binary 0)
Request priority	See *Request I/O (REQIO)* in Chapter 17
Request identification	See *Request I/O (REQIO)* in Chapter 17
Function field	Hex 80
Control field	N or C
Key length	Bin(2)
Key offset	Bin(2)
RD count	0 for Request I/O (continue) instruction 1 for Request I/O instruction
RD offset	Bin(2)

The 3203-5 Printer supports only one RD for each Request I/O instruction. The RD count field must be one for request I/O (normal) operations; however, for request I/O (continue) operations, this field is ignored. The RD offset field indicates the offset from the start of the of the SSR to the 16-byte RD. The source/sink data area (SSD) contains the data to be printed corresponding to the command in the RD.

The format of the RD is as follows:

Byte 0	Command Hex 41 – Print SCS data Hex 42 – Continue printing after error	Char(1)
Bytes 1-3	Command modifiers	Char(3)
Bytes 4-15	Reserved (binary 0)	Char(12)

Print SCS Data Command (hex 41)

The Print SCS Data command causes the data in the SSD to be printed in the format specified by the SCS command embedded in the SSD. The command modifier bytes (bytes 1-3 of the RD) contain additional information and have the following format:

Byte 1 — Char(1)

- Bits 0-4 — Reserved (binary 0)
- Bit 5 — Force full completion
  - 0 = Maximum throughput (normal mode)
  - 1 = Return completion status only after all data is printed
- Bit 6 — Unprintable character detection
  - 0 = Signal unprintable character detected error
  - 1 = Do not signal an error to the user
- Bit 7 — Continue
  - 0 = Start printing at beginning of the SSD
  - 1 = Retain any data saved from previous Print SCS Data command and continue printing where printing stopped

Bytes 2-3 — Data block group count — Bin(2)

The force full completion bit, set to 0, allows the channel to be released for other operations before a print line complete status is returned from the print adapter for each line of print. The user should be aware that an error on the last line of print will be indicated on the next Request I/O instruction.

When the force full completion bit is set to 1, the channel is not released until all the data is printed.

Normally, the continue bit should be on to ensure that all data is printed.

The data block group count field is the number of 8-byte groups in the SSD. The data block count must be from 1 to 8192; otherwise, an error occurs.

The print data area must be loaded before the Print SCS Data command is issued.

Continue Printing After Error (hex 42)

The Continue Printing After Error command (hex 42) can be used to recover from an error that occurred on a print Request I/O instruction that used a Print SCS Data RD command (hex 41). When an error occurs, a print Request I/O instruction with a Continue Printing After Error (hex 42) RD command can be issued to continue printing as if the previous error had not occurred. The SCS data is saved from the previous SCS print, and printing is continued from where the error occurred. The command modifier bytes (bytes 1-3 of the RD) contain additional information and have the following format:

Byte 1 Char(1)

- Bits 0-4 Reserved (binary 0)
- Bit 5 Unused
- Bit 6 Unprintable character detection
  0 = Signal unprintable character detected error
  1 = Do not signal an error to the user
- Bit 7 Unused

Bytes 2-3 Unused

The SSD pointer value is ignored by this command.

This command should be used for only hardware errors that do not require a response. If this command is used for errors that require a response, the results are unpredictable. The user of this command should not change the print data area that was used by the Request I/O instruction that encountered the error. An invalid RD command error is signaled if the preceding Request I/O instruction did not have a valid hardware error.

A Request I/O (continue) instruction must be issued to continue processing.

STANDARD CHARACTER STREAM (SCS)

The SSD contains the SCS. The SCS is used by the 3203-5 Printer for transferring print data and SCS commands from the system to the printer. With SCS, Print Data and SCS commands are sent to the attachment in free-form; that is, SCS commands can appear anywhere within the print data stream. The SCS commands have values of hex 00 through hex 3F, and hex FF.

Print data characters have values from hex 40 through hex FE. Any character not recognized as a printable character prints as a blank and an unprintable character condition is returned in the feedback record. If the translate option is used (write control bit is on in the LUD), characters are handled as defined by the user.

SCS COMMANDS

The SCS commands control carriage operations. The SCS commands for the 3203-5 Printer are the same as the SCS commands for the 3262/5211 Printer. The 3262/5211 Printer section of this chapter has a detailed description of the SCS commands.

3203-5 FEEDBACK RECORD AND ERROR RECOVERY PROCEDURE

The format of the feedback record is as follows:

Bytes 0-15	SSR address	Space pointer
Bytes 16-17	Request identification	Bin(2)
Bytes 18-19	Error summary	Bin(2)
Bytes 20-21	RD number	Bin(2)
Bytes 22-23	RIU segment count	Bin(2)
Bytes 24-63	Device-dependent area	Char(40)
• Bytes 24-25	Device-dependent error code	Char(2)
• Bytes 26-27	Hardware error code	Char(2)
• Bytes 28-35	Time stamp	Char(8)
• Bytes 36-37	Operating unit number	Char(2)
• Bytes 38-63	Reserved (binary 0)	Char(26)

See *Request I/O (REQIO)* in Chapter 17 for descriptions of the request address and request ID fields.

The error summary field defines the status of the Request I/O instruction as defined under *Request I/O (REQIO)* in Chapter 17. The specific values possible for the 3203-5 Printer are shown in Figure 23-8.

The RD number is the index of the last RD processed or the RD in error if an error is indicated. For the 3203-5 Printer, the RD number is 1.

The RIU segment count is the number of forms completed by the printer before an error occurred. If no error occurred, this field is not defined.

The device-dependent area is all binary 0's unless the presence of device-dependent data is indicated by the error summary value in the error summary field. (See *Description* under *Request I/O (REQIO)* in Chapter 17 for the error summary field definition.) If device-dependent data is present, the field has the values shown in Figure 23-8.

As shown in Figure 23-8, the device-dependent error code is a further categorization of the hardware error codes.

The hardware error code is the same value as that logged in the hardware error log and indicates the specific hardware error encountered. The possible values are shown in Figure 23-8.

The time stamp and operating unit number are the same values present in the hardware error log entry and are used to correlate the FBR and the error log entry for maintenance purposes.

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
0000	N/A	N/A	Normal completion	N/A
0008	N/A	N/A	REQIO (continue) instruction response	N/A
C009	N/A	N/A	Partially processed request terminated because of reset session	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (de-activate) instruction must be issued to destroy the session.
C00A	N/A	N/A	Unprocessed request because of reset session	An MODLUD (activate) instruction must be issued to restart processing, or an MODLUD (de-activate) instruction must be issued to destroy the session.
C043	N/A	N/A	Invalid SSD boundary alignment	Correct the boundary alignment, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C044	N/A	N/A	SSD too small	Correct the boundary alignment, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C084	N/A	N/A	Invalid pointer to SSD	Replace SSD pointer with a valid pointer, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C085	N/A	N/A	Invalid function field	Correct the function field value, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
C086	N/A	N/A	Invalid RD count	Remove extra RDs, reissue an REQIO instruction, and issue an REQIO (continue) instruction to restart processing.
E010	0002	4450	Invalid SCS command	Check SCS codes, correct command in error, and reprint the form. An REQIO (continue) instruction must be issued to restart processing.

Figure 23-8 (Part 1 of 3). 3203-5 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	0003	5437	Forms jam	Correct the forms jam and reprint the form. An REQIO (continue) instruction must be issued to restart processing.
		7030	Stacker full or jammed	Correct the stacker jam and reprint the form. An REQIO (continue) instruction must be issued to restart processing.
E010	0005	N/A	Unrecoverable I/O error (OU task failure)	The MODLUD (reset, de-activate, and vary off) instructions are required to clear this condition.
E010	0006	N/A	Unprintable character detected	A line has printed with blank substitution. If blank substitution is acceptable, issue an REQIO (continue) instruction and reissue the failing REQIO message. If blank substitution is not acceptable, then either specify ignore unprintable characters in the DSA, correct the print data, correct the translate table, or correct the train image. An REQIO (continue) instruction must be issued to restart processing.
E010	0009	N/A	Interlock open	Close interlock on printer and reprint form. An REQIO (continue) instruction must be issued to restart processing.
E010	000C	4020 4040 4080 7004 7006 7014 7101 7102 7103 7106 7108 7109 7120 7121 7122	Printer or hardware adapter failures	Operator intervention is required. Disengage paper and restore to the top of the form. Reprint the form. An REQIO (continue) instruction must be issued to restart processing. If the error persists, call your service representative.

Figure 23-8 (Part 2 of 3). 3203-5 Error Summary Values

Error Summary (hex)	Device-Dependent Error Code (hex)	Hardware Error Code (hex)	Meaning	Recovery Action
E010	000D	3203 3205 4034 4038 4452 7010	System error	Call the service representative.
E010	0019	N/A	Printer offline	Attach the printer to the line, power on, switch online, and make ready. Issue an REQIO (continue) instruction and reissue the failing REQIO message.
E010	001A	N/A	Out of forms	Install forms. An REQIO (continue) instruction must be issued to restart processing.
E087	0007	N/A	Invalid RD command detected or a hex 42 RD command was issued that could not be accepted	Correct RD command byte, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	0008	N/A	Zero data block field in the RD or more than 8192 bytes of data	Correct the data block field, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
E087	0017	N/A	RD invalid, reserved field violated	Correct the RD, reissue the REQIO instruction, and issue an REQIO (continue) instruction.
F075	N/A	4101 4102 4103 4110 4111	Error during error recovery	Call the service representative.

Figure 23-5 (Part 3 of 3). 3262/5211 Error Summary Values

Events

In addition to the events specified under *Request I/O (REQIO)* in Chapter 17, the following events are signaled. For a complete description of the following events, see Chapter 21.

- LUD contact event (hex 000B 06 01)

  This event is signaled when the MSCP when vary on processing is completed for this device. Only subtype hex 01 is signaled upon successful contact; however, subtype hex 02 is never signaled since the MSCP processing is synchronous to the Modify LUD (vary on) instruction and an exception is signaled instead.

- LUD failure event (hex 000B 08 01)

  The LUD failure event is signaled if the printer has a problem that requires the service representative to be called. As part of the recovery action for this event, the LUD should be varied off before attempting further operations.

  The event-related data for this event consists of:

Bytes 0-1	Hardware error log code
Bytes 2-9	Error log time stamp or 0's
Bytes 10-11	Operational unit number (hex 0040 or hex 0041)
Bytes 12-13	Optional data (not used)

  This event is signaled only for those failures that have hardware error codes that correspond to the error summary codes hex F075 and hex E010 0005.

- Operator intervention required event (hex 000B 07 01)

  The operator intervention required event is signaled when the device requires the operator to take some action, but no error has occurred.

  The 14-byte variable data returned with this event is formatted as follows:

Bytes 0-1	Status – code that was included in the error log message if there was a corresponding error log message; otherwise, the status bytes contain a 0.
Bytes 2-9	Time stamp – contains the time stamp of the corresponding error log message; otherwise, the time stamp is 0.
Bytes 10-11	Operational unit number – Hex 0040 or Hex 0041
Bytes 12-13	Optional data – format is as follows:

Condition	Indication	Description
Byte 12 bit (0)	Ready light off	SCS command was received, and the printer was not ready. Printer is stopped and is not ready as a result of a normal stop condition, and no errors exist.

  Recovery Procedures

  Press the Start key.

  The remaining bits of bytes 12 and 13 are 0.

- Request I/O complete event (hex 000B 09 01)

  This event is signaled if the user requests it. See *Request I/O (REQIO)* in Chapter 17 for details.

- Request I/O response queue destroyed event (hex 000B 0A 01)

  This event is signaled if the user truncates or destroys the request I/O response queue while the machine is processing Request I/O instructions.

Exceptions

The following table gives the cases where the source/sink resource not available exceptions (hex 3404) are signaled when a Modify LUD instruction for the machine console LUD is executed.

Command	Defect Code (hex)	Device-Specific Return Code (hex)	Meaning
Power On	2306	0602	I/O device power failure has occurred. LUD failure event has been signaled.
Power Off	2307	0602	I/O device power failure has occurred. LUD failure event has been signaled.
Suspend Session	2312	1203	Suspend session rejected because an operator intervention condition exists.
Quiesce Session	2313	1302	Quiesce failure because a terminating error condition exists.
		1303	Quiesce session rejected because an operator intervention condition exists

Chapter 24. Communications and Locally Attached Work Stations

MACHINE SERVICES CONTROL POINT (MSCP)

The MSCP coordinates supervisory service requests for all users of the shared source/sink resources. Requests for services are entered through the Modify ND, Modify CD, Modify LUD, and Request I/O instructions and may be processed directly by the MSCP or may be redirected to other supervisory components that provide the requested function. Supervisory service requests include:

- Internal machine functions associated with varying on and varying off a communications line or a device
- Switched connection support such as initiating and completing calls
- Requests initiated at a remote device

Certain requests received by the MSCP cannot be serviced within the machine. These requests result in an event being signaled. To ensure that these requests are properly serviced, there must always be a process with the responsibility for handling these requests. Depending on the conditions that exist and the event signaled, system protocol may require further communication with the MSCP or with the device that initiated the request.

Modification of Source/Sink Objects

Many of the functions provided by the MSCP are implicitly requested when an instruction to modify a source/sink object to a vary on or vary off state is requested. Such functions as the activation of the local or channel attached devices are not apparent to the user; however, for communications devices such as logical units on the shared communications links, the availability, allocation, and use of these resources must be understood by the user.

Modify Network Description (MODND)

During processing of the Modify ND (vary on) instruction, the MSCP is requested to activate the link represented by the ND. Activation of the link includes all the logical and physical initialization required before stations on that link can be contacted. When the ND being activated (varied on) represents a connection into the switched network, that communications adapter is initialized but is not prepared for calls until it has been placed in the switched enabled state by a Modify ND (enable) instruction. Once placed in the switched enabled state, the usage of the ND is governed by the option specified in the switched connection method field of the ND template. If only dial in allowed is specifed, the link is activated for incoming calls. If only dial out allowed is specified, the link is activated for outgoing calls. If either allowed is specified, the link is activated for incoming calls; however, if it is necessary to use this facility for a dial out call before a dial in call has occurred, the resource is available and can be allocated. Allocation and use of switched line facilities is controlled by the Modify ND, Modify CD, and Modify LUD instructions to vary the source/sink objects on or off. To control the allocations and use of switched line facilities, an ND candidate list is maintained in the CD for each controller capable of using the switched network facilities.

Modify Controller Description (MODCD)

During the processing of a Modify CD (vary on) instruction, a request is initiated to direct the MSCP to establish contact with the appropriate station.

When a Modify CD (vary on) instruction is being processed, the MSCP is always requested to contact a station; however, contact with the station cannot always be made at this time. The station cannot be contacted if a switched connection is required and has not been completed or if the station is not powered on. The CD is placed in a vary on pending state until the time when the contact is made. For remote stations, completion of the vary on processing is always done asynchronous to the Modify CD instruction.

Switched network resources are allocated and connection to the station is established as a result of receiving an incoming call or an explicit request (Modify CD (dial) instruction) to dial out is initiated by another process. Upon completion of the switched connection, the MSCP receives the station identification information as sent by the secondary station in response to an Exchange Identification (XID) command. The MSCP validates the identity of the secondary station and completes the vary on processing based on the following checks:

- The station identification matches the XID field in a CD.
- This CD is in the vary on pending or dialing out state.
- The ND representing the line on which the call is made is included in the ND candidate list of this CD.

If any of these checks fail, the connection is broken and the controller description unsuccessful contact event is signaled.

Contact with the station is established and completion of the Modify CD (vary on) instruction always occurs asynchronous to the process executing the Modify CD instruction. When the Modify CD (vary on) instruction is complete, the controller description successful contact event (event class hex 0004, type hex 04, subtype hex 01) is signaled.

Modify Logical Unit Description (MODLUD) Instruction

When the LUD representing a local or direct attached device is varied on, all necessary functions are synchronously completed and the LUD goes to the vary on state before the Modify LUD instruction completes.

Synchronous Data Link Control (SDLC) Logical Unit Description

An LUD representing a logical unit attached to a station on a leased line or a station using the switched network facilities, attains the same states as the CD with which it is associated. For example, the vary on pending state is set until contact with the station is established, at which time the MSCP completes all vary on pending functions. At this time, the logical unit description successful contact event (event class hex 000B, type hex 06, subtype hex 01) is also signaled, provided a positive response to the activate logical unit session with device available indication has been received.

When the Modify LUD (vary on) instruction is being processed, the MSCP sends the activate logical unit session control message to the logical unit to establish an MSCP-to-logical unit session. This session remains active as long as the LUD is in the vary on state. It is the responsibility of the MSCP to terminate this MSCP-to-logical unit session as well as to control the routing of all messages flowing on this session.

When the MSCP-to-logical unit session is established, the logical unit can initiate unsolicited requests to the MSCP. Since the MSCP cannot always service these requests, machine events are used to make the information available to other processes. The events, request data, and the procedure for servicing the request are discussed under *Supervisory Service Events* later in this chapter.

MSCP Operation

The MSCP coordinates all supervisory service requests but does not necessarily provide the requested functions. That is, all supervisory service requests are sent to the MSCP, but the MSCP may in turn route the request on to the ultimate request processor. The MSCP can communicate with all logical units and physical units that are varied on by the MSCP-to-logical unit and MSCP-to-physical unit sessions respectively. However, when requests are initiated by or destined for a process a different means of communication is required.

Requests are signaled to a process by machine events. The Request I/O instruction, with the function field in the SSR set to indicate an MSCP message, is used by the process to route a response or a new request to the MSCP. As before, the MSCP is responsible for routing the message to the appropriate destination.

Supervisory Service Events

Several events are defined which require definite action to be taken by a process in order to continue normal processing.

- Controller Description Manual Intervention Event

  Event Class – Hex 0004
  Type – Hex 06
  Subtype – Hex 01

  This event is signaled by the MSCP when a request is received to establish a switched connection with an SDLC (synchronous data link control) station and the link attachment to be used does not support the autodial feature. The event-related data returned with the event includes a system pointer to the CD representing the station to be called, a system pointer to the ND representing the line on which to place the call, and a 2-byte status field that indicates manual dial operation is required. See Chapter 21 for a detailed description of the event-related data. The process must materialize the CD and ND if it is necessary to retrieve additional information such as the telephone number, ND name, or line number. This information is used to generate and display a message informing the operator that the manual dial operation is to be performed.

  When the connection is made, a Modify ND (manual start data) instruction is issued and the vary on CD processing is completed. If the operator response to the process is that the call cannot be completed, a Modify CD instruction may be issued to abandon the dialing state of the CD (return to vary on pending state).

- Controller Description Successful Contact

  Event Class – Hex 0004
  Type – Hex 04
  Subtype – Hex 01

  This event is signaled to indicate a successful completion of the vary on processing for this controller description for either communications or locally attached controllers. For communications stations the MSCP-to-physcial unit session is active at this time because a positive response was received from the far end secondary station to the activate physical unit request or that the System/38 had a positive response to the far end primary stations activate physical unit request. Event-related data returned with this event consists of a system pointer to the CD and a system pointer to the ND. The data length field is set to 0. The actual exchange identification data received is not provided as event data but is inserted directly into the exchange identification area of the CD object.

- Controller Description Unsuccessful Contact

  Event Class – Hex 0004
  Type – Hex 04
  Subtype – Hex 02

  This event is signaled when the vary on processing failed. The event-related data returned with this event includes a system pointer to the CD, a system pointer to the ND, and up to 66 bytes of data including a 2-byte status code that defines the reason for the failure. Also included in the event-related data, if appropriate, is the exchange identification data received from the failing station. See Chapter 21 for a detailed description of the event-related data.

- Controller Description Loss of Contact

  Event Class – Hex 0004
  Type – Hex 04
  Subtype – Hex 03

  This event is signaled to indicate loss of contact whenever the primary SDLC station has sent a disconnect command to System/38. The event-related data returned with this event includes a system pointer to the CD, a system pointer to the ND (supplied for only CD type 10; otherwise, 0's), and up to 14 bytes of data including a 2-byte status code that defines the reason for the failure. See Chapter 21 for a detailed description of the event-related data.

- Logical Unit Description Unformatted Supervisory Service Request Event

  Event Class – Hex 000B
  Type – Hex 04
  Subtype – Hex 01

  This event is signaled by the MSCP when an unformatted supervisory service request is received on the MSCP-to-logical unit session. The supervisory service request can be initiated by an operator at a work station by pressing the Sys Req key, keying in the optional data, then pressing the Enter key. When this procedure is followed, the MSCP determines that the data is to be routed to a process and then signals the event. The event-related data consists of a system pointer to the LUD that identifies the logical unit that initiated the request. In addition, the data length field of the event-related data indicates the number of bytes of request unit data that follows.

- Logical Unit Description Formatted Supervisory Service Request Event

  Event Class – Hex 000B
  Type – Hex 04
  Subtype – Hex 02

  This event is signaled when the MSCP receives a formatted supervisory service request on the MSCP-to-logical unit session. An example of a formatted request is the request test message sent when the Test Req key is pressed. The event-related data includes a system pointer to the LUD associated with the device that originated the request. The data length field of the event-related data contains the number of bytes of data in the message. The entire message is included in the event-related data just as for the unformatted request.

  When the MSCP receives one of these requests on the MSCP-to-logical unit sessions, the event is signaled and the MSCP sends a positive response to the device that originated the request.

- Logical Unit Description Successful Contact Event

  Event Class – Hex 000B
  Type – Hex 06
  Subtype – Hex 01

  This event is signaled to indicate a successful completion of the vary on process for the logical unit. The MSCP-to-logical unit session is active at this time because a positive response was received to the activate logical unit request and the device available indicator in the logical unit description is set on. For a secondary station, this event is signaled when an activate logical unit request is received from the far end primary station. If the device is not available, this event is not signaled until a request is received by the MSCP to indicate that the device is available. The event-related data consists of a system pointer to the LUD, a system pointer to the ND (if appropriate), and a data length field that is set to 0.

- Logical Unit Description Unsuccessful Contact Event

  Event Class – Hex 000B
  Type – Hex 06
  Subtype – Hex 02

  This event is signaled when the vary on LUD process fails, as in the case where a negative response to an activate logical unit is received by the MSCP. The event-related data includes sense data or exchange identification data and a reason code for the failure. See Chapter 21 for a detailed description of the event-related data.

- Logical Unit Description Device Not Available Event

  Event Class – Hex 000B
  Type – Hex 08
  Subtype – Hex 02

  This event is signaled by the MSCP when logical unit status is received indicating that a previously active device is now unavailable. The event-related data includes a system pointer to the LUD, a data length field of 14 bytes, and a status field containing the SNA logical unit status message.

- Logical Unit Failure Event

  Event Class – Hex 000B
  Type – Hex 08
  Subtype – Hex 03

  This event is signaled by the MSCP whenever System/38 is a secondary station and the far end primary station has indicated that the SSCP to physical unit session is no longer active.

- Network Description SDLC Exchange Identification Failure Event

  Event Class – Hex 000E
  Type – Hex 04
  Subtype – Hex 01

  This event is signaled by the MSCP when switched line connection cannot be established for one of the following reasons:
  - The exchange identification received from a secondary SDLC station does not match the exchange identification of any CD that is in the vary on pending state.
  - A primary SDLC station has sent an Activate Physical Unit Data command to System/38 but the command SSCP identification does not match the SSCP identification of any of the CDs that are in the vary on pending state.

  The event-related data includes a system pointer to the ND, a data length field, and the actual exchange identification (a maximum of 64 bytes) that was received.

- Network Description Line Failure Event

  Event Class – Hex 000E
  Type – Hex 05
  Subtype – Hex 01

  This event is signaled by the MSCP when an unrecoverable line error occurs.

- Network Description SNA Protocol Violation Event

  Event Class – Hex 000E
  Type – Hex 05
  Subtype – Hex 02

  This event is signaled for secondary support situations when the far end primary station has violated the SNA protocol. This event is signaled if the far end station sends information frames before the activate physical unit is processed by System/38.

Request I/O (MSCP) Instruction

The process that handles supervisory service requests communicates with the MSCP by using the Request I/O (MSCP) instruction. The MSCP-to-logical unit session is accessible to the process by using the Request I/O (MSCP) instruction with the object pointer in the SSR addressing the appropriate LUD. The MSCP-to-physical unit session is accessible to the process by using the Request I/O (MSCP) instruction with the system pointer in the SSR addressing the appropriate CD.

The format of the Request I/O (MSCP) instruction and the required operands is described in Chapter 17. For rules that apply to messages that flow on the MSCP-to-logical unit and MSCP-to-physical unit sessions, refer to the *SNA Format and Protocol Reference Manual*, SC30-3112. In addition, the functional specifications for the SNA device supported must be referenced for additional restrictions or requirements. Compliance with the protocol governing data flow on the MSCP sessions is the responsibility of the user.

MSCP Source/Sink Request (SSR): The SSR space object used with a Request I/O (MSCP) instruction is described in Chapter 17. The SSR must contain a system pointer to an LUD, CD, a system pointer to a response queue, and, optionally, a space pointer to the SSD (source/sink data). The function field in the SSR is used to identify the Request I/O instruction as being directed to the MSCP.

The system pointer to an LUD or CD specifies the logical unit or controller to which the message is directed. Contrary to a normal Request I/O instruction, the LUD need not be in the active session state when the Request I/O (MSCP) instruction is executed. The MSCP-to-logical unit and MSCP-to-physical unit sessions are implicitly activated when the CD and LUD are varied on. Therefore, a Request I/O (MSCP) instruction using these sessions is valid anytime the CD or LUD is in the vary on state.

The system pointer to the SSD space is used in the Request I/O (MSCP) instruction the same way as it is used in the normal Request I/O instruction since it provides addressability to the optional input or output data buffers as required by the RD fields in the SSR.

See *Communications Device Management* later in this chapter for a more detailed definition of the format and use of the RD as required in communicating with an SNA device. The use of the SSD space on the MSCP sessions follows the same rules as defined for a Request I/O (MSCP) instruction on the logical unit-to-logical unit session.

MSCP Feedback Record (FBR): An FBR is generated and sent to the response queue specified in the SSR for each Request I/O (MSCP) instruction. The 2-byte error summary field in the FBR contains the appropriate information about the completion status of the Request I/O (MSCP) instruction. The MSCP FBR does not include any device-dependent error status in the optional field.

FBR error summary codes used by the MSCP are as follows:

Error Summary Code	Description	Meaning
Hex 4830	Invalid LUD or CD type	The device represented by this LUD or CD does not support an MSCP-to-logical unit or an MSCP-to-physical unit.
Hex 4831	LUD or CD not varied on	The MSCP-to-logical unit or the MSCP-to-physical unit sessions is not active.

Communications Device Management

This section describes the programming considerations and specific device support requirements for communications source/sink devices. For general information regarding the commands and objects, see the *Functional Concepts Manual.*

Communications device management operates in support of a controller (station) and the logical units (devices) attached to the controller. Device management performs the system network functions necessary to complete a Request I/O instruction. Communications devices supported are as follows:

- 5251 Display Station (secondary device)
- 5252 Dual Display Station (secondary device)
- 5256 Printer (secondary device)
- Work Station Controller (primary device)

All communications devices require an LUD (logical unit description) object to be created for their support. These logical unit descriptions are a type 30 LUD, which requires that both an ND (network description) object and a CD (controller description) object be created to communicate between the LUD and System/38. See Figure 24-1. All information necessary to create LUDs for each specific communications device is supplied in the subsequent sections. The information common to all LUDs, such as name, forward pointers, and backward pointers is defined in Chapter 17.

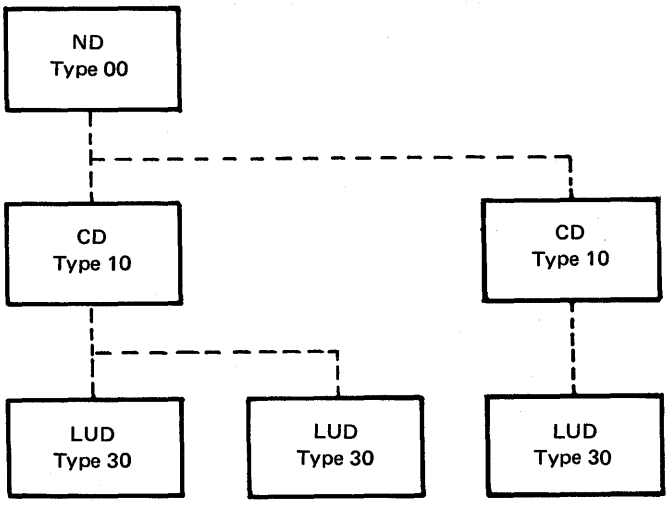


Figure 24-1. Communications Object Configuration

PROGRAMMING CONSIDERATIONS

Communications with a CPU, station, or device is accomplished through system network sessions. For primary device support, an MSCP (machine services control pointer)-to-physical unit path is activated when the station is varied on; an MSCP-to-logical unit path is activated when a logical unit is varied on; and a logical unit-to-logical unit path is activated when an MODLUD (activate) instruction is executed. For secondary station support, an SSCP (system service control point)-to-physical unit path is activated when the ND and CD are varied on and an activate physical unit command is received; an SSCP-to-logical unit path is activated when a logical unit is varied on and an activate logical unit command is received; and a logical unit-to-logical unit path is activated when a MODLUD (activate) instruction is executed.

To manage these system network sessions paths, the device management supports the following instructions.

- Modify Network Description (MODND)
  - Vary On/Off
  - Enable/Disable
  - Manual Answer/Abandon Call
  - Start Data
- Modify Controller Description (MODCD)
  - Vary On/Off
  - Dial/Abandon Connection
- Modify Logical Unit Description (MODLUD)
  - Vary On/Off
  - Activate/De-activate
  - Suspend
  - Quiesce
  - Reset
- Request I/O (REQIO)
  - Functions
    Normal
    MSCP
  - Control
    Normal
    Continue

Prior to using any of these instructions, the user must create the ND describing the line, the CD describing the station(s), and the LUD describing the device(s). This is done by using the following instructions:

- Create Network Descriptor (CRTND)
- Create Controller Descriptor (CRTCD)
- Create Logical Unit Descriptor (CRTLUD)

INSTRUCTIONS

MODND (Vary On/Off)

These instructions are used to activate or de-activate the communications line.

MODND (Enable/Disable)

These instructions are used on switched communications lines to enable the line to accept incoming calls or transmit outgoing calls based on the entry in the switched connection method field in the ND.

MODND (Manual Answer/Abandon Call)

These instructions are used to synchronize the hardware adapter signals with operator actions for completing or discontinuing a switched manual answer connection. Specifically, this instruction indicates that the operator has established the connection and causes the machine to initialize the line.

MODND (Start Data)

This instruction indicates that the operator has manually placed the coupler in data mode, and the line is now ready for data communications.

MODCD (Dial/Abandon Connection)

These instructions are used to allow a station to be dialed manually, automatically, or disconnected.

MODCD (Vary On/Off)

These instructions are used to establish or break the communications path to the physical unit in the station represented by the CD.

MODLUD (Vary On/Off)

For primary device support, this instruction is used to establish or break the communications path from the MSCP to the LU.

MODLUD (Activate/De-activate)

These instructions are used to establish or break the communications path from the machine LU to the CPU or station's LU. MODLUD (activate) is also used to activate the logical unit-to-logical unit path when it is in any one of the three inactive states (suspended, quiesced, or reset).

MODLUD (Quiesce)

When the device management receives this instruction it completes all Request I/O instructions in progress or waiting on the internal LU queue. Upon completing this instruction, the device management comes to a normal completion and all request I/O processing is discontinued until the path is re-activated with a MODLUD (activate) instruction.

The MODLUD (quiesce) instruction is rejected if the unit is already in a terminating error mode upon receipt of the quiesce, if the unit goes into a terminating error mode while trying to complete the MODLUD (quiesce) instruction, or if the quiesce cannot complete within the time-out value specified because of a long running REQIO instruction in process. In these situations, the user must analyze the feedback record that is on the return queue and perform the necessary recovery for the terminating error situation.

MODLUD (Reset)

This instruction causes all Request I/O instructions to be flushed back to the user's return queue, in an orderly fashion, with a feedback status indicating the flushed condition. The number of request descriptors processed is also set in the feedback record. This instruction also places the path in an inactive state, which can be activated with an MODLUD (activate) instruction. All available unsolicited data is relinquished by this instruction. It is possible for the reset to cause partial damage to the LUD. This would happen if the reset were issued with a time-out value specified that is shorter than the time needed to complete a transmit to hardware. For printers or displays using a copy to printer, this time may be as long as 60 seconds. Therefore, the time specified in the reset must be 60 seconds or more to ensure the reset completes normally.

MODLUD (Suspend)

This instruction ensures the user that all REQIO instructions for this logical unit are in the suspended state. This means that all the REQIO instructions that have been started complete to a transmit/receive request descriptor boundary. The REQIO instructions that are completed are placed on the return queue with the appropriate feedback code and number of request descriptors done. Any transmit/receive type of REQIO with only the transmits complete is still on the logical units queue. The suspend may be completed with a reject status if the suspend cannot complete within the time-out value specified because of a long running REQIO instruction in process.

Notes:

1. It is possible for MODLUD (suspend) to complete normally and the logical unit to be placed in terminating error mode. Therefore, it is recommended that the user check the queue for any feedbacks indicating a terminating error condition.
2. If an MODLUD (suspend) instruction is issued when a transmission is in progress that causes input, the input causes the unsolicited data event to be signaled. This is because all I/O operations have been suspended by this command from the viewpoint of the machine but not the communications device.

MODLUD (Suspend, De-activate, and Activate)

This combination of MODLUD instruction operations can be executed to perform a logical unit save type of operation. This atomic set of instructions perform a suspend, reset, de-activate, and activate. Therefore, the user is assured that all REQIO instructions are called back and placed on the user's queue and that the logical unit is in an active path state ready to process more REQIO instructions.

Note: A reset causes any available unsolicited data to be relinquished, as the device management has no way to give the data to the user.

Request I/O Instruction

This instruction is used for two purposes. First, it is used to perform I/O on the various paths.

- Primary Devices
  - MSCP-to-Physical Unit
  - MSCP-to-Logical Unit
  - Logical Unit-to-Logical Unit
- Secondary Stations
  - Physical Unit-to-SSCP
  - Logical Unit-to-SSCP
  - Logical Unit-to-Logical Unit

Second, the Request I/O (continue) instruction is used to return the logical unit to active path state after it has encountered a terminating error. See Chapter 17 of this manual for a detailed description of the Request I/O instruction.

A Request I/O (normal) instruction can contain all the transmit RDs (request descriptors), all the receive RDs, or both transmit or receive RDs. All transmit RDs must occur before the receive RDs.

A Request I/O (receive immediate) instruction is indicated by the first RD being a receive with the immediate bit set on and can contain only receive RDs.

A Request I/O (transmit immediate) instruction contains only one transmit immediate RD and is processed before any SNA normal flow transmit that is on the queue for the LUD.

Note: A REQIO transmit only on the *any* flow defaults to a REQIO transmit only on the *normal* flow.

Device Management Source/Sink Request (SSR) General

The bits in the SSR function field have the following meanings for device management:

Bits 0-3

Hex 8 = Normal Request I/O
Hex 4 = MSCP Request I/O

Bits 4-5

These bits define the SNA flow that all the request descriptors (RD) are to be processed on. The bit values and their meanings are:

Bits	Meaning
00 and 10	The RDs are to flow on the SNA normal flow.
01	The RDs are to flow on the SNA expedited flow.
11	The RDs are to flow on either the SNA normal or SNA expedited flow. **Note:** This bit setting is changed to 10 (normal flow) if the first RD is a transmit.

Bits 6-7 Reserved (binary 0)

Device Management Request Descriptor

This RD is used for communications devices. It contains information for the transmission header and the request/response header. The RD also indicates the length and location of the request unit (RU) in the source/sink data area. Both transmit and receive RDs may be specified in the source/sink data area, but all of the transmit RDs must precede the receive RDs.

The request descriptor field format (in bytes) is:

0	1-5	6-7	8-10	11	12-15
RU Flow Type	Reserved	RU Length	Request Response Header	Reserved	RU Offset

The meaning of the request descriptor format is as follows:

Byte	Meaning
0	Bit 0 of the RU flow type indicates whether this is a transmit or receive RD. If bit 0 has a value of 0, it is a transmit RD. If bit 0 has a value of 1, it is a receive RD. Bit 1 indicates whether the RD has been processed by the machine or is awaiting processing. The device management ensures that this bit is set to 0 when the Request I/O instruction is received. This bit cannot be used by the user to cause device management to skip the RD. The device management always assumes that all the RDs in the REQIO instruction are to be processed. In a terminating or nonterminating error situation, the RD completion count points to the RD on which the error was discovered. Bit 1 would indicate that this RD was processed. The values for bit 1 are: 0 = Not processed 1 = Processed Bits 2-7 for primary devices are reserved and must be binary 0's. Bits 2-3 for secondary stations are reserved and must be binary 0's. Bit 4 for secondary stations 0 = Normal I/O operation 1 = Immediate I/O operations Bits 5-7 for secondary stations must be binary 0's.
1-5	Reserved (binary 0)
6-7	For a transmit RD, this field contains the length of the output RU. A length of 0 indicates no RU is associated with this RD. For a receive RD, before it is processed, this field specifies the space available for the RU in the SSD. After the receive RD is processed, this field contains the length of the received RU. If a group of receive RDs is specified in the SSR, the space available for all of the RUs can be specified in the first RD of the group. The RU length field in the remaining RDs within the group is set to 0. When RDs in the group are processed, the RU length and offset fields are updated to the length and offset of the received RUs. The SSR can have more than one group of receive RDs. The RUs for all RDs in the SSR must be located in the same SSD.

Byte	Meaning
8-10	These 3 bytes contain the information for the request/reponse header (RH). For a transmit RD, the information for the transmitted RH is supplied by the machine in this field. For a receive RD, the information from the received RH is inserted in this field when the RD is processed. All unused bits are reserved and must be set to 0. Byte 8 (request header) Bit 0 = 0 indicates this RU is an SNA request. Bits 1-2 identify which type of RU is being processed. The bits and their meanings are: 00 = Function manager data (FMD) 01 = Network control 10 = Data flow control 11 = Session control Bit 3 is reserved (binary 0). Bit 4 indicates which format is used in the associated RU. The use of formats is session and RU type dependent. One use of this bit is to indicate the presence of an FM header in the RU. Bit 5 0 Indicates a valid request 1 Indicates the first 4 bytes of the RU are sense data for exception requests. Sense data must always be included on a response that indicates an error condition. Bits 6-7 indicate which element of a chain of RUs is being sent. The bit settings are: 00 = Middle of chain 01 = Last of chain 10 = First of chain 11 = Only element in chain

Byte	Meaning
8-10 (continued)	Byte 9 (request header) Bits 0, 2, and 3 are used in combination to specify the type of response required. Bit 0 is always 1 (except isolated pacing response) and bit 2 is always 0 for SNA 0081 implementation. If bit 3 is 0, a definite response must be sent. If bit 3 is 1 (exception response mode), a negative response is sent only if an error is encountered; otherwise, no response is sent. Bit 1 is reserved (binary 0). Bits 4-6 are reserved (binary 0). Bit 7 is used by device management for pacing control. Device management ensures that it contains the correct setting. Byte 10 (request header) Bits 0-1 are used for bracket control. Bracket use is session dependent. Bit 0 set to binary 1 indicates the beginning of a bracket. Bit 1 set to binary 1 indicates the end of a bracket. Bit 2 is a change direction indicator. It is used in a half duplex environment to control the orderly transmission of messages. The bit settings are: 0 = Do not change direction. 1 = Change direction. Bit 3 is reserved (binary 0). Bit 4 is a code selection indicator. It allows the sender to specify that a previously defined alternate code is used in this RU. Bits 5-7 are reserved (binary 0).

Byte	Meaning
8-10 (continued)	

Byte 8 (response header)

Bit 0 = 1 indicates this RU is an SNA response.

Bits 1-2 identify which of the four types of RUs is being processed:

00= Function manager data (FMD)
01= Network control
10= Data flow control
11= Session control

Bit 3 is reserved (binary 0).

Bit 4 indicates which of the format is used in the associated RU. The use of the formats is session and RU type dependent.

Bit 5 indicates whether any sense data is included in the response.

0 = Sense data not included
1 = The first 4 bytes of the RU contain sense data

Bits 6-7 indicate which element of a chain of RUs is being sent. The bit settings are:

00= Middle of chain
01= Last of chain
10= First of chain
11= Only element in chain

Byte	Meaning
8-10 (continued)	

Byte 9 (response header)

Bit	Meaning
0	Always binary 1 (except IPRs)
1	Reserved (binary 0)
2	Always (binary 0)
3	0 = Positive response 1 = Negative response
4-6	Reserved (binary 0)
7	Used by device management for pacing control. Device management ensures that it contains the correct setting.

Byte 10 (response header)

All of byte 10 is reserved (binary 0).

11 Reserved (binary 0)

12-15 This 4-byte field indicates the displacement, in bytes, to the beginning of an RU from the beginning of the SSD space. An RU associated with a specific RD can be located by displacing into the SSD space an amount equivalent to the RU offset. The RU offset for a receive RD is updated by the machine if the RU length in the RD is specified as 0 (that is, the RD is designated as a part of an RD group).

The SSD for the device management RDs contains RUs as defined by the RDs location in the SSR. One RD defines one RU. The location of the RU within the SSD is determined by the RU offset field in the RD.

For transmit RDs, the offset must be supplied to the machine in the RD. For receive RDs, the offset can be supplied for each RD or the offset can be supplied for the first RD of a group and the machine determines the offset for each subsequent RD in the group, based on the length of the received RU. A receive RD is considered a member of a group if its RU length is 0. More than one group of receive RDs can be specified in an SSR but all of the RUs for those RDs must be located in one SSD.

FEEDBACK RECORD

The feedback information for the device management consists of the error summary field and the number of request descriptors processed. All other status fields are not used.

The following table gives the error conditions, the severity code, and recovery procedures. The severity codes are returned in the error summary field.

Note: All feedback codes may have the MSCP bit on. This bit is turned on only if the REQIO function field indicates an REQIO (MSCP). Normal completion for an REQIO (MSCP) is hex 0800.

Error Code	Severity	Error Condition	Recovery Procedure
0000	Normal	None	REQIO completed normally; continue normal processing.
0008	Normal	None	REQIO (continue) completed normally; continue normal processing.
0018	Normal	REQIO Receive Immediate complete with no unsolicited data to be received on indicated SNA flow.	REQIO completed normally; continue normal processing.
0044	Normal	Source/sink data area too small. The current request descriptor has an SSD remaining length field that is smaller than the request unit received from the station.	1. Handle this feedback as if it were an event for unsolicited data. 2. Send device management an REQIO to get the data. 3. If data is not desired, do an MODLUD (Reset, Activate). **Note:** This procedure also recalls all outstanding REQIOs for this logical unit and purges all unsolicited data.
4009	Nonterminating	REQIO partially complete due to the execution of the MODLUD (Reset) command. Some RDs were processed.	Perform the reset cleanup process.
400A	Nonterminating	REQIO complete due to the execution of an MODLUD (Reset) command. No RDs were processed.	Perform the reset cleanup process.
4013	Nonterminating	Negotiable bind response received which was not long enough to contain pacing information.	Do not allow the LU to LU SNA session to go to the bound state.

Error Code	Severity	Error Condition	Recovery Procedure
4024	Nonterminating	REQIO Transmit Immediate completed with a send/receive error.	1. REQIO Transmit Immediate was attempted with a frame that was not valid to be sent at this time. 2. Reissue at a later time or use the nonimmediate REQIO form.
4029	Nonterminating	Pacing error in the bind response. The terminal has attempted to override the hosts pacing parameters.	1. Vary the logical unit off. 2. This is probably a microcode problem in the terminal.
4088	Nonterminating	Invalid RD sequence. A sequence of transmit/receive/transmit is not allowed in an REQIO operation.	Split the REQIO operation into two REQIO operations. The first one does transmit/receive, and the second one does the remaining transmit. **Note:** Device management REQIOs must be one of the following: • Transmit only • Transmit/receive • Receive only
C020	Terminating	LUD failure. The Logical Unit Session was canceled due to some host problem.	1. The logical unit must be varied off only after it has been de-activated. 2. The specific logical unit failure reason can be obtained by monitoring for LUD event hex 000B, type hex 08, subtype hex 01. 3. The error is also recorded in the error log.
C021	Terminating	Line failure. The communications link is broken.	1. The station must be varied off only after each LUD has been varied off. 2. See *Communications Error Recovery Procedures* later in this chapter for the specific line failure. 3. The error is also recorded in the error log.

Error Code	Severity	Error Condition	Recovery Procedure
C022	Terminating	Station or CPU failure. The station has encountered a catastrophic error.	1. The station must be varied off only after each LUD has been varied off. 2. The specific reason may be obtained by monitoring for: CD Event – Hex 4 Type – Hex 5 Subtype – Hex 1 3. The error is also recorded in the error log.
C028	Terminating	An invalid information unit was received.	1. Issue a MODLUD (reset, activate) or a REQIO (continue). 2. Correct the situation.
C045	Terminating	The REQIO instruction has an RD in it that specifies an output RU that is larger than the allowable RU size for this controller.	Set the correct RU size in the RD. Reissue the REQIO instruction after issuing an REQIO (continue) instruction.
C054	Terminating	REQIO for LU-LU session attempted prior to session being bound.	1. Issue a MODLUD (reset, activate) or a REQIO (continue). 2. Prepare to receive a bind session from the host system.
C055	Terminating	REQIO for LU-LU session attempted prior to start data traffic occuring.	1. Issue a MODLUD (reset, activate) or a REQIO (continue). 2. Prepare to receive a start data traffic from the host system.
C084	Terminating	REQIO instruction has an invalid SSD pointer. SSD pointer cannot be 0 if the data length is not 0.	1. Correct either the SSD pointer or the data length field. 2. Issue an REQIO (continue) instruction. 3. Reissue the REQIO instruction.

Note: Terminating error severity requires the user to use a Request I/O (continue) instruction to bring the path back to an active path state. If an MODLUD (reset) instruction has been completed, the user can bring the path back to an active path state by using an MODLUD (activate) instruction.

EVENTS SIGNALED BY COMMUNICATIONS SUPPORT

Network Description Events

SDLC XID Failure Event (000E 04 01): This event is signaled by the MSCP component whenever switched line connection cannot be established to the communications station.

The line connection cannot be established because of one of the following reasons:

- A primary SDLC station has sent ACTPU data to System/38 and the data contains an SSCP ID field that does not match the SSCP ID on any CD that is in the vary on pending state.
- The XID received from a secondary SDLC station does not match the XID of any CD that is in the vary on pending state.

Line Failure Event (000E 05 01, 02): This event is signaled by the MSCP when an unrecoverable line error has occurred.

Subtype 1 is signaled when the line IOM has an unrecoverable hardware error indication.

Subtype 2 is signaled when System/38 is the secondary station and the primary station violates the SNA protocol (such as sending information frames prior to the ACTPU command being processed by System/38).

Controller Description Events

Successful Contact, Unsuccessful Contact, and Loss of Contact Events (0004 04 01, 02, 03): Subtype 1 of this event is signaled to indicate successful completion of the vary on processing for this communications controller (CD type 10). The MSCP-PU session is active at this time, which means that System/38 responded positively to the far end station activate PU request. Event-related data consists of a system pointer to the CD and a system pointer to the ND. The data length field is set to 0. The actual XID data received is not provided as event related data but has been inserted directly into the XID information area of the CD object. Subtype 1 is also signaled for local controllers (CD type 00) directly upon completion of the Modify CD (vary on) instruction. There are no XID exchanges in these cases.

Subtype 2 of this event is signaled when the vary on processing fails. The event-related data includes a system pointer to the CD, a system pointer to the ND, and up to 66 bytes of data including a 2-byte status code which defines the failure reason and, in appropriate cases, the XID data received from the failing station.

Subtype 3 indicates a loss of contact whenever the primary SDLC stations has sent a disconnect command to System/38.

Station Inoperative and Protocol Violation Detected, and SSCP-PU Session Inactive Events (0004 05 01, 02, 03): Subtype 1 (station inoperative) is signaled by communications management when an unrecoverable station error has occurred. The event-related data provides a 2-byte error code, a time stamp, and an operational unit number.

Subtype 2 (protocol violation detected) is signaled by communications management when an SNA path error has been detected. Event-related data includes a 2-byte error code, a time stamp, and the operational unit number.

Subtype 3 is signaled by the MSCP and indicates that System/38, as a secondary station, has received an indication from the primary station that the SSCP-PU session is inactive (for example, DACTPU received).

Controller Description Manual Intervention Event (0004 06 01): This event indicates that a manual dial operation is required.

Logical Unit Description Events

Formatted and Unformatted Supervisory Services Request Events (000B 04 01, 02): Subtype 1 is signaled by the MSCP when an unformatted SNA request is received on the MSCP-LU session. This request can be initiated by an operator at a work station by pressing the Sys Req (System Request) key, keying in the optional data, and then pressing the Enter key. When this message is received, the MSCP determines that the information is to be routed to a process and signals the event. The event-related data consists of a system pointer to the LUD that identifies the logical unit initiating the request. In addition, the data length field indicates the number of bytes of SNA RU data that follows.

Subtype 2 is the machine event signaled when the MSCP receives a formatted SNA request on the MSCP-LU session. One example of a formatted request is the request text message sent when the Test Req (Test Request) key is pressed. Event-related data includes a system pointer to the LUD associated with the device that originated the request. The data length field contains a count of the number of bytes of data in the request test message. The entire message is included in the event-related data just as for the unformatted requests.

When the MSCP receives one of these requests on the MSCP-LU session, the event is signaled and the MSCP sends a positive SNA response to the device that originated the request.

Expedited or Nonexpedited Unsolicited Incoming Messages Event (000B 05 01, 02): This event indicates that unsolicited data is available for this LU. The data can be retrieved only by issuing a Request I/O instruction.

For the subtype 1 event, the data is on the expedited SNA flow.

For the subtype 2 event, the data is on the normal SNA flow.

This event is raised only for the first frame received and anytime there are not enough receive RDs available.

Successful and Unsuccessful Contact Event (000B 06 01, 02): This event is signaled by the MSCP upon completion of the vary on processing for the logical unit.

Subtype 1 of this event is signaled to indicate successful completion of the vary on processing for the logical unit. The MSCP-LU session is active at this time, meaning a positive response was received to the ACTLU request and the device available indicator is set. If the device is not available, this event is not signaled until a request is received that the device is available. Event-related data consists of a system pointer to the LUD and a data length field that is set to 0.

Subtype 2 for this event is signaled when the vary on LUD processing fails, as in the case where a negative response to ACTLU is received. The event-related data includes sense data or XID data and a reason code for the failure.

Logical Unit Description Device Not Available Event (000B 08 02, 03): This event is signaled by the MSCP when logical unit status is received indicating that a previously active device is now unavailable. The event-related data includes a system pointer to the LUD, a data length field of 14 bytes, and a status field containing the SNA logical unit status message.

Subtype 3 is signaled by the MSCP and indicates that System/38, as a secondary station, has received an indication from the primary station that the SSCP-PU session is no longer active (for example, DACTLU received).

Request I/O Complete Event (000B 09 01): This event indicates that the Request I/O instruction asynchronous processing has been completed. This event is signaled only at the request of the user.

Request I/O Response Queue Destroyed Event (000B 0A 01): This event indicates that the return queue specified in the Request I/O instruction has been destroyed.

EXCEPTIONS SIGNALED FOR COMMUNICATION DEVICES

The following table gives the cases where the source/sink resource not available exception (hex 3404) is signaled when a Modify ND instruction is executed.

	Exception Data	
Command	**Generic (hex)**	**Specific**
Vary On	2102	Status code (activate link failures)
Manual Answer	2103	Status code (initialize line failures)
Enable Switched Line	2105	Status code (initialize line failure)

The following table gives the cases where the source/sink resource not available exception (hex 3404) is signaled when a Modify CD instruction is executed.

	Exception Data	
Command	**Generic (hex)**	**Specific**
Dial Out	2204	Hex 0401 (dial out failure because all ND candidates in this CD are in use or not enabled)

The following table gives the cases where the source/sink resource not available exception (hex 3404) is signaled when a Modify LUD instruction is executed.

	Exception Data	
Command	**Generic (hex)**	**Specific**
Vary On	2302	Switched line – Not signaled Nonswitched line – Hex C821 if a line failure occurred – Hex C822 if a station failure occurred – Hex 0800 if an ACTLU failure is indicated by the device
Suspend	2312	Hex 1201 Suspend time-out
Quiesce	2313	Hex 1301 Quiesce time-out Hex 1302 if quiesce is rejected because of a terminating error condition

OBJECT CREATION DATA FOR SUPPORTED DEVICES

This section defines the data needed to create the objects necessary to attach a specific device to the machine.

CD Template Data for 5251 Work Station

The following fields of the controller description template must be initialized as indicated for the 5251 work station.

Field Name	Length	Entry
CD type	Char(2)	Char 10
Unit type	Char(4)	Char 5251
Model number	Char(4)	Char 0002 for 960-character screen Char 0012 for 1920-character screen
Physical address	Char(8)	
• Reserved	Bin(4)	Bin 0
• Station's line address	Bin(2)	
– Reserved	Bin(1)	Bin 0
– Station's line address	Bin(1)	All values except hex 00 and hex FF (supplied by service representative)
• Operational unit number	Bin(2)	Same as in ND if nonswitched line; hex 0000 if switched line
Power control	Char(2)	hex 0000
Station control information	Char(32)	
• XID information	Char(21)	
– XID format	Bit(4)	Bin 0001
– Physical unit type	Bit(4)	Bin 0001
– XID information length	Bit(8)	Hex 14
– Station's block number	Bit(12)	Hex 020
– Specific ID	Bit(20)	User-assigned ID; must be uniquely assigned within the user's network
– Reserved	Char(2)	Bin 0
– Configuration flags	Bit(8)	Hex 00 = SDLC Hex 02 = Loop
– PU characteristics	Bit(8)	Hex B0
– Maximum length received	Bin(2)	Hex 0105
– Reserved	Char(4)	Bin 0
– Frames limit	Bit(8)	Hex 0003 = Without remote cluster feature Hex 0007 = With remote cluster feature
– Reserved	Char(4)	Bin 0

Field Name	Length	Entry
• Station definition	Char(1)	
– SDLC link	Bit(2)	Bin 10
– Link type	Bit(1)	Bin 0 = Nonswitched Bin 1 = Switched
– Station's role	Bit(1)	Bin 0 = Secondary
– Switched network backup	Bit(1)	Bin 0 = No Bin 1 = Yes
– Data rate selection feature	Bit(1)	Bin 0 = No Bin 1 = Yes
• Reserved	Char(2)	Bin 0
• Path information unit type	Bin(2)	Hex 0003
• Reserved	Char(6)	Bin 0
Selected modes	Char(2)	Hex 0000 = Leased Hex 1000 = Switched network backup mode **Note:** Switched Network Backup feature must be specified in the station definition field and in the corresponding ND.
Delayed contact control	Char(2)	Hex 0000 = Not in effect Hex 0100 = In effect
Activate physical unit data	Char(16)	Bin 0
Dial digits (telephone number if switched station)	Char(32)	
• Reserved	Char(6)	Bin 0
• Length	Bin(2)	Number of dial digits
• Digits	Char(16)	Actual telephone number–May contain only digits 0 through 9 or separator (SEP) character Hex 7D or end-of-number (EON) character Hex 5C. These two special characters are represented by an apostrophe (') or an asterisk (*) key on an EBCDIC keyboard.
• Reserved	Char(8)	Bin 0
Specific characteristics	Bin(2)	Hex 0000
XID information area		
• Length of XID data	Bin(2)	Hex 0014
• XID data	Char(20)	Bin 0
Unit-specific data area	Char(2)	
• Length of unit-specific area	Bin(2)	Hex 0002
• Length of modifiable area	Bin(2)	Hex 0002
Unit-specific contents area	Bin(2)	Hex 0000

Logical Unit Description Template Data for 5251 Display

The following fields of the logical unit description template must be initialized as indicated for the 5251 Display Station.

Field Name	Length	Entry
LUD type	Char(2)	Char 30
Device type	Char(4)	Char 5251
Model number	Char(4)	Char 0001 for 960-character screen Char 0011 for 1920-character screen
Physical address	Char(8)	
• Reserved	Bin(2)	Bin 0
• Logical unit address	Bin(2)	Hex 00 for first display Hex 01 for second display on 5252 Hex 02-Hex 09 for displays attached through the remote cluster feature
• Station line address	Bin(2)	Same address as in CD to which this LUD attaches
• Operational unit number	Bin(2)	Same as in CD and ND if nonswitched line; hex 0000 if switched line
Power control	Char(2)	Hex 0000
Session information	Char(20)	Bin 0
• Inbound pacing	Bin(2)	Hex 0000
• Outbound pacing	Bin(2)	Hex 0000
• Request unit buffer size	Bin(2)	Hex 0100
• ACTLU	Char(1)	Hex 01
• ACTLU parameters	Char(3)	Hex 0D0101
• ACTLU response	Char(8)	Bin 0
Load/dump indicator	Char(4)	Bin 0
Specific characteristics	Bin(2)	Hex 0000
Retry value sets	Bin(2)	Hex 0000
Error threshold sets	Bin(2)	Hex 0000
Device-specific data area	Char(2)	
• Length of device-specific area	Bin(2)	Hex 0002
• Length of modifiable area	Bin(2)	Hex 0002
Device-specific contents	Bin(2)	Hex 0000

Logical Unit Description Template Data for the 5252 Display

The following fields of the logical unit description template must be initialized as indicated for the 5252 Display Station. A separate LUD is needed for each display screen.

Field Name	Length	Entry
LUD type	Char(2)	Char 30
Device type	Char(4)	Char 5252
Model number	Char(4)	Char 0001 for 960-character screen
Physical address	Char(8)	
• Reserved	Bin(2)	Hex 0000
• Logical unit address	Bin(2)	Hex 02-Hex 09 for displays attached through the remote cluster feature
• Station's line address	Bin(2)	Same address as in CD to which this LUD is attached
• Operational unit number	Bin(2)	Same as in CD and ND if nonswitched line; hex 0000 if switched line
Power control	Char(2)	Hex 0000
Session information	Char(20)	
• Inbound pacing	Bin(2)	Hex 0000
• Outbound pacing	Bin(2)	Hex 0000
• Request unit buffer size	Bin(2)	Hex 0100
• ACTLU	Char(1)	Hex 01
• ACTLU parameters	Char(3)	Hex 0D0101
• ACTLU response	Char(8)	Bin 0
Load/dump indicator	Char(4)	Bin 0
Specific characteristics	Bin(2)	Hex 0000
Retry value sets	Bin(2)	Hex 0000
Error threshold sets	Bin(2)	Hex 0000
Device-specific data area	Char(2)	
• Length of device-specific area	Bin(2)	Hex 0002
• Length of modifiable area	Bin(2)	Hex 0002
Device-specific contents	Bin(2)	Hex 0000

Logical Unit Description Template Data for the 5256 Printer

The following fields of the logical unit description template must be initialized as indicated for the 5256 Printer.

Field Name	Length	Entry
LUD type	Char(2)	Char 30
Device type	Char(4)	Char 5256
Model number	Char(4)	Char 0001 for 40 characters per second Char 0002 for 80 characters per second Char 0003 for 120 characters per second
Physical address	Char(8)	
• Reserved	Bin(2)	Hex 0000
• Logical unit address	Bin(2)	Hex 02-Hex 09 for printers attached through the remote cluster feature
• Station's line address	Bin(1)	Same address as in CD to which this LUD is attached
• Operational unit number	Bin(2)	Same as in ND if nonswitched line; hex 0000 if switched line
Power control	Char(2)	Hex 0000
Session information	Char(20)	
• Inbound pacing	Bin(2)	Hex 0000
• Outbound pacing	Bin(2)	Hex 0003
• Request unit buffer size	Bin(2)	Hex 0100
• ACTLU	Char(1)	Hex 01
• ACTLU parameters	Char(3)	Hex 0D0101
• ACTLU response return area	Char(8)	Bin 0
Load/dump indicator	Char(4)	Bin 0
Specific characteristics	Bin(2)	Hex 0000
Retry value sets	Bin(2)	Hex 0000
Error threshold sets	Bin(2)	Hex 0000
Device-specific data area	Char(2)	
• Length of device-specific area	Bin(2)	Hex 0002
• Length of modifiable area	Bin(2)	Hex 0002
Device-specific contents	Bin(2)	Hex 0000

CD Template Data When System/38 is Attached as a Secondary Station

The following fields of the controller description template, that represent the SDLC primary station, must be initialized as indicated when the System/38 is attached as a secondary station.

Field Name	Length	Entry
CD type	Char(2)	Char 10
Unit type	Char(4)	PU2
Model number	Char(4)	Char 0000
Physical address	Char(8)	
• Reserved	Bin(4)	Bin 0
• Station's line address	Bin(2)	
– Reserved	Bin(2)	Bin 0
• Operational unit number	Bin(2)	
– Operational unit number from the corresponding ND if a nonswitched line	Bin (2)	
– Zero if a switched line	Bin (2)	Hex 0000
Power control	Char(2)	Hex 0000
Station control information	Char(32)	
• XID information	Char(21)	
– XID format	Bit(4)	Bin 0001
– Physical unit type	Bit(4)	Bin 0001
– XID information length	Bit(8)	Hex 14
– Station's block number	Bit(12)	Hex 000
– Specific ID	Bit(20)	User-assigned ID
– Reserved	Char(2)	Bin 0
– Configuration flags	Bit(8)	Hex 00
– PU characteristics	Bit(8)	Hex B0
– Maximum length received Hex 0109 = 256-byte RUs with 9-byte TH + RH Hex 0209 = 512-byte RUs with 9-byte TH + RH	Bin(2)	
– Reserved	Char(4)	Bin 0
– Frames limit	Bit(8)	Hex 07
– Reserved	Char(4)	Bin 0

Field Name	Length	Entry
• Station definition	Char(1)	
– SDLC link	Bit(2)	Bin 10
– Link type	Bit(1)	Bin 0 = Nonswitched Bin 1 = Switched
– Station role	Bit(1)	Bin 1 = Primary station
– Switched network backup	Bit(1)	Bin 0 = No Bin 1 = Yes
– Data rate selection	Bit(1)	Bin 0 = No Bin 1 = Yes
– Reserved	Bit (2)	Bin 00
• Reserved	Char(2)	Bin 0
• Path information unit type	Hex 0002	
• Reserved	Char(6)	Bin 0
Selected modes	Char(2)	Hex 0000 = Nonswitched Hex 0001 = Switched network backup mode
Delayed contact control	Char(2)	Hex 0000 = Not in effect
Activate physical unit data	Char(16)	
• Activate physical unit data required	Char (1)	Hex 00
• Request code	Char (1)	Hex 11
• Type activation	Char (1)	Hex 01 = Cold start Hex 02 = Error recovery procedure
• Profiles	Char(1)	Hex 01
• SSCP identification	Char (6)	
– PU type 2 installation	Char (1)	Hex 05
– SSCP identification as defined for host (primary) system	Char (5)	SSCP identification
• Reserved	Char (6)	Bin 0
Dial digits	Char(32)	
• Reserved	Char(6)	Bin 0
• Length	Bin(2)	Number of dial digits
• Digits	Char(16)	Actual telephone number may contain only digits 0 through 9 and separator character hex 7D and end-of-number character hex 5C. These special characters are represented by the apostrophe and the asterisk keys on an EBCDIC keyboard.

Field Name	Length	Entry
• Reserved	Char(8) Bin 0	
Specific characteristics	Bin(2)	Hex 0000
XID information area		
• Length of XID data	Bin(2)	Hex 0014
• XID data	Char(20)	Bin 0
Unit-specific data area	Char(2)	
• Length of unit-specific area	Bin(2)	Hex 0002
• Length of modifiable area	Bin(2)	Hex 0002
Unit-specific contents area	Bin(2)	Hex 0000

Logical Unit Description Template Data When System/38 Is Attached as a Secondary Station

The following fields of the logical unit description, that represent the SDLC primary station, must be initialized as indicated when System/38 is attached as a secondary station.

Field Name	Length	Entry
LUD type	Char(2)	Char 30
Device type	Char(4)	PLU 1
Model number	Char(4)	Hex 0000
Physical address	Char(8)	
• Reserved	Bin(2)	Hex 0000
• Logical unit address	Bin(2)	Hex 0000 through Hex 00FF
• Station's line address	Bin(1)	Hex 0000
• Operational unit number	Bin(2)	Same as in ND if nonswitched line; hex 0000 if switched line
Power control	Char(2)	Hex 0000
Session information	Char(20)	
• Inbound pacing	Bin(2)	Hex 0000
• Outbound pacing	Bin(2)	Hex 0000
• Request unit buffer size	Bin(2)	Hex 0100 for 256-byte RU or Hex 0200 for 512-byte RU
• ACTLU	Char(1)	Hex 00
• ACTLU parameters	Char(3)	Hex 000000
• ACTLU response return area	Char(8)	Bin 0
Load/dump indicator	Char(4)	Bin 0
Specific characteristics	Bin(2)	Hex 0000
Retry value sets	Bin(2)	Hex 0000
Error threshold sets	Bin(2)	Hex 0000
Device-specific data area	Char(2)	
• Length of device-specific area	Bin(2)	Hex 0002
• Length of modifiable area	Bin(2)	Hex 0002
Device-specific contents	Bin(2)	Hex 0000

COMMUNICATIONS LINES SPECIALIZATION

Communications line specialization is the means by which the proper microcode components are associated with the protocol and hardware assigned to the line.

Since the microcode components are designed to support various protocols and hardware based upon parameters maintained in the ND, it is essential that the complete line configuration be carefully analyzed so the parameters match the desired support. For example, failure to correctly specify modem characteristics such as nonclocked modem, NRZI, or others results in a poorly functioning or nonfunctioning line.

The following shows the field in the ND template used in creating an ND and the specific entries in these fields for each type of communications line.

ND Template Elements	SDLC Primary/Secondary Line	SDLC Loop
ND Definition Data		
– ND type	Char 00	Char 00
– Maximum number of CDs attached to this ND	10	64
Physical Definition Data		
– Physical address		
Bytes 0-5	Hex 000000000000	Hex 000000000000
Bytes 6-7	Hex 0020 = First line (IOC-1)	Hex 0020 = First line (IOC-1)
(OU Number)	Hex 0021 = Second line (IOC-1)	Hex 0021 = Second line (IOC-1)
	Hex 0022 = Third line (IOC-1)	Hex 0022 = Third line (IOC-1)
	Hex 0023 = Fourth line (IOC-1)	Hex 0023 = Fourth line (IOC-1)
	Hex 0060 = First line (IOC-2)	Hex 0060 = First line (IOC-2)
	Hex 0061 = Second line (IOC-2)	Hex 0061 = Second line (IOC-2)
	Hex 0062 = Third line (IOC-2)	Hex 0062 = Third line (IOC-2)
	Hex 0063 = Fourth line (IOC-2)	Hex 0063 = Fourth line (IOC-2)
Line Definition Data		
– Line discipline	SDLC	SDLC
– Switched network	*	No
– Switched network backup	*	No
– Data rate select	*	No
– Role	Primary/Secondary	Primary
– NRZI	*	No
– Nonclocked modem	*	No

ND Template Elements	SDLC Primary/Secondary Line	SDLC Loop
– OEM modem	*	No
– Four-wire line(s)	*	No
– Multipoint	*	No
– Loop	No	Yes
– Autodial feature	*	No
– Autoanswer feature	*	No
– Autoanswer sequence	*	No
– Answer tone generation	*	No
– Marks/spaces for answer tone	*	No
– Special answer tone	*	No
– Data communications equipment	*	None
– Line speed	12, 24, 48, 72, or 96*	192
– Secondary address	*	Hex 0000
Selectable Modes Data		
– Switched network backup	*	0
– Selected rate	*	0
– Switched connect method	*	0
– Autodial mode	*	0
– Autoanswer mode	*	0
– Switched secondary inactive time-out	*	0
Communications Subsystems		
Parameters Data		
– Data terminal ready delay	0-15*	0*
– Idle state detection timer	0-255*	0-255*
– Nonproductive receive timer	0-255*	0-255*
Specific Characteristics		
– Specific characteristic length	Hex 0000	Hex 0000
Retry Value Sets		
– Retry value length	144 Bytes	96 Bytes
– Error type	See Figure 24-2	See Figure 24-3
– Error retry value	See Figure 24-2	See Figure 24-3
Line-Specific Contents		
– Line-specific contents length	Hex 0002	Hex 0002
– Line-specific contents modifiable length	Hex 0002	Hex 0002
– Line-specific contents	Hex 0000	Hex 0000

*See *Create Network Description (CRTND)* in Chapter 17 for a description of the requirements for the ND creation.

Type Code (hex)	Error Description	Recommended Retry Value
E324	Clear to send inactive	7
E2E4	Clear to send inactive before request to send	7
E526	Data line occupied	1
E566	Present next digit inactive after call request set (Autodial feature)	1
E586	Present next digit active after digit present set (Autodial feature)	1
E5E6	Present next digit inactive after digit present resent (Autodial feature)	1
E606	Distant station connected inactive	1
E546	Present next digit inactive after call request set – ACR (abandon call and retry) (Autodial feature)	1
E5A6	Present next digit inactive after digit present set – ACR (Autodial feature)	1
E5C6	Present next digit inactive after digit present reset – ACR (Autodial feature)	1
E626	Distant station connected inactive – ACR	1
E2C4	Request to send inactive	Same as for E324
E506	General autodial error	1
F4C2	Adapter overrun/underrun	7
F507	Unrecognizable SDLC control field	7
F517	SDLC sequence number error (Transmit)	7
F527	SDLC sequence number error (Receive)	7
F805	CRC error	7
F7C5	Frame abort detected (Pattern binary 01111111)	7
F660	CPU buffer overflow (on input)	7
F705	Idle state detected	7
F7A5	Nonproductive receive time-out	7
F441	Data overrun (Receive)	7
F481	Data overrun (Transmit)	7

Note: The allowable retry value entry is 0-21.

Figure 24-2. SDLC Primary/Secondary Error Types and Retry Values

Type Code (hex)	Error Description	Recommended Retry Value
E324	Clear to send inactive	7
E2E4	Clear to send active before request to send	7
E2C4	Request to send inactive	Same as for E324
E685	Beacon detected	7
F660	CPU buffer overflow	7
F705	Idle state detected	7
F7A5	Nonproductive receive time-out	7
F4C2	Adapter overrun/underrun	7
F441	Data overrun	7
F507	Unrecognizable SDLC control field	7
F517	SDLC sequence number error (Transmit)	7
F527	SDLC sequence number error (Receive)	7
F805	CRC error	7
F7C5	Frame abort detected (Pattern binary 01111111)	7
F481	Data underrun	7
F537	Station not responding	7

Figure 24-3. SDLC Loop Error Types and Retry Values

It is suggested that the line specialization interface to the user categorize the preceding errors and allow individual error retry specification only at the user's request. Errors can be categorized as follows:

Error Category	Type Codes (hex)
Dial errors	E526, E566, E586, E5E6, E606, E546, E5A6, E5C6, E626, E506
Telecommunications errors	E324, E2E4, E2C4, F4C2, F507, F517, F527, F805, F7C7, F660 F705, F7A5, F441, F481

COMMUNICATIONS ERROR RECOVERY PROCEDURES

Device-specific error codes identifying the type of communications errors are returned in the event-related data when the following events are signaled:

- Event class name is controller description

  Event class number is hex 0004

  Event type and subtype is hex 0501 (controller description lost contact)

- Event class name is network description

  Event class number is hex 000E

  Event type and subtype is hex 0501 (network description line failure)

When either of these events is received, the station represented by the controller description or the communications line represented by the network description becomes inoperative. In addition to these events, an error log entry that contains more detailed information is recorded.

Device-specific error codes identifying the type of communications errors are also returned in the exception data for a source/sink resource not available exception (hex 3404).

For controller descriptions, the source/sink resource not available exception can occur on Modify CD instructions that attempt to change the state of the CD to the vary on, or dial out state.

For network descriptions, the source/sink resource not available exception can occur on Modify ND instructions that attempt to change the state of the ND to the vary on, manual answer, manual start data, or enable switched line states.

Since the exception device-specific error codes for the Modify CD and Modify ND instructions are a subset of those supplied as event data for the CD and ND failure events, the recovery action should be based on the event data rather than the exception data.

The following chart gives the device-specific error codes, error name and descriptions, and recovery procedures for the errors.

Device-Specific Error Code	Error Name and Description	Error Recovery Procedure Number
8049	Operational program error • The operational unit task microcode set the program error bit in the BSTAT of an operation request element on.	2
808B	Channel error • The channel IOM detected a channel error.	1
80AC	IOC command reject • IOC microcode detected a command reject condition.	1
8103	Post event error • The IOC microcode detected an unrecoverable hardware error.	1
8107	Invalid BSTAT • The IOC returned an invalid BSTAT encoding to the IOM.	1
8137	Read sense failure • A Read Sense command issued to the IOC failed to return in 15 seconds.	1
81C3	IOC or CA parity error, addressing modem port • An IOC parity error on the DBI is detected during an I/O Read command. or A channel address parity error on the address bus out of the IOC is detected during an I/O Read or I/O Write command. or A channel address parity error on the data bus out of the IOC is detected during an I/O Write command.	1
81E3	IOC or channel address parity error, addressing the autocall unit • See error code 81C3.	1
8204	Data terminal ready line inactive • Data terminal ready line is not active when expected.	1
8224	Data set ready line inactive • Data set ready line inactive for 7 seconds.	4
8244	Invalid autoanswer request • Data set ready line is active at the start of the autoanswer operation.	4
8264	Data set ready line time-out during autoanswer sequence • Ring indicate line became active but data set ready line did not become active within 30 seconds.	4

Device-Specific Error Code	Error Name and Description	Error Recovery Procedure Number
8284	Data set ready and ring indicate sequence error during autoanswer sequence • Data set ready line became active while ring indicate line became inactive. or Ring indicate line could not be reset in 7 seconds.	4
8360	Line not initialized • Read Data or Enable Switch Connection commands pending for 100 seconds without the line being initialized. or Write Data, Autopoll, Auto Optional Response Poll, or Auto Dial command occurred during a transmit operational unit task, before the Initialize command was given. or The I/O controller entered a not initialized state because of a violation of protocol.	2
8380	Switched line disconnect time-out • The adapter is not currently processing a command and has not received a command from the I/O manager for 30 seconds. The switched line has been disconnected.	2
8401	I/O controller channel disconnect • The IOC controller could not transfer data because of a channel disconnect condition.	3
84E2	Adapter interrupt request time-out • A byte transfer request failed because of a hardware error.	1
9207	SDLC command reject • Command reject received from the secondary station. – Secondary station buffers may have overflowed. – Secondary station may have detected out of range sequence numbers or a nonsupported command.	15
9217	SDLC disconnect mode received • The far end station has entered disconnect mode when it should be in normal mode.	13
9227	SDLC out of range sequence numbers • The far end station is sending out of range sequence numbers.	11
9237	Invalid SDLC address • The polled station responded with the wrong address.	11

Device-Specific Error Code	Error Name and Description	Error Recovery Procedure Number
96A5	Request-to-transmit line time-out • System/38 attempted to transmit data after line turnaround, and the far end station continued transmitting for 10 seconds.	11
A200	Unexpected SDLC set-normal-response-mode • An SDLC set-normal-response-mode command was received when the secondary station was already in normal-response-mode.	11
A210	Unexpected SDLC disconnect received • An SDLC disconnect was received by the secondary station IOM prior to receiving a de-activate physical unit request from the primary station. The secondary station will: – Perform an 'abnormal de-activate physical unit' – Send a message to the MSCP indicating 'abnormal disconnect received' (an abnormal DACTPU was performed)/ – Log the error	
A220	SDLC Command Reject • One or more of the following conditions causes command reject: – Invalid SDLC control field – Out of range SDLC sequence numbers – I-frame with control field that does not allow information data • An SDLC command reject has been sent to the primary station.	11
A230	Unexpected de-activate physical unit • A DACTPU command was received prior to receiving the required number of DACTLU commands. The secondary station IOM will: – Perform an abnormal DACTPU – Send a message to the MSCP indicating that an abnormal DACTPU was performed – Log the error	
A250	Unexpected ACTPU received • A second ACTPU was received prior to receiving a DACTPU command. The secondary station IOM will: – Perform a second ACTPU function – Send a message to the MSCP indicating that a second ACTPU command was received and requesting that the MSCP validate the SSCP ID – Log the error	

Device-Specific Error Code	Error Name and Description	Error Recovery Procedure Number
A380	Switched line disconnect time-out • The adapter has not received an input frame on the line for the duration of the nonproductive receive time-out value. or The adapter has not received a command from the IOM for 30 seconds when it is not currently working on a command. • The switched line is disconnected to save tariff charges.	9
E2C4	Request-to-send line inactive • Request-to-send line became inactive when not expected.	1
E2E4	Clear-to-send line active at initialization • Clear-to-send line active for 7 seconds when initialization was attempted.	4
E324	Clear-to-send line inactive • Clear-to-send line did not become active within 3.4 seconds. or Clear-to-send line became inactive during transmission.	4
E506	Reset autocall unit time-out • Any of the lines, present next digit, distant station connected, or abandon call/retry are active for 3.4 seconds at the start of Autocall command execution.	6
E526	Data line occupied time-out • An attempt to reset the autocall unit failed and the data line occupied line signal remained active for 3.4 seconds.	6
E546	Abandon call/retry line active during a call request to the autocall unit	6
E566	Present next digit line inactive while abandon call/retry line is active	6
E586	Digit present line time-out • Present next digit line continually active.	6
E5A6	Present next digit line active after digit present line is set • Present next digit line active after digit present line is set during abandon call/retry.	6
E5C6	Digit present line time-out on first digit • No present next digit line received during the initial call request.	6

Device-Specific Error Code	Error Name and Description	Error Recovery Procedure Number
E5E6	Present next digit line time-out • One or more digits transferred, waiting for present next digit line and digit present reset line to become active.	6
E606	Distant station connected line time-out • No distant station connected line received.	16
E626	Distant station connected line inactive, abandon call/retry line active • Abandon call/retry line signal received, waiting for distant station connected line to become active.	5
E685	Beacon detected, loop	7
F441	Data overrun or channel disconnect • The I/O controller is unable to transfer data to main storage fast enough when receiving data from the secondary station.	3
F481	Data underrun or channel disconnect • Data from main storage to the I/O controller was not available to transmit.	3
F4C2	Adapter overrun or an adapter underrun • The channel adapter card was not serviced by the I/O controller microcode within 1-byte time. or On a receive operation, data is arriving too fast for the adapter (adapter overrun). or On a transmit operation, the adapter cannot send data fast enough (adapter underrun).	10
F507	Unrecognizable SDLC control field • The far end station sent an invalid SDLC control field (primary mode only). This error is reported as error code 9207 for secondary mode.	11
F517	SDLC sequence error, transmit failure • The sequence numbers indicate that frames were lost on transmit. The number of frames received by the far end station does not coincide with the number of frames transmitted by System/38.	9

Device-Specific Error Code	Error Name and Description	Error Recovery Procedure Number
F527	SDLC sequence error, receive failure • The sequence numbers indicate that frames were lost on receive. The number of frames received by System/38 does not coincide with the number of frames transmitted by the far end station.	9
F537	No response, loop • A station is not responding when given an optional response poll, and a response is due.	14
F660	Main storage buffer overflow • The number of bytes of data received before the ending sequence occurred exceeded the value in bytes 2-3 of the Read Data command.	2
F705	Idle state detected	8
F7A5	Nonproductive receive time-out	9
F7C5	Frame aborted • The SDLC abort sequence was received.	11
F805	Redundancy check error (CRC) • There is a mismatch between the System/38-computed CRC and the CRC sent by the far end station.	12

Error Recovery Procedures

Error Recovery Procedure Number	Error Recovery Procedures
1	Hardware or hardware microcode problem Contact your service representative.
2	Programming problem Contact your service representative.
3	Resources may be overcommitted with insufficient main storage or excessive paging. Contact your service representative.
4	Local modem or autoanswer problem 1. If a stand-alone (external) modem, verify that power is turned on and that all switches are in the correct position. 2. If power is on and all switches are in the correct position, then contact your service representative. 3. If not a stand-alone modem, contact your service representative.
5	Far end, local coupler, or autocall unit problem Contact your service representative.

Error Recovery Procedure Number	Error Recovery Procedures
6	Autocall unit problem Verify that the autocall unit power is turned on, that autocall unit and modem switches are in the correct position, and that the cable to the autocall unit is connected. If all these conditions are met, contact your service representative.
7	Customer loop problem or loop station problem Execute the communications problem determination procedure. The problem is before the station that is generating the Beacon message or at the generating station.
8	Idle state detected on the line 1. Verify that the line is connected. 2. Verify that the far end station and the secondary modem is powered on. 3. Verify that the far end station address agrees with the address in the configuration area of the CD. 4. Verify that the idle state detection timer value in the ND object provides sufficient time to propagate this particular network. 5. If all of these conditions are met, contact your service representative.
9	Line, far end hardware, or local hardware problem Contact your service representative.
10	Adapter overrun or underrun 1. Check the aggregate data rate for the adapter. 2. Check the line speed selector on both ends. Although this error will not occur if the speeds do not match, both speeds could be set too high. 3. Contact your service representative.
11	Far end problem 1. Contact the far end operator and report the symptoms of the problem. 2. Contact your service representative.
12	Cyclic redundancy check error If switched: 1. Retry the switched connection. 2. Run the SDLC link test utility. If remote or leased: 1. Run the SDLC link test utility.
13	Far end has entered disconnect mode 1. Contact the far end operator and check for any possible error conditions. 2. Contact your service representative.

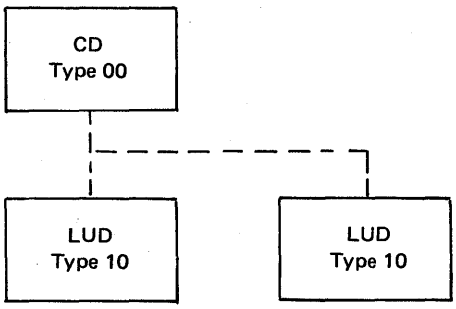
Error Recovery Procedure Number	Error Recovery Procedures
14	Loop station not responding 1. Station may have to be powered on. 2. Station may have to be plugged into the loop. 3. The station address may not agree with the address provided by the user. 4. The station switches may be set wrong.
15	SDLC command reject 1. Reset the secondary station (power off then power on). 2. Contact your service representative.
16	Far end not responding to autodial 1. Check for correct telephone number. 2. Check for busy line. 3. Check far end operating status. 4. Contact your service representative.

Work Station Controller Management (Locally Attached)

The work station controller management operates in support of a work station controller and the devices attached to the work station controller. The work station controller management performs the system network functions necessary to complete a Request I/O instruction and to manage data flow to and from the devices. The work station controller management supports 5251 Displays, 5252 Displays, and the 5256 Printer.

All work station controller devices require a LUD to be created for their support. These LUDs are type 10 LUDs which require a controller description (CD) so they can be attached to the machine.

Only one work station controller object configuration exists for attachment of work station controller LUD objects to the machine. See the following chart.



PROGRAMMING CONSIDERATIONS

Communication with a work station controller or device is accomplished through system network paths. An MSCP-to-physical unit path is activated when the work station controller is varied on, an MSCP-to-logical unit path is activated when the logical unit is varied on, and a logical unit-to-logical unit path is activated when a Modify LUD (activate) instruction is executed. To manage these system network paths the device management supports the following instructions.

- Modify Controller Description (MODCD)
  - Vary on/off

- Modify Logical Unit Description (MODLUD)
  - Vary on/off
  - Activate/de-activate
  - Suspend
  - Quiesce
  - Reset
  - Device-specific contents

- Request I/O (REQIO)
  - Functions
    Normal
    MSCP
  - Control
    Normal
    Continue

Before using any of these commands, the user must create the controller description object describing the work station controller and the LUDs describing the device(s). This is done by using the following instructions:

- Create Controller Description (CRTCD)

- Create Logical Unit Description (CRTLUD)

See *Object Creation Data for Supported Devices* later in this chapter for the template data needed to create the CD and LUD(s) for the work station controller and supported device(s).

INSTRUCTIONS

MODCD (Vary On/Off)

This instruction is used to establish or break the communications path to the physical unit in the work station controller represented by the CD.

MODLUD (Vary On/Off)

This instruction is used to establish or break the communications path from the MSCP-to-logical unit. It also performs preliminary setup for the logical unit-to-logical unit path.

MODLUD (Activate/De-activate)

This instruction is used to establish or break the communications path from the machine logical unit to the work station controller's logical unit. MODLUD (activate) is also used to activate the logical unit-to-logical unit path when the LUD is in any one of the three inactive states (suspended, quiesced, or reset).

MODLUD (Quiesce)

Once the work station controller management receives this request, it completes all REQIOs that are in progress or waiting on the internal LU queue. Upon completion of this command, the device management comes to a normal completion and all REQIO processing is discontinued until the path is re-activated with an MODLUD (activate) instruction.

The quiesce is completed with a rejected status in three situations. The first is that the unit is already in a terminating error mode when it receives the quiesce. The second is that the unit goes into terminating error mode while it is trying to complete the quiesce. The third is that the quiesce cannot be completed within the time-out value specified due to a long running REQIO in process. In these situations, the user must analyze the feedback record(s) that is on his return queue and perform the necessary recovery for the terminating error situation.

Note: If this instruction is issued when an REQIO with a transmit containing a Read From Device instruction exists, the work station controller management is unaware of it. Therefore, the completion of quiesce is dependent on the operator pressing the Enter key. If the Read From Device instruction is not completed within the time-out interval specified in the Modify LUD instruction, a partial damage exception is signaled and the LUD must be varied off to recover.

MODLUD (Reset)

This instruction causes all REQIOs to be returned to the user's return queue, in an orderly fashion, with a feedback status indicating that the REQIOs were returned to the user's return queue. The number of request descriptors processed is also set in the feedback record. This instruction also places the path in an inactive state, which can be reactivated with a MODLUD (activate) instruction. All available unsolicited data is discarded by this instruction.

The reset can cause partial damage to the LUD if the reset is issued with a time-out value specified that is shorter than the time needed to complete a transmit to hardware. For printers or displays using a copy to printer, this time may be as long as 60 seconds. Therefore, the time specified in the reset must be 60 seconds or more to ensure the reset will be completed normally.

MODLUD (Suspend)

This instruction ensures the user that all REQIOs for this logical unit are in the suspended state. This means that all the REQIOs that have been started are completed to a transmit/receive request descriptor boundary. The REQIOs that are completed are placed on the return queue with the appropriate feedback code and number of RDs done. Any transmit/receive REQIO with only the transmits complete is still on the logical unit's queue. The suspend may be completed with a reject status if the suspend cannot complete within the time-out value specified because of a long running REQIO in process.

Note: An MODLUD (suspend) may complete normally, and the logical unit may be placed in terminating error mode. Therefore, the user should check the queue for any feedback records indicating a terminating error condition.

MODLUD (Suspend, De-activate, and Activate)

This combination of functions of the MODLUD instruction can be executed to perform a logical unit save operation. This atomic set of instructions perform a suspend, reset, de-activate, and activate. Therefore, the user is assured that all REQIOs are called back and placed on the user's queue and that the logical unit is in an active path state ready for a new set of REQIOs to process.

Note: A reset causes any available unsolicited data to be relinquished, because the device management has no way to give the data to the user.

REQIO

This instruction is used for two purposes. First, it is used to perform I/O operations on the various paths (MSCP-to-physical unit, MSCP-to-logical unit, and logical unit-to-logical unit). Second, the REQIO (continue) is used to return the logical unit to active path state after it has encountered a terminating error.

Notes:

1. An REQIO transmit only on the SNA any flow defaults to an REQIO transmit only on the SNA normal flow.
2. All transmit RUs should be doubleword aligned and should not cross segment boundaries. The device management takes care of these situations, but the user encounters a performance degradation when the transmit RUs are not properly aligned.
3. One REQIO cannot contain more than 32 transmit RDs if it is to be sent to a display.
4. The display REQIO transmit RDs must begin with a request header containing a first of chain entry and end with a request header containing a last of chain entry (full SNA chains only).

Device Management Source/Sink Request (SSR) General

The bits in the SSR function field have the following meanings for device management.

Bits 0-3

Hex 8 = Normal Request I/O
Hex 4 = MSCP Request I/O

Bits 4-5

These bits define the SNA flow that all the request descriptors (RD) are to be processed on. The bit values and their meanings are:

Bits	Meaning
00 and 10 10	The RDs are to flow on the SNA normal flow.
01	The RDs are to flow on the SNA expedited flow.
11	The RDs are to flow on either the SNA normal or SNA expedited flow (that is, it implements a receive any function). **Note:** This bit setting is changed to 10 (normal flow) if the first RD is a transmit.

Bits 6-7 Reserved (binary 0)

Device Management Request Descriptor

This RD is used for communications devices. It contains information for the transmission header (TH) and the request/response header (RH). The RD also indicates the length and location of the request/response unit (RU) in the SSD (source/sink data area). Both transmit and receive RDs may be specified in the SSR, but all of the transmit RDs must precede any receive RDs.

The request descriptor field format (in bytes) is:

0	1-3	4-5	6-7	8-10	11	12-15
RU Flow Type	Reserved	Sequence Number	RU Length	Response Header	Reserved	RU Offset

The meaning of the request descriptor format is as follows:

Byte **Meaning**

0 Bit 0 of the RU flow type indicates whether this is a transmit or receive RD.

0 = Transmit RD
1 = Receive RD

Bit 1 indicates whether the RD has been processed by the machine or is awaiting processing. The device management ensures that this bit is set to 0 when the Request I/O instruction is received. This bit cannot be used by the user to cause the device management to skip the RD. The device management always assumes that all the RDs in the REQIO instruction are to be processed.

In a terminating or nonterminating error situation, the RD completion count points to the RD on which the error was discovered. Bit 1 is marked processed. The bit values are:

0 = Not processed
1 = Processed

Bits 2-7 are reserved and must be binary 0's.

1-3 Reserved (binary 0)

Byte **Meaning**

4-5 The 2-byte binary sequence number is assigned and controlled by the machine. For a transmit RD, the machine assigns a sequence number and inserts it into the sequence number field. For a receive RD, the machine inserts the received sequence number into the sequence number field.

6-7 For a transmit RD, this field contains the length of the output RU. A length of 0 indicates there is no RU associated with this RD. For a receive RD, before it is processed, this field specifies the space available for the RU in the SSD. After the receive RD is processed, this field contains the length of the received RU. If a group of receive RDs is specified in the SSR, the space available for all of the RUs can be specified in the first RD of the group. The RU length field in the remaining RDs within the group is set to 0. When the RDs in the group are processed the RU length and offset fields are updated to the length and offset of the received RUs. The SSR can have more than one group of receive RDs. The RUs for all RDs in the SSR must be located in the same SSD.

Byte	Meaning
8-10	These 3 bytes contain the information for the request/response header (RH). For a transmit RD, the information for the outbound RH is supplied by the machine in this field. For a receive RD, the information from the received RH is inserted in this field when the RD is processed. All unused bits are reserved and must be set to 0. Byte 0 (request header) Bit 0 equals 0 indicates whether this RU is an SNA request or an SNA response: 0 = SNA request 1 = SNA response Bits 1-2 identify which type of RU is being processed. The bits and their meanings are: 00 = Function manager data (FMD) 01 = Network control 10 = Data flow control 11 = Session control Bit 3 is reserved (binary 0). Bit 4 indicates which format is used in the associated RU. The use of formats is session and RU type dependent. One use of this bit is to indicate the presence of an FM header in the RU. Bit 5 0 Indicates a valid request 1 Indicates the first 4 bytes of the RU are sense data for exception request. Bits 6-7 indicate which element of a chain of RUs is being sent. The bit settings are as follows: 00 = Middle of chain 01 = Last of chain 10 = First of chain 11 = Only in chain

Byte	Meaning
8-10 (continued)	Byte 1 (request header) Bits 0, 2, and 3 are used in combination to specify the type of response required. Bit 0 is always 1 (except isolated pacing response) and bit 2 is always 0 for SNA 0081 implementation. If bit 3 is 0, a definite response must be sent. If bit 3 is 1 (exception response mode), a negative response is sent only if an error is encountered, otherwise, no response is sent. Bit 1 is reserved (binary 0). Bits 4-6 are reserved (binary 0). Bit 7 is used by device management for pacing control. Device management ensures that it contains the correct setting. Byte 2 (request header) Bits 0-1 are reserved (binary 0). Bit 2 is a change direction indicator. It is used in a half duplex environment to control the orderly transmission of messages. The bit settings are: 0 = Do not change direction 1 = Change direction Bits 3-7 are reserved (binary 0).

Byte	Meaning
8-10 (continued)	
	Byte 0 (response header)
	Bit 0 = 1 indicates this RU is an SNA response.
	Bits 1-2 identify which of the four types of RUs are being processed: 00 = Function manager data (FMD) 01 = Network control 10 = Data flow control 11 = Session control
	Bit 3 is reserved (binary 0).
	Bit 4 is the same as for request header.
	Bit 5 indicates whether any sense data is included in the response. 0 = Sense data not included. 1 = The first 4 bytes of the RU contain sense data.
	Bits 6-7 indicate which element of a chain of RUs is being received. The bit settings are: 00 = Middle of chain 01 = Last of chain 10 = First of chain 11 = Only in chain

Byte	Meaning
8-10 (continued)	
	Byte 1 (response header)
	Bit **Meaning** 0 Same as the request header 1 Reserved (binary 0) 2 Same as the request header 3 0 = Positive response 1 = Negative response 4-6 Reserved (binary 0) 7 Used by device management for pacing control. Device management ensures that it contains the correct setting.
	Byte 2 (response header)
	All of byte 2 is reserved (binary 0).

Byte	Meaning
11	Reserved (binary 0)
12-15	This 4-byte RU offset is the displacement, in bytes, to the beginning of an RU from the beginning of the SSD space. An RU associated with a specific RD can be located by displacing into the SSD space an amount equivalent to the RU offset. The RU offset for a receive RD is updated by the machine if the RU length in the RD is specified as 0 (that is, the RD is designated as a part of an RD group). **Note:** It is strongly recommended that the offsets for send RDs point to RUs on double-word boundaries and that the RU does not cross an SID boundary. If either of these two conditions exists, the device management allocates storage and moves the RU data. This action degrades performance considerably. These alignments are required by the machine's operational units.

The SSD for the device management RDs contains RUs as defined by the RD's location in the SSR. One RD defines one RU. The location of the RU within the SSD is determined by the RU offset field in the RD.

For transmit RDs, the offset must be supplied to the machine in the RD. For receive RDs, the offset can be supplied for each RD or the offset can be supplied for the first RD of a group and the machine determines the offset for each subsequent RD in the group, based on the length of the received RU. A receive RD is considered a member of a group if its RU length is 0. More than one group of receive RDs can be specified in an SSR but all of the RUs for those RDs must be located in one SSD.

FEEDBACK RECORD

The feedback information for the device management consists of the error summary field and the number of request descriptors processed. All other status fields are not used by the work station controller.

The following table gives the error conditions, the severity code, and recovery procedures. The severity codes are returned in the error summary field.

Note: All feedback codes may have the MSCP bit on. This bit is turned on only if the REQIO function field indicates an REQIO (MSCP). (For example, normal completion for an REQIO (MSCP) is hex 0800 instead of hex 0000.)

Code	Severity	Error Condition	Recovery Procedure
0000	Normal	None	REQIO completed normally; continue normal processing.
0008	Normal	None	REQIO (continue) completed normally; the station is no longer in terminating error mode. Continue normal processing.
0044	Normal	Source/sink data area too small. The current request descriptor has an SSD remaining length field that is smaller than the request unit received from the station.	1. Handle this feedback as if it were an event for unsolicited data. 2. Send the device management a REQIO to pick up the data. 3. If data is not desired, do a MODLUD (reset, activate). **Note**: This also recalls all outstanding REQIOs for this logical unit and purges all unsolicited data.
4009	Nonterminating	REQIO partially complete due to the execution of the MODLUD (reset) command. Some RDs were processed.	Perform the reset cleanup process.
400A	Nonterminating	REQIO complete due to the execution of an MODLUD (reset) command. No RDs were processed.	Continue performing the cleanup process.
4031	Nonterminating	REQIO (continue) was executed, and there was no trace of the object being varied on.	1. Vary on the object.
4088	Nonterminating	Invalid RD sequence. A sequence of transmit/receive/transmit is not allowed in a REQIO operation.	Split the REQIO operation into two REQIO operations. The first one does transmit/receive and the second one does the remaining transmit. **Note:** Device management REQIOs can be only of the following variety: • Transmit only • Transmit/receive • Receive only

Code	Severity	Error Condition	Recovery Procedure
C022	Terminating	Station/terminal failure. The station/terminal has encountered a serious error.	1. The terminal receiving this feedback code must be varied off. If the CD event hex 0004 is received, the station must be varied off only after each LUD has been varied off. 2. For a station failure, the specific reason may be obtained by monitoring for: CD Event – Hex 4 Type – Hex 5 Subtype – Hex 1
C045	Terminating	REQIO instruction has an RD in it that specifies an output request unit that is larger than the allowable request unit size for this controller.	Set the correct request unit size in the RD. Reissue the REQIO instruction after issuing a Request I/O (continue) instruction.
C084	Terminating	The REQIO instruction has an invalid SSD pointer. The SSD pointer cannot be 0 if the data length is not 0.	Correct either the SSD pointer or the data length field.
C086	Terminating	The REQIO that was sent to a display device has more than 32 transmit RDs in it.	Structure the REQIOs so that they have no more than 32 transmit RDs.
C087	Terminating	The REQIO that was sent to a display device was not a complete SNA chain.	Make all REQIOs sent to displays, a complete SNA chain.

Note: Terminating error severity requires the user to use a REQIO (continue) command to bring the path back to an active session state or a MODLUD (reset, activate) to clear the path for normal traffic.

EVENTS SIGNALED BY WORK STATION CONTROLLER SUPPORT

Logical Unit Description Events

Supervisory Service Request Event: Hex 000B

Type: 04

Subtype: 01

02

These events are signaled when an SNA FM data request is received by the MSCP. See *Machine Services Control Point (MSCP)* section in this chapter and Chapter 21 for a more detailed description of this event.

The RU data provided by work station controller devices for these events consists of up to 80 bytes of data as indicated in the data length field for the event.

LUD Contact Event: Hex 000B

Type: 06

Subtype: 01 This subtype is signaled upon successful LUD contact.

02 This subtype (unsuccessful LUD contact), is never signaled for work station controller devices. In these cases, the Modify LUD (vary on) instruction signals a source/sink resource not available exception.

This event is signaled when the LUD vary on processing is completed by the MSCP. See Chapter 21 and the *Machine Services Control Point (MSCP)* section in this chapter for a more detailed description of this event.

Unsolicited Event: Hex 000B

Type: 05 This event indicates that unsolicited data is available for this LU.

Subtype: 01 The data is on the expedited SNA flow.

02 The data is on the normal SNA flow.

This event is signaled only for the first frame received and any time not enough receive RDs are available. The following events are signaled by the common function, build feedback record, for device management.

LUD Failure Event: Hex 000B

Type: 08 The device failure event is signaled whenever a varied on device goes to a state where it can no longer be used.

Subtype: 01 This subtype is never signaled for work station controller devices.

02 This subtype is signaled when an SNA LUSTAT request is received on the MSCP-LU session indicating that the device is not available.

REQIO Complete Event: Hex 000B

Type: 09 This event indicates that the REQIO is completed.

Subtype: 01 This event is signaled only if the user requests it.

Queue Destroyed Event: Hex 000B

Type: 0A This LUD event indicates that the return queue specified in the REQIO instruction has been destroyed.

Subtype: 01 Queue destroyed.

Note: This LU path is also placed in terminating error mode.

Controller Description Events

CD Contact Event: Hex 0004

Type: 04

Subtype: 01 This subtype is signaled upon successful CD contact.

02 This subtype (unsuccessful CD contact) is never signaled for work station controller devices. In these cases, the Modify CD (vary on) instruction signals the source/sink resource not available exception and the CD failure event.

This event is signaled when the CD vary on processing is completed by the MSCP. See Chapter 21 and *Machine Services Control Point (MSCP)* section in this chapter for a more detailed description of this event.

CD Failure Event: Hex 0004

Type: 05 This event indicates a CD failure.

Subtype: 01 The work station controller is not capable of normal operations. If the error code in the feedback record is hex 0405, the optional pointer in the event-related data will be a system pointer to the LUD that is attached to this CD, which had an invalid data problem in the LUD device-specific area.

02 Data is being received from a device and the logical unit is not varied on or the path is not activated.

EXCEPTION CODES SIGNALED BY WORK STATION CONTROLLER SUPPORT

Modify CD Exceptions

For the work station controller CD object, the source/sink resource not available exception (hex 3404) is signaled only by the Modify CD (vary on) instruction. In this case, the generic exception data is hex 2202 and the specific data field contains the same error code provided as event-related data for the concurrently signaled CD event failure.

Modify LUD Exceptions

For the LUDs attached to the work station controllers, the source/sink resource not available exception (hex 3404) is signaled by the Modify LUD instruction for the following state changes.

	Exception Data	
State	**Generic (hex)**	**Specific**
Vary On	2302	Hex C822 – Station failure occurred. CD failure event has been signaled. Hex 0800 – Activate logical unit failure indicated by the device.
Suspend	2312	Hex 1201 – Suspend was rejected because it could not be completed within the time-out specified.
Quiesce	2313	Hex 1302 – Quiesce rejected due to a terminating error condition on a Request I/O instruction. Hex 1301 – Quiesce was rejected because it could not be completed within the time-out specified.

OBJECT CREATION DATA FOR SUPPORTED DEVICES

This section defines the data needed to create the objects necessary to attach a specific device to the machine. A section is presented for each device supported.

Work Station Controller

CD Template Data for Work Station Controller

Field Name	Length	Entry
CD type	Char(2)	Char 00
Unit type	Char(4)	Char WSCb
Model number	Char(4)	All blanks
Physical address	Char(8)	
• Reserved	Bin(4)	Bin 0
• Station's line address	Bin(2)	Bin 0
• Operational unit number	Bin(2)	Hex 0030, Hex 0070, Hex 00B0, or Hex 00F0
Power control	Char(2)	Hex 0000
Station control information	Char(32)	
• XID information	Char(21)	
– XID format	Bit(4)	Bin 0000
– Physical unit type	Bit(4)	Bin 0000
– XID information length	Bit(8)	Hex 00
– Station's block number	Bit(12)	Hex 000
– Specific identification	Bit(20)	User-assigned ID that must be uniquely assigned for each work station controller within the system.
– Reserved	Char(4)	Hex 00000000
– Maximum length received	Bin(2)	Hex 0000
– Reserved	Char(4)	Hex 00000000
– Frames limit	Bit(8)	Hex 0000
– Reserved	Char(4)	Hex 00000000
• Station definition	Char(1)	Hex 00
• Reserved	Char(2)	Hex 0000
• Path information unit type	Bin(2)	Hex 0000
• Reserved	Char(6)	Bin 0

CD Template Data for Work Station Controller (continued)

Field Name	Length	Entry
Selected modes	Char(16)	Bin 0
Activate physical unit data	Char(16)	Bin 0
Dial digits	Char(32)	Bin 0
Specific characteristics	Bin(2)	Hex 0000
XID information	Char(2)	
• Length of XID data	Bin(2)	Hex 0000
Unit-specific data area	Char(6)	
• Length of unit-specific area	Bin(2)	Hex 0002
• Length of modifiable area	Bin(2)	Hex 0002
Unit-specific contents area	Bin(16)	Hex 0000

LUD Template Data for the 5251 or 5252 Display

Field Name	Length	Entry
LUD type	Char(2)	Char 10
Device type	Char(4)	Char 5251 or Char 5252
Model number	Char(4)	Char 0001 for 960-character screen 5251 Char 0011 for 1920-character screen 5251 Char 0001 for 960-character screen 5252
Physical address	Char(8)	
• Reserved	Bin(2)	Hex 0000
• Logical unit address	Bin(2)	Hex 0000-Hex 0013 Select the lowest number for the first LUD created. Best performance is obtained when these logical session identification fields (LSID) are assigned contiguously for each succeeding LUD starting with LSID hex 0000.
• Station line address	Bin(2)	Hex 0000
• Operational unit number	Bin(2)	Same number as in CD to which this LUD attaches.
Power control	Char(2)	Hex 0000
Session information	Char(20)	
• Inbound pacing	Bin(2)	Hex 0000
• Outbound pacing	Bin(2)	Hex 0000
• Request unit buffer size	Bin(2)	Hex 0100 = 256 bytes Hex 0C00 = 3072 bytes Select 256 bytes for the 5251 Model 2 or 12 compatibility operation of the WSC display.
• ACTLU	Char(1)	Hex 01
• ACTLU parameters	Char(3)	Hex 0D0101
• ACTLU response	Char(8)	Bin 0
Load/dump indicator	Char(16)	Bin 0
Specific characteristics	Bin(2)	Hex 0014 = 20 bytes Length of specific characteristics area
• Device descriptor	Bit(16)	Hex 0041 = 5252 Hex 0081 = 5251 Model 0001 Hex 00C2 = 5251 Model 0011
• Keyboard option	Bit(16)	Hex 0000 = No keyboard attached Hex 0002 = Typewriter keyboard Hex 0003 = ASCII keyboard Hex 0004 = Data-entry keyboard

LUD Template Data for the 5251 or 5252 Display (continued)

Field Name	Length	Entry
• Keyboard option (continued)	Bit(16)	Hex 0005 = Data-entry keyboard with inverted key option (proof) Hex 0008 = International/ASCII data-entry keyboard Hex 0009 = International/ASCII data-entry keyboard with inverted key option (proof) Hex 000A = World Trade typewriter keyboard Hex 000B = International typewriter keyboard Hex 000C = World Trade data-entry keyboard Hex 000D = World Trade data-entry keyboard with inverted key option (proof)
• Cable address for this LU	Bit(16)	Hex 0000-hex 0007 or hex 0010-hex 0017 Cable address represents the physical address of a twinaxial or coaxial cable. It is derived from the connector and OU number as follows: Connector numbers 0 through 15 are attached to OU 30; connector numbers 16 through 31 are attached to OU 70; connector numbers 32 through 47 are attached to OU BO; connector numbers 48 through 63 are attached to OU FO. Cable addresses 0-7 correspond to connectors 0-7 on OU 30, connectors 16-23 on OU 70, connectors 32-39 on OU BO, and connectors 48-55 on OU FO. Cable addresses 10-17 correspond to connectors 8-15 on OU 30, connectors 24-31 on OU 70, connectors 40-47 on OU BO, and connectors 56-63 on OU FO.
• Head (station) address	Bit(16)	Hex 0000-Hex 0006 for 5251 Hex 0000-Hex 0005 for 5252 This is the physical address of a head (display or printer) on a daisy chained twinaxial (Cable Thru feature). Allowable addresses are 0-6 for the 5251 and 0-5 for the 5252. The values (0-6) are selected via rocker switches on the devices.
• Reserved	Bit(16)	Hex 0000
• Katakana keyboard attached	Bit(16)	Hex 0000 = Not attached Hex 0001 = Attached
• Translate table name	Char(8)	(See the following table.)

LUD Template Data for the 5251 or 5252 Display (continued)

Translate Table	Keyboard	Language or Country	WSC Char Generator
#TPNDAGB	Data-Entry	Austria/Germany	Basic
#TPNDAGI	Data-Entry	Austria/Germany	International
#TPNDBLB	Data-Entry	Belgium	Basic
#TPNDBLI	Data-Entry	Belgium	International
#TPNDBRB	Data-Entry	Brazil	Basic
#TPNDBRI	Data-Entry	Brazil	International
#TPNDCAB	Data-Entry	Canada (French)	Basic
#TPNDCAI	Data-Entry	Canada (French)	International
#TPNDDMB	Data-Entry	Denmark	Basic
#TPNDDMI	Data-Entry	Denmark	International
#TPNDFAB	Data-Entry	French (Azerty)	Basic
#TPNDFAI	Data-Entry	French (Azerty)	International
#TPNDFNB	Data-Entry	Finland	Basic
#TPNDFNI	Data-Entry	Finland	International
#TPNDFQB	Data-Entry	French (Qwerty)	Basic
#TPNDFQI	Data-Entry	French (Qwerty)	International
#TPNDINB	Data-Entry	International	Basic
#TPNDINI	Data-Entry	International	International
#TPNDITB	Data-Entry	Italian	Basic
#TPNDITI	Data-Entry	Italian	International
#TPNDJEB	Data-Entry	Japan (English)	Basic
#TPNDJEI	Data-Entry	Japan (English)	International

LUD Template Data for the 5251 or 5252 Display (continued)

Translate Table	Keyboard	Language or Country	WSC Char Generator
#TPNDKAB	Data-Entry	Japan (KATA)	Basic
#TPNDNWB	Data-Entry	Norwegian	Basic
#TPNDNWI	Data-Entry	Norwegian	International
#TPNDPRB	Data-Entry	Portuguese	Basic
#TPNDPRI	Data-Entry	Portuguese	International
#TPNDSPB	Data-Entry	Spanish	Basic
#TPNDSPI	Data-Entry	Spanish	International
#TPNDSSB	Data-Entry	Spanish Speaking	Basic
#TPNDSSI	Data-Entry	Spanish Speaking	International
#TPNDSWB	Data-Entry	Swedish	Basic
#TPNDSWI	Data-Entry	Swedish	International
#TPNDUKB	Data-Entry	United Kingdom	Basic
#TPNDUKI	Data-Entry	United Kingdom	International
#TPNDUSB	Data-Entry	United States	Basic
#TPNDUSI	Data-Entry	United States	International
#TPNTAGB	Typewriter	Austria/Germany	Basic
#TPNTAGI	Typewriter	Austria/Germany	International
#TPNTBLB	Typewriter	Belgium	Basic
#TPNTBLI	Typewriter	Belgium	International
#TPNTBRB	Typewriter	Brazil	Basic
#TPNTBRI	Typewriter	Brazil	International

LUD Template Data for the 5251 or 5252 Display (continued)

Translate Table	Keyboard	Language or Country	WSC Char Generator
#TPNTCAB	Typewriter	Canada (French)	Basic
#TPNTCAI	Typewriter	Canada (French)	International
#TPNTDMB	Typewriter	Denmark	Basic
#TPNTDMI	Typewriter	Denmark	International
#TPNTFAB	Typewriter	French (Azerty)	Basic
#TPNTFAI	Typewriter	French (Azerty)	International
#TPNTFNB	Typewriter	Finland	Basic
#TPNTFNI	Typewriter	Finland	International
#TPNTFQB	Typewriter	French (Qwerty)	Basic
#TPNTFQI	Typewriter	French (Qwerty)	International
#TPNTINB	Typewriter	International	Basic
#TPNTINI	Typewriter	International	International
#TPNTITB	Typewriter	Italian	Basic
#TPNTITI	Typewriter	Italian	International
#TPNTJEB	Typewriter	Japan (English)	Basic
#TPNTJEI	Typewriter	Japan (English)	International
#TPNTKAB	Typewriter	Japan (KATA)	Basic
#TPNTNWB	Typewriter	Norwegian	Basic
#TPNTNWI	Typewriter	Norwegian	International
#TPNTPRB	Typewriter	Portuguese	Basic
#TPNTPRI	Typewriter	Portuguese	International
#TPNTSPB	Typewriter	Spanish	Basic

LUD Template Data for the 5251 or 5252 Display (continued)

Translate Table	Keyboard	Language or Country	WSC Char Generator
#TPNTSPI	Typewriter	Spanish	International
#TPNTSSB	Typewriter	Spanish Speaking	Basic
#TPNTSSI	Typewriter	Spanish Speaking	International
#TPNTSWB	Typewriter	Swedish	Basic
#TPNTSWI	Typewriter	Swedish	International
#TPNTUAB	Typewriter	United States (ASCII)	Basic
#TPNTUAI	Typewriter	United States (ASCII)	International
#TPNTUKB	Typewriter	United Kingdom	Basic
#TPNTUKI	Typewriter	United Kingdom	International
#TPNTUSB	Typewriter	United States	Basic
#TPNTUSI	Typewriter	United States	International

Field Name (continued)	Length (continued)	Entry (continued)
Retry value sets	Bin(2)	Hex 0000
Error threshold sets	Bin(2)	Hex 0000
Device-specific contents		
• Length of device contents	Bin(2)	Hex 0004
• Length of device-specific contents modifiable and materializable	Bin(2)	Hex 0004
• Device-specific area	Char(4)	Hex 0000
– Reserved	Bit(16)	
– Device format table length	Bit(16)	Hex 0001 = 256 bytes Hex 0002 = 512 bytes Hex 0003 = 768 bytes Use as small a value as possible since this uses extended data store shared with other devices on this WSC and could limit the number of devices on this WSC.

LUD Template Data for the 5256 Printer

Field Name	Length	Entry
LUD type	Char(2)	Char 10
Device type	Char(4)	Char 5256
Model number	Char(4)	Char 0001 for 40 characters per second Char 0002 for 80 characters per second Char 0003 for 120 characters per second
Physical address	Char(8)	
• Reserved	Bin(2)	Hex 0000
• Logical unit address	Bin(2)	Hex 0000-hex 0013 Best performance is obtained when these logical session identification fields (LSID) are assigned contiguously for each succeeding LUD starting with LSID (hex 0000).
• Station line address	Bin(2)	Hex 0000
• Operational unit number	Bin(2)	Same as in CD to which this LUD attaches
Power control	Char(2)	Hex 0000
Session information	Char(20)	
• Inbound pacing	Bin(2)	Hex 0000
• Outbound pacing	Bin(2)	Hex 0002-hex 0007 This field cannot be greater than the number of print buffers specified in the device-specific area of the LUD.
• Request unit buffer size	Bin(2)	Hex 0100 = 256 bytes
• ACTLU	Char(1)	Hex 01
• ACTLU parameters	Char(3)	Hex 0D0101
• ACTLU response	Char(8)	Bin 0
Load/dump indicator	Char(4)	Bin 0
Specific characteristics	Bin(2)	Hex 0014 = 20 bytes Length of specific characteristics area
• Device descriptor	Bit(16)	Hex 0020 = Printer
• Keyboard option	Bit(16)	Always hex 0000

LUD Template Data for the 5256 Printer (continued)

Field Name	Length	Entry
Specific characteristics (continued)		
• Cable address for this LU	Bit(16)	Hex 0000-hex 0007 or hex 0010-hex 0017 Cable address represents the physical address of a twinaxial or coaxial cable. It is derived from the connector and OU number as follows: Connector numbers 0 through 15 are attached to OU 30; connector numbers 16 through 31 are attached to OU 70; connector numbers 32 through 47 are attached to OU BO; connector numbers 48 through 63 are attached to OU FO. Cable addresses 0-7 correspond to connectors 0-7 on OU 30, connectors 16-23 on OU 70, connectors 32-39 on OU BO, and connectors 48-55 on OU FO. Cable addresses 10-17 correspond to connectors 8-15 on OU 30, connectors 24-31 on OU 70, connectors 40-47 on OU BO, and connectors 56-63 on OU FO.
• Head (station) address	Bit(16)	Hex 0000-Hex 0006 This is the physical address of a head (display or printer) on a twinaxial cable (Cable Thru feature). Allowable addresses are 0-6. The values (0-6) are selected via rocker switches on the device.
• Reserved	Bit(16)	Hex 0000
• Katakana keyboard attached	Bit(16)	Always hex 0000
• Translated table name	Char(8)	Always binary 0
Retry value sets	Bin(2)	Hex 0000
Error threshold sets	Bin(2)	Hex 0000
Device-specific contents		
• Length of device contents	Bin(2)	Hex 0004
• Length of device-specific contents modifiable and materializable	Bin(2)	Hex 0004
• Device-specific area	Char(4)	Hex 0000
– Reserved	Bit(16)	
– Number of 256-byte printer buffers	Bit(16)	Hex 0003-hex 0007 This field must be equal to or greater than the outbound pacing.

Chapter 25. Load/Dump Object Management

The load/dump function enables the user to first save (back up) certain permanent objects by dumping these objects to a load/dump medium (such as diskettes or magnetic tape) and then load (restore) these objects when needed.

The objects that can be processed by the load/dump function are:

- Data spaces
- Data space indexes
- Programs
- Space objects
- Independent indexes

Except for programs, these objects can be dumped, loaded over an existing object, or loaded onto a system where they currently do not exist.

An existing program cannot be overlaid (loaded) on the system. Programs can only be loaded as new objects with the Create and Load command.

LOAD/DUMP COMMANDS

The L/D (load/dump) function uses unique commands to process permanent objects. The L/D commands are as follows:

- Dump
- Load
- Create and Load
- Set User Profile
- Set Context
- Read Object Identification

Session Types

For the L/D function, there can be two session types (dump or load) in the LUD (logical unit description). Only the Dump command is allowed for the dump session; all other L/D commands are allowed for the load session. When the Read Object Identification command (load session) is issued, no other command types are allowed in the Request I/O instruction.

Sequence of Operations

The following is an overview of an L/D function:

1. An L/D Modify Logical Unit Description (activate) session is issued to open the L/D session. The LUD must be properly initialized to the operation mode (load or dump) at this time.
2. A normal Request I/O instruction that points to an SSR (source/sink request) is issued. This SSR contains commands and pointers to the objects to be loaded or dumped from the LD medium.
3. The L/D function checks the Request I/O instruction for errors and processes all commands.
4. The L/D function sends a feedback message to the Request I/O response queue to indicate the completion of the function.
5. A Modify LUD (de-activate) session is issued to close the L/D session.

Note: The L/D function operates only in the EBCDIC mode for tape and diskette; this function operates with 1 K byte sector sizes on the diskette. The user has no control over these values, and any value the user puts in the LUD is ignored. For tape operations, the user may specify the block size by setting a field in the device-specific area of the LUD or let block size default to the optimum block size.

Dump Command (D)

A permanent object is copied to the L/D medium.

Coding the Dump Command

1. The hex value (hex 01) for the Dump command is placed in the command field of the RD (request descriptor).

2. A system pointer to the object to be dumped is placed in the RD pointer field.

3. The object ID (identification) of the object to be dumped is placed in the RD object ID field.

Programming Considerations

- The object need not be addressed by a context.
- Indexes with pointers are not dumped.
- Both observable and nonobservable programs can be dumped.
- Dumping the object does not change the object.
- Damaged objects cannot be dumped.
- Partial damaged objects cannot be dumped.
- In a Request I/O instruction the same object can be dumped any number of times.
- A suspended object can be dumped.
- The object name, type, and subtype (in the object ID field) must match the object pointed to by the system pointer in the RD.
- When dumping a data space index, all data spaces must immediately precede the associated data space index. (See *Load/Dump Data Base Networks* in this chapter.)

Lock Enforcement

- Materialization
  - On all objects to be dumped

Load Command (L)

A permanent object is copied from the L/D medium to overlay an object on the system. The object ID field (in the RD) is compared to the object ID on the L/D medium. When a match is found, that object is loaded, replacing the object specified by the pointer field of the RD. A value in the compare length of the RD specifies the number of positions to be compared.

Coding the Load Command

1. The hex value (hex 02) for the Load command is placed in the command field of the RD.

2. The compare length for the object ID search is placed in the compare length field of the RD.

3. A system pointer to the object in the system to be replaced is placed in the pointer field.

4. The ID of the object is placed in the object ID field of the RD. This field may be initialized with 0-64 bytes of the object ID. The number of bytes initialized should not be less than the compare length in the RD. The object ID field is only used to search the L/D device and check for an object ID match.

Programming Considerations

- A program object cannot be loaded. It can be only created and loaded.
- The object in the system is truncated or extended as necessary to fit the version being loaded. The object owner's user profile is charged/credited for the space.
- The size of the object and the associated space of the object can be larger after a load than it was when the object was dumped because of internal machine allocations.
- The object that is to be overlaid on the system can be suspended.
- The object that is to be loaded from the L/D medium can be suspended.

- The authority (public and owners), object owner's user profile, and context addressing object attributes of the object to be overlaid are not changed by the Load operation. All the other object attributes are from the object just loaded.
- The DS (data space) and DSI (data space index) must not have an active cursor in use.
- When a DS is being loaded, the entry definition table of the object on the system must match the one to be loaded before a load operation can take place.
- If, in the same Request I/O instruction, all the DSs under a DSI are not loaded before the DSI, the DSI is not loaded. Intertwined networks on the media can be loaded (see *Load/Dump Data Base Networks* in this chapter).
- The DSs under the DSI to be overlaid must also be under the DSI to be loaded.
- The object that is to be overlaid on the system must have the same name, type, and subtype as the object on the L/D media that is to be loaded.
- The pointers in a space object or in any associated space are not resolved by a Load command. These pointers are made to look like data. The user must restore these pointers, if desired.
- The associated space of an object in the system is replaced by the associated space of the object being loaded.
- A data space index invalidated event is signaled for any DSI that is not overlaid but addresses data spaces that are overlaid.
- External links for DSIs and DSs are resolved after the DSI is loaded.
- If the object to be overlaid is damaged, the Load operation does not take place.
- If the object to be overlaid is only partially damaged, the load operation does take place.
- Only versions of objects that are older than or as old as the version of the L/D code are loaded. The version level check is also done on the user exit program in the DSI object.
- In a Request I/O instruction, the same object can be loaded any number of times.
- When a DS is loaded, all DSIs (even those that may not be in the REQIO) over that DS must not be in use, damaged, or destroyed.
- When a DSI is loaded, the DS key specification in that DSI and in the DSI to be overlaid must match and have the same number of records.
- The ID for the L/D media is the ID that was supplied in the dump object ID field when the object was dumped.
- The object to be overlaid need not be addressed by a context.

Lock Enforcement

- Object control
  - On the objects to be loaded
  - On any DSI associated with a DS to be loaded even if the DSI is not loaded

Create and Load Command (CL)

The current file on the L/D medium is searched for the object to be created by comparing the object ID field in the RD to the object ID on the L/D medium. When a match is found, the system allocates space for the object and then loads the object into this space. Addressability to the object is placed in the context specified by the previous Set Context command only if the addressability field in the RD equals hex 00. The L/D function provides addressability to the created object by unconditionally inserting a system pointer in the pointer field of the RD. Ownership of the object is assigned to the user profile specified by the previous Set User Profile command.

Coding the Create and Load Command

1. The hex value (hex 16) for the Create and Load command is placed in the command field of the RD.

2. The compare length for the object ID search is placed in the compare length field of the RD.

3. The addressability field of the RD is set to hex 00 or hex 01 to indicate whether the object is to be addressed by the context specified by the previous Set Context command or whether the object is not to be addressed by any context.

4. The object ID is placed in the object ID field of the RD. This field may be initialized with 0-64 bytes of the object ID. The number of bytes initialized should not be less than the compare length in the RD. This field is only used to search the L/D device and check for an object ID match.

Programming Considerations

- The pointer field is not initialized because the L/D function returns a system pointer.

- If the addressability field of the RD is hex 00, there must not be two objects of the same name, type, and subtype in the context specified by the previous Set Context command.

- The ID on the L/D media is the one that was supplied when the object was dumped by the dump object ID field.

- The associated space of the object on the L/D medium becomes the associated space of the object created.

- Internal pointers within the object are corrected to reflect address changes.

- External links only for DSI and DS are resolved after the DSI is created and loaded.

- The pointers in a space object or in any associated space are not resolved by a Create and Load command. These pointers are made to look like data. The user must restore these pointers, if desired.

- A Set User Profile command must precede any Create and Load command within the same Request I/O instruction.

- If the addressability field in the Create and Load command RD is hex 00, a Set Context command must precede this RD within the same Request I/O instruction.

- If the addressability field in the Create and Load command RD is hex 01, the addressability to the object is not placed in any context.

- Only versions of objects that are older than or as old as the version of the L/D code are created. A version level check is also done on the user exit program in the DSI object.

- Except for the user profile and context, all attributes of the created object are identical to the attributes it had when it was dumped.

- When a DS or DSI is created, an attempt is made to place it on the unit from which it was dumped.

- If, in the same Request I/O instruction, all the DSs under a DSI are not loaded or created and loaded before the DSI, the DSI is not loaded. Intertwined networks on the media can be created (see *Load/Dump Data Base Networks* in this chapter).

Lock Enforcement

- None

ət User Profile Command (SUP)

ıe Set User Profile command must precede any Create ıd Load command within a normal Request I/O struction because the SUP command specifies a user ·ofile for the object(s) created by the Create and Load ɔmmand. The pointer field in the RD must contain a ˈstem pointer that has addressability to the desired ɜer profile. After a Set User Profile command is ·ocessed, all subsequent Create and Load commands that same Request I/O instruction use that user ·ofile until another Set User Profile command is ıcountered.

ɔding the Set User Profile Command

1. The hex value (hex 24) for the Set User Profile command is placed in the command field of the RD.

2. A system pointer to the desired user profile is placed in the RD pointer field.

rogramming Considerations

The object ID field is not used.

The user profile must not be damaged.

ɔck Enforcement:

Modification
- On the user profile

Set Context Command (SCTX)

A specific context that can receive addressability to the object(s) created by the Create and Load command is selected. Within a normal Request I/O instruction, the Set Context command must precede the first Create and Load command that has a value of hex 00 in the addressability field of the RD. This requirement exists because the hex 00 value directs the L/D function to put addressability to the created object in a context. The pointer field in the RD must contain the address of this context. After a Set Context command is processed, all subsequent Create and Load commands with an addressability field equal to hex 00 in that same Request I/O instruction, use that same context until another Set Context command is processed.

Coding the Set Context Command

1. The hex value (hex 44) for the Set Context command is placed in the command field of the RD.

2. A system pointer to the desired context is placed in the RD pointer field.

Programming Considerations

- The object ID field is not used.
- The context must not be damaged.

Lock Enforcement

- Modification
  - On the context

Read Object Identification Command (ROID)

The data in the ID portion of an object on the L/D medium is retained and inserted into the ID field of the RD. The retrieved data can then be used to compile a listing of the objects on the file.

Coding the Read Object Identification Command

The hex value (hex 86) for the Read Object Identification command is placed in the command field of the RD.

Programming Considerations

- The pointer field is not used.
- No other command types may be issued in the same Request I/O instruction.

Lock Enforcement

- None

LOAD/DUMP REQUEST I/O (REQIO)

The user's interface to the L/D function is the Request I/O instruction. Two types of Request I/O instructions are used, the normal Request I/O and Request I/O (continue). The normal Request I/O instruction contains commands and the needed information to load objects in the system and dump objects from the system. The Request I/O (continue) instruction is used for error processing; it indicates that processing of the next normal Request I/O instruction should continue from the point where the error occurred (the user must build a new SSR for the continued operation). The L/D function uses the standard format of the SSR and an extended RD. No SSD (source/sink data) object is used.

Request Descriptor (RD)

The format of the RD in the SSR is as follows:

Command	Char(1)
Compare length	Char(1)
RD number for exception	Bin(2)
Addressability	Char(1)
Reserved	Char(1)
Pointer	Char(16)
Object ID	Char(64)

The length of each RD must be 96 bytes; the RD must be located on a 16-byte boundary.

Each RD must contain the necessary information to process at least one object. The maximum number of RDs allowed in the Request I/O SSR is 4000.

The L/D function always processes request descriptors in the order they appear in the source/sink request; that is, the first request descriptor is processed first, and the last request descriptor is processed last.

Command Field

The user must specify one of the following commands for each RD:

- Load (hex 02)
- Create and Load (hex 16)
- Dump (hex 01)
- Set User Profile (hex 24)
- Set Context (hex 44)
- Read Object ID (hex 86)

Compare Length Field

If the command is Load or Create and Load, the compare length field specifies how many bytes of the object ID on the L/D medium are to be compared with the object ID field. The compare length can be any value from 0 through 64. A length of 0 indicates that the next object on the L/D medium should be loaded. A length of 64 indicates that an exact match of the object ID is required before the load can occur.

Note: Because the tape and diskette are serial devices, the user should exercise caution when a compare length of less than 64 bytes is specified. This caution is necessary because the device starts searching for a match to the ID field from the point where the device was last positioned. Therefore, the system may load the wrong object if the user does not know the exact data in the object IDs and the sequence of the objects on the device.

RD Number for an Exception Field

If an exception error code is generated by a Request I/O instruction, the exception field can contain the RD number that caused the exception. This field is used only in the first RD of the SSR.

Addressability Field

The contents of the addressability field are meaningful only when the associated command is Create and Load. If this field contains a value of hex 00, addressability for the object being created is put into the context specified by the last Set Context command. If this field contains a value of hex 01, addressability for the object being created is not put into any context.

Reserved Field

This field is reserved for use by the machine and any value put in it by the user will be overlaid.

Pointer Field

The pointer field provides addressability to some of the objects associated with the L/D function. The pointer contained in this field for each L/D command is as follows:

- Dump – A system pointer to the object that is to be dumped to the L/D medium.
- Load – A system pointer to the object that is to be overlaid by the object from the L/D medium.
- Create and Load – The pointer field can be any value (pointer or data) when the RD is built for this command. The L/D function inserts a system pointer in this field after the object has been created and loaded. The pointer contains addressability to the created object. No authority is placed in this pointer.
- Set User Profile – A system pointer to the user profile that is given ownership of the object(s) on all subsequent Create and Load commands within the same normal Request I/O instruction.
- Set Context – A system pointer to the context where addressability can be inserted for the objects created by the Create and Load commands. Addressability to the object is inserted in the context only if the addressability field within the same normal Request I/O instruction contains a value of hex 00.
- Read Object ID – the pointer field is not used for Read Object ID commands.

Object ID Field

The object ID field consists of an object name (30 characters), object type (1 character), object subtype (1 character), and an ID extension (32 characters). The use of the object ID field depends on the specified L/D command.

Dump Command: The object name, type, and subtype must be supplied. The ID extension is optional; however, the entire object ID field and the object are dumped to the L/D medium.

Load Command: The object ID field is used when the L/D medium is searched for the correct object(s). The search operation consists of comparing the object ID field on the L/D medium to the object ID field in the RD until an equal condition occurs. The number of characters compared in the search operation is determined by the value in the compare length field of the RD.

Create and Load Command: The object ID field is used when the L/D medium is searched for the correct object(s). The search operation consists of comparing the object ID field on the L/D medium to the object ID field in the RD until an equal condition occurs. The number of characters compared in the search operation is determined by the value in the compare length field of the RD.

Read Object ID Command: The 64-byte object ID is read from the L/D medium, and the data is inserted into this field.

Set User Profile Command: The object ID field is not used for this command.

Set Context Command: The object ID field is not used for this command.

LOAD/DUMP MODIFY LUD

The L/D function conforms to the normal operation for the various Modify LUD sessions except when the session is changed from load to dump or dump to load. To change the sessions from load to dump or dump to load the user must:

1. Issue a Modify LUD (de-activate) instruction.
2. Change the operation mode byte in the LUD.
3. Issue a Modify LUD (activate) instruction.

The Modify LUD (reset) session (which may be required after an error condition) causes the L/D function to immediately stop processing the current normal Request I/O instruction and to send a feedback record for the associated normal Request I/O instruction. Included in the feedback record is the proper error code and an indicator that show how many RDs have already been processed. The L/D function then flushes the unprocessed Request I/O instructions by sending feedback records for each Request I/O instruction with the proper error code. After the L/D queue has been flushed, the Modify LUD (reset) operation is completed. The L/D function does a cleanup procedure, if necessary, on the RD it was processing when the Modify LUD (reset) request was received. The Modify LUD (reset) session state can leave the L/D device read/write head at an unknown location on the diskette; therefore, it is the user's responsibility to correctly position the L/D device read/write head after a Modify LUD (reset) instruction is processed.

The Modify LUD (suspend) session is used to interrupt the L/D function so that data interchange can occur or so that processing can be halted. The Modify LUD (suspend) session causes the L/D function to stop processing the current Request I/O (normal) instruction (1) on an RD boundary (2) at the end of a volume, or (3) at the end of a file. A feedback record is sent for the associated Request I/O (normal) instruction after the suspend session has occurred.

LOAD/DUMP FEEDBACK RECORD

For every Request I/O instruction received, the L/D function responds with a standard feedback record that contains the status of that Request I/O instruction. The feedback record is visible to the user when a Dequeue instruction is processed. Load/dump does not respond with the standard feedback record when the Request I/O instruction specifies dump and the object being dumped has undetected partial damage.

LOAD/DUMP ERROR PROCESSING

In L/D processing, there are four types of errors: exceptions, severe errors (nonrecoverable), recoverable errors (such as end of volume (EOV), end of file (EOF), end of tape (EOT)), and other errors. How these errors are processed depends on the type of error encountered.

Exceptions

The exception errors are detected by the Request I/O instruction. They result in an exception being generated and signaled to the user's program. At this time, I/O operations have not started because the L/D object handler has not been invoked. To recover from an exception, correct the error and resume the Request I/O instruction.

Exception error codes (in addition to normal exceptions) are generated by the Request I/O instruction when the associated preprocessing L/D errors are detected. The number of the RD being processed when the error causing the exception occurred can be found in the RD number for exception fields of the first RD in the SSR.

The following is a list of exceptions and the exception error codes signaled for L/D and the command(s) that could cause the exceptions.

Exception Code (hex)	Exception	Command Causing Exception
0A01	Unauthorized for operation	All L/D commands
0A04	Special authorization required	All L/D commands
1004	Damaged system object	Load, Dump, Set User Profile, and Set Context
1044	Partial system object damage	Dump
1A01	Invalid lock state	Load, Dump, Set User Profile, and Set Context
1C03	Machine storage limit exceeded	All L/D commands
2201	Object not found	Load, Dump, Set User Profile, and Set Context
2202	Object destroyed	Load, Dump, Set User Profile, and Set Context
2204	Object not eligible for operation	Load and Dump
2401	Pointer does not exist	Load, Dump, Set User Profile, and Set Context
2402	Pointer type invalid	Load, Dump, Set User Profile, and Set Context
2403	Pointer addressing invalid object	Load, Dump, Set User Profile, and Set Context
2E02	Process storage limit exceeded	All L/D commands
3801	Template value invalid	All L/D commands

Severe Errors

Severe errors are nonrecoverable errors, and they seldom occur until after an I/O operation has started. A Modify LUD (reset) or Modify LUD (de-activate) instruction must be issued if this type of error occurs.

The following is a list of the severe-error codes returned in the status field and explanations of those error codes.

Error Code (hex)	Definition
C4C0-C5C0	Storage operation error or machine storage exceeded.
C4C1-C5C1	The maximum auxiliary storage for the permanent objects field in the object owner's user profile has been exceeded for a Load command or Create and Load command.
C4C2-C5C2[1]	Invalid lock (I/O processing has not yet started except for the Create and Load command).
C5C3	User profile or context is full of entries for the Create and Load command.
C5C4	Duplicate Object (an object of the same name, type, subtype exists in the context specified by the Set Context command for this Create and Load command).
C4C5	Data base network violation on a Load or Create and Load command. • The DSI cannot be loaded because the DSs under it were created and loaded. • All DSs under this DSI do not immediately precede the DSI(s) for this network. • The DS(s) under the DSI to be overlaid must be under the DSI to be loaded and must be in the same internal order. • The number of key specification records in the DSI to be loaded does not match the DSI on the system.
C4C6-C5C6[1]	Object destroyed (I/O processing has not yet started except for the Create and Load command).
C4C7	Invalid object • The field descriptor table field in the DS does not match the one to be loaded. • The DS key specification field in the DSI does not match the one to be loaded.
C4C8-C5C8	A serious problem that L/D cannot handle occurred. An unexpected condition occurred in L/D for an unknown reason. The reason could be a bad object, a hardware error, or a software error. Information about this error is logged. The RD number in the feedback record indicates the RD that L/D was working on when the error occurred but that RD may not be the reason for the error. If the reason for the error is a bad object, it could be the object in the current RD or any object in the next three RDs. If the error occurred during loading of a data base object, the bad object could also be any previous DS or DSI immediately preceding the current RD. The LUD is marked partially damaged and the partial system object damage set exception is signaled. To recover from this error, a Modify LUD (reset, de-activate, and vary off) must be issued.
C4C9	Object name, type, and subtype of the object to be loaded does not match the one pointed to by the user-supplied pointer.
C4CA[1]	DS or DSI has an active cursor in use during the Load command (I/O processing has not started).
C4CB	Not used.

[1]If this error occurs on an RD that has a Create and Load command, all RDs before this one have been processed. The invalid lock, damaged object, or destroyed object is not the one associated with the Create and Load RD; instead either the user profile ownership is being put in, or the context addressability is being put in for the object just created.

Error Code (hex)	Definition
C5CC	All objects (DS or DSI) in this network have been loaded but not properly linked together. The RD number field in the feedback record contains the RD number of the object in the network that is in error. This is a serious machine problem that occurs when data base objects are damaged but not marked as damaged. All the DS(s) within the network and any DS(s) immediately preceding the network are damaged when the Modify Logical Unit Description (reset) instruction is issued.
C4CD-C5CD[1]	Object is damaged (I/O processing has not yet started except for the Create and Load command).
C4CE	Invalid version level on object to be loaded or created.
C4CF	Unmodified Request I/O instruction was not returned while L/D was in a recoverable error mode. The RDs were modified in the SSR.
C4D0	Object cannot be dumped (I/O processing has not started).
C4D1	The reserved field of the first RD in this SSR was not set to hex 00 for this Request I/O instruction.
D4FF	An attempt has been made to load an object that has an invalid L/D object descriptor.

For error codes not listed in this section, see the error codes listed for the individual devices in Chapter 23 of this manual.

Note: Bits 4 through 7 of the status field for all errors listed in this section indicate the following:

Hex 4 – The object being processed will not be damaged by the L/D function if a Modify Logical Unit Description (reset) instruction is issued.

Hex 5 – A cleanup is done on the object being loaded or created if a Modify Logical Unit Description (reset) instruction is issued.

If the error occurs on a Load command, the object being overlaid on the system will be damaged. If the error occurs on a Create and Load command, the space allocated for the object being created is destroyed.

The normal Request I/O and the Request I/O (continue) instructions cannot be issued.

Recoverable Errors

When a recoverable error occurs (for example, an EOV (end of volume), EOT (end of tape), EOF (end of file), or suspended), the L/D function returns the feedback record with the status of the error. The L/D function does not process any other normal Request I/O instructions until a Modify LUD (reset) instruction or a Request I/O (continue) instruction is issued.

If the user can correct the error (for example, by positioning the L/D medium on the next volume or at the beginning of the file), the same normal Request I/O instruction that encountered the error must be returned unmodified and ahead of (lower value in the request priority field of the SSR header) all other normal Request I/O instructions that have been previously issued. A Request I/O (continue) instruction must then be issued.

A Modify LUD (reset) instruction must be issued when the user cannot correct the EOV, EOT, or EOF error.

Use the Request I/O (continue) instruction to cause the L/D function to finish processing the next normal Request I/O instruction from the point where the recoverable error occurred.

The following is a list of the recoverable error codes returned in the status field and the definition of those error codes.

Error Code (hex)	**Definition**
8417	No error in the Request I/O instruction but EOT was reached while dumping to tape.
C416-C516	Device EOF
C417-C517	Device EOV or EOT
C4DF	This Request I/O instruction was suspended on an RD boundary before all RDs were processed.

For error hex 8417 the same Request I/O instruction should not be reissued but a Request I/O (continue) must be issued if more Request I/O instructions are to be processed on the next volume. For all other recoverable errors, the same Request I/O instruction must be reissued along with a Request I/O (continue) instruction.

Other Errors

Other types of errors only indicate status. They do not terminate the L/D function or take the L/D function into or out of error mode.

The following is a list of the other error codes returned in the status field and the definition of those error codes.

Error Code (hex)	**Definition**
0400	No error on a Request I/O instruction
0408	No error on a Request I/O (continue) instruction
4409-4509	Partially processed request because of a Modify Logical Unit (reset) instruction. If bits 4-7 equal hex 4, the object was not damaged. If bits 4-7 equal hex 5, the object may have been damaged by the cleanup procedure. This depends on the command being processed.
440A	Unprocessed request because of Modify Logical Unit Description (reset) instruction.
4489	Request I/O (continue) instruction issued when the L/D function is not in a recoverable error mode.

Processing a Modify LUD (Reset) Instruction

During the processing of a Modify LUD (reset) instruction after a severe error or a recoverable error, the L/D function executes a cleanup procedure, if necessary, on the object that was being processed when the error occurred; the L/D function then returns a feedback record for each pending Request I/O instruction (unprocessed) and completes processing of the Modify LUD (reset) instruction. The L/D function is then ready to process additional Request I/O instructions.

The user must reposition the L/D media at the correct starting location after a Modify LUD (reset) instruction is processed.

Cleanup Procedure

The cleanup procedure is performed on an object that is being processed when a Modify Logical Unit Description (reset) instruction is issued and an error occurs. The cleanup procedure depends on the command being processed (Load or Create and Load).

If a Load command is being processed and the object was partially loaded at the time the error occurred, the object is flagged and logged as damaged. The object is considered damaged because the contents of the data portion of the object is unknown.

If a Create and Load command is being processed and a space for an object was already allocated when the error occurred, that space is destroyed.

If a Modify LUD (reset) instruction is issued before a network has been completely restored, all DSs in the network and any DSs immediately before the network are left in the damaged state. The RD number returned in the feedback record indicates the RD that is in error. By using the RD number and the following rules, the user can determine whether any DSs are damaged.

- If the RD that is in error was restoring a DS, all DSs immediately before this RD are damaged.
- If the RD that is in error was restoring a DSI, all the DSs in that network and all the DSs immediately before that network are damaged.
- If the RD that is in error was creating and loading any object and the previous RD contained a DS or DSI, then damage to the DSs may occur for previous RDs according to the first two rules.

  A damaged object event is signaled for the previously damaged DSs, but no DSI invalidated event is signaled for the invalidated DSIs.

LOAD/DUMP EVENTS

The following table gives the event that can be signaled by the L/D function. See Chapter 21, *Event Specifications* in this manual for a detailed description of the events and event identification.

0002 Authorization
- 0101 Authorization violation
- 0301 Special authorization required

0008 Data Space Index
- 0301 Data space index invalidated

000B Logical Unit Description
- 0901 Request I/O completed (signaled only if indicated by the request ID in the SSR)

000C Machine Resource
- 0201 Machine auxiliary storage exceeded

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

LOAD/DUMP AUTHORITY

The Request I/O instruction provides the required authority checking to determine whether the proper authorization is available for the L/D function. If the proper authority is not available, an exception is signaled. The following chart shows the possible authorizations for each L/D command; only one type of authorization is needed to satisfy the requirements for the L/D command.

L/D Commands	Authorization
Dump	• Unrestricted dump (special authorization)
	• Restricted dump (special authorization) and all object authority (special authorization)
	• Restricted dump (special authorization), retrieve, object management (if DS object), and space (if object has associated space)
Load	• Unrestricted load (special authorization)
	• Restricted load (special authorization) and all object authority (special authorization)
	• Restricted load (special authorization) and object control
	• Restricted load (special authorization) and ownership
Create and Load	• Unrestricted load (special authorization)
	• Restricted load (special authorization)
Read Object ID	• Unrestricted load (special authorization)
	• Restricted load (special authorization)
Set User Profile	• Unrestricted load (special authorization)
	• All object Authority (special authorization)
	• Insert
Set Context	• Unrestricted load (special authorization)
	• All object authority (special authorization)
	• Insert

LOAD/DUMP DATA BASE NETWORKS

An L/D data base network consists of one or more data space indexes and all the data spaces associated with each data space index. When a network is dumped, the L/D function saves the information that links a network together. Then, when the network is loaded, the L/D function restores the information that links the network together.

The following rules apply during the use of L/D data base networks.

- Other (nonnetwork) objects can be processed in the same Request I/O instruction.
- Networks are supported by all the L/D commands.
- A DSI cannot be dumped or loaded alone. All its associated DSs must be dumped or loaded along with the DSI.
- When a DSI within a network is loaded, the links to the DSs within the same Request I/O instruction are connected.
- Any DSI that is over a DS that is being loaded is invalidated if the DSI is not loaded in the same Request I/O instruction.
- An event is generated when a DSI is invalidated.
- All DSs must appear immediately before all associated DSIs in a network.

Intertwined networks that are on the media may be loaded or created and loaded as long as there are no nondata base objects between them.

The following is an example of a network that was dumped on the media:

DS-A
DS-B
DSI-1 (over DS-A)
DSI-2 (over DS-A and DS-B)
DSI-3 (over DS-B)

The two intertwined networks in the previous example can be loaded by a Request I/O instruction as follows:

DS-A
DS-B
DSI-1 (over DS-A)
DSI-3 (over DS-B)

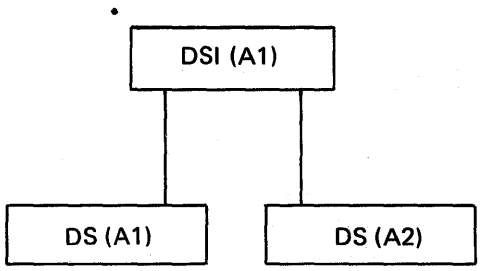
The L/D function updates the two networks (DSI-1 over DS-A and DSI-2 over DS-B) after DSI-3 is loaded.

- An active cursor (in use) may not be over a DS or DSI when the DS or DSI is being loaded.
- When a DS is loaded, all DSIs associated with it must not be in use, damaged, or destroyed.
- When a DSI is loaded, the DS key specification field in the DSI to be overlaid must match the key specification field in the DSI to be loaded.
- A DSI is not loaded if any of the DSs associated with it are created and loaded.
- When a DSI is loaded, the same DSs must be associated with the DSI to be loaded and the DSI to be overlaid. These DSs must also be in the same internal order in the DSI.

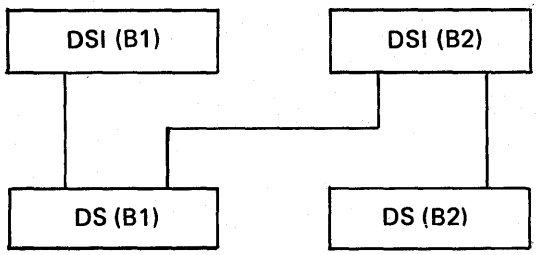
An example of ordering objects in an SSR is as follows. The objects to be processed are:

- Space object
- Network A
- Data space (nonnetwork)
- Program
- Network B

Network A



Network B



A way of ordering in the SSR is:

1. Space object
2. DS (A1) } Network A
3. DS (A2) } Network A
4. DSI (A1) } Network A
5. Data space (nonnetwork)
6. Program
7. DS (B1) } Network B
8. DS (B2) } Network B
9. DSI (B1) } Network B
10. DSI (B2) } Network B

LOAD/DUMP PERFORMANCE

To achieve maximum performance from the L/D function, the user should follow these guidelines.

- Load the objects in the same order as they were dumped. If objects A, B, and C were dumped in alphabetic order, then objects A, B, and C should be loaded in alphabetic order. This procedure minimizes the number of EOV, EOF, and EOT conditions to be processed.

- When processing objects in a dedicated mode, the user can increase performance by taking advantage of the asynchronous characteristics of the L/D function.

  The greatest performance gain occurs when many large objects are to be processed on a slow L/D device. The idea is to keep the machine busy while I/O transfer is taking place. For example, if 500 objects are to be processed, have 50 objects in 10 Request I/O instructions. Build the first Request I/O instruction and issue it soon as possible. This starts the I/O operation. Then build and issue the second Request I/O instruction.

  The user can then look at the queue to see whether the first Request I/O instruction has completed. If the first Request I/O instruction has not completed, the user should not wait for the feedback record; instead, the user should build and issue the third Request I/O instruction.

  Each time a Request I/O instruction is issued, check to see whether one of the previous Request I/O instructions has completed, but never wait in the queue.

- Put the RDs in the first 64 K bytes of the SSR.

- One large object can be processed faster than a number of small objects; therefore, keep the data in large objects.

LOAD/DUMP INTERRUPTED FOR DATA INTERCHANGE

It is possible for an L/D session to be interrupted by data interchange if the load/dump device that is being used supports interruptible load/dump operations. The following steps should be followed if the L/D function is to be interrupted for data interchange.

1. Modify the LUD to the suspend session.

2. Modify the LUD to normal mode. At this time, a copy of the information necessary (L/D volume label, L/D header label, and L/D interrupt location) for repositioning (step 5) is put in the device-specific area of the LUD.

3. Modify the LUD to the active session.

4. Issue Request I/O instructions for normal processing.

5. The repositioning and verification to start L/D processing again can be done as the last Request I/O instruction in data interchange mode or can be done any time before step 9. The information needed for this step was saved in the LUD at the completion of step 2. No checking of this sequence protocol is done by the L/D function and it is the user's responsibility to ensure that the media is always positioned correctly.

6. Modify the LUD to a quiesce session.

7. Modify the LUD to an L/D mode.

8. Modify the LUD to an active session.

9. Issue the L/D Request I/O instructions. It may be necessary to issue the same Request I/O instruction (with unmodified RDs) if the L/D function was suspended before completing the Request I/O instruction. A Request I/O (continue) instruction must then be issued.

LOAD/DUMP OBJECT AVAILABILITY

Sometimes objects processed by the L/D function are unavailable to the user for reading or modifying. The term *unavailable* is not related to locks but is a means by which the system avoids deadlocks (system hangups) and provides machine integrity.

The length of time and the extent to which the objects are unavailable is dependent on the commands, errors encountered, and LUD states.

Commands

Generally, before I/O processing starts and after the Request I/O instruction has been checked for errors, objects are made unavailable. After an object has been processed, it is made available. Therefore, the fewer the number of objects to be processed in a Request I/O instruction, the shorter the time the objects are unavailable.

Dump Command

Objects to be dumped are made unavailable for modifying before the Request I/O instruction starts processing. After each object is dumped (RD is completed), it becomes available again.

Load Command

Objects to be loaded and DSIs (even if they are not to be loaded) associated with any DSs to be loaded are made unavailable for reading and modifying before the process starts. Space objects, indexes, and data spaces that are not in a network become available after each object is loaded (RD is completed). The objects within a network (DSs and DSIs that are being replaced and all other DSIs associated with any DSs that are being replaced) become available after the last DSI in the network is loaded.

Create and Load Command

Newly created and loaded objects are always available because they are not on the system at the time they are created and loaded.

Set User Profile and Set Context Commands

The user profiles and contexts associated with these commands become unavailable for reading and modifying after the object associated with the Create and Load command is created. Once ownership and addressability are inserted, the user profiles and the context become available. This procedure is done for each Create and Load command as it is processed.

Read Object ID Command

All objects are available for this command.

Errors Encountered

When a severe or recoverable error is encountered, the object associated with the RD that has the error and any unprocessed objects remain unavailable. If a network is being restored and the error occurred on an RD within that network, all objects associated with that network are unavailable even though they have been processed. The user then has control and must either correct the recoverable error or issue a Modify LUD (reset) instruction. It is extremely important that the user's process does not attempt to read or modify any unavailable objects being restored or to modify any unavailable objects being saved. If this happens, the user's process becomes deadlocked. User profiles and contexts are always available when errors are encountered.

Logical Unit Description (LUD) States

The two LUD states in which the L/D function may have objects unavailable and in which the user has control (in a deadlock) are the active and suspend states. When the LUD is in the reset, quiesce, or de-activate state, all objects are available.

Active State

Objects can be unavailable and the user given control when an error condition exists. See the first paragraph under *Errors Encountered*, earlier in this chapter.

Suspend State

If the L/D function is to be suspended before a Request I/O instruction has completed processing, the objects and/or networks that have not yet been processed are unavailable. User profiles and contexts are always available when they are in a suspend state. The user must use extreme caution to avoid a deadlock when this situation occurs. Should any unavailable objects be accessed for modification on a save operation or for reading and modification on a restore operation, a deadlock could occur. A Modify LUD (reset) or Modify LUD (de-activate) instruction to the L/D function makes these objects available.

Reset State

When a reset operation is completed, all objects that were unavailable are made available.

Quiesce State

Once the quiesce process has completed successfully, all Request I/O instructions issued have also been completed; therefore, all objects are available. If an error occurs while the L/D function is processing a Request I/O instruction and is attempting to obtain the quiesce, the L/D function aborts the quiesce process and puts the LUD into an active state so that the error can be handled.

De-activate State

After the de-activate process is completed, all objects are available. If the L/D function was in an active state and the user issued a Modify LUD (de-activate) instruction, an implicit quiesce would be issued to the L/D function before the Modify LUD (de-activate) instruction is issued. If the L/D function was in a suspend state when a Modify LUD (de-activate) instruction is issued, an implicit reset operation would be issued to the L/D function before the Modify LUD (de-activate) instruction is issued.

Notes:

1. If a space object is unavailable for reading or modifying, some arithmetic and computational operations can still be performed.
2. A cursor cannot be activated or de-activated over a DSI while the DSI is being dumped.

LOAD/DUMP OBJECT STATUS AFTER A SYSTEM FAILURE

In the event of a system failure, objects being loaded or created and loaded may be damaged. Objects being dumped are not damaged. Any objects that are damaged should be destroyed; then they should be created and loaded after the system is restored.

The object being restored at the time of a system failure is always left in a damaged state. If a system failures should occur during the restore of a data base network, all DSs (within the network and immediately preceding the network) that have been restored will be damaged. All DSIs associated with the damaged DSs are invalidated, but no event indicating that they are invalid is signaled. The DSIs that have been restored are not damaged, but they are of little use because the associated DSs are damaged.

Appendix A. Machine Initialization

This appendix describes the various functions used for machine initialization.

MACHINE INITIALIZATION

Machine initialization is the means by which the machine performs the functions required to support the System/38 instructions and initiate a machine interface process. The machine initialization function:

- Provides a consistent well-defined interface for initialization of the machine.
- Provides a means for the machine to initialize itself automatically without communicating with the machine console.
- Provides a means to implicitly create the initial process from user-provided data.

MACHINE INITIALIZATION TERMS AND DEFINITIONS

The following are the terms associated with the machine initialization functions and their definitions:

- *Power on* is the activation of the machine power supply by a manual power on or an automatic power on (auto-IMPL). Power on, whether manual or automatic, always performs an implicit initiation of machine processing.
- *IMPL/IMPLA* sequences are the functions performed by the machine necessary to initiate machine processing. IMPL/IMPLA always refers to the use of internal storage as the data source for the initial loading of microcode.
- *IMPL/IMPLA halt* causes the machine to enter the check stop state as the result of a machine malfunction while the IMPL/IMPLA sequence is executing.
- *Machine-to-programming transition* is the mechanism whereby a machine function implicitly creates an initial process as the final function of the machine initialization sequence. The setting of the second rotary switch on the operator/service panel is used to determine whether an AIPL or IPL is to be performed.
- *AIPL* (alternate initial process load) is the loading of the source data from external storage data media (load/dump).
- *IPL* (initial process load) is the loading of the encapsulated data from the machine internal storage.

MACHINE INITIALIZATION OVERVIEW

The machine provides the user, through the operator/service panel, the ability to perform the function of installing and initiating a process independent of the existence of an initiated process.

Installation of a process into System/38 is done by using the AIPL (alternate initial process load) function. AIPL always performs installation and initialization of a process as a contiguous operation.

Initialization of a process by the machine can also be done independent of the AIPL function. In this case, the objects required to initialize a process exist within the machine as encapsulated objects. This function is called IPL (initial process load).

The machine initialization function also provides a means of notifying the machine interface about the status of the machine after it has been initialized. This information resides in the machine as a machine attribute called the MISR (machine initialization status record).

The machine initialization function is divided into two subfunctions that are performed in sequence. The first subfunction is called IMPL (initial microprogram load) or IMPLA (initial microprogram load abbreviated). These subfunctions cause the machine to automatically initialize itself. The second subfunction of machine initialization is the machine-to-programming transition.

MACHINE-TO-PROGRAMMING TRANSITION

The machine-to-programming transition function provides a means for the machine to initiate a process independent of the execution of the Initiate Process instruction. This function is always performed as the final part of the machine initialization function.

The process that is initiated is defined from the data that exists in one of the following forms:

- AIPL machine interface source data
- IPL machine interface encapsulated data

AIPL Source Data

The AIPL source data is loaded from the primary load/dump device and must exist on the load/dump device as either a character scalar (data interchange) or a space object dumped by load/dump.

The source data required to perform an AIPL consists of the following templates:

- User profile template
- Program template
- Process definition template

The templates are encapsulated into the corresponding objects so that the final product is an initiated process. If any abnormal condition occurs while the AIPL templates are being encapsulated, the machine-to-programming transition function terminates. Termination causes an IMPL/IMPLA halt to occur.

AIPL User Profile Template

The format for the AIPL user profile template is shown in Chapter 7. Because the user profile that is created does not have an owning user profile created by the machine, it owns itself. If a user profile exists in the machine with the same name as is defined on the user profile template, the machine uses the user profile currently in the machine. If AIPL attempts to use the user profile in the machine and the user profile is damaged, it is destroyed, and a new one is created according to the input user profile template. Because addressability to the created user profile is also placed in the machine context, the name of the user profile can be obtained by using the Materialize Context instruction. If the Materialize Context instruction is used to obtain the name of the user profile, the user must know the name of all the user profiles in the machine context so that the user profile name created by the machine can be selectively determined. If the machine context is damaged during AIPL, the machine removes all entries in the machine context. This is noted in the MISR.

AIPL Program Template

The AIPL program template source data defines the program that receives control when the initial process is initiated. The format and description of the program template are defined in Chapter 8. The machine only encapsulates one problem program per AIPL.

AIPL Process Definition Template

The AIPL process definition template source data defines the process to be initiated. The format and description of the template are defined in Chapter 11. The AIPL process definition template should contain no resolved system pointers. The machine fills in the system pointer to the user profile created from the AIPL user profile template and the system pointer to the program created from the AIPL program template. The process control attribute (process type) of the AIPL process definition template should be set to an independent process. If the process type does not indicate an independent process, the machine sets it to an independent process before initiating the process.

All space objects required for the initiation of a process are created by the machine-to-programming transition function. The process automatic storage area, process static storage area, and process control space are created with temporary extendable attributes. System pointers to these spaces may be obtained from the MISR with the Materialize Machine Attributes instruction.

IPL Encapsulated Data

The IPL function performs an implicit initiation of a process from data residing in the machine as encapsulated data. The data is a process definition template for the Initiate Process instruction. This process definition template is a machine attribute called the initial process definition template. The IPL function uses the process definition template to initiate the initial process. The process control space, process static storage area, and the process automatic storage area for the IPL process are created by the machine. System pointers to these spaces are saved in the MISR and may be materialized with the Materialize Machine Attributes instruction.

AIPL/IPL MACHINE ATTRIBUTES

The AIPL/IPL machine attribute provides a way to save, within the machine, information required to perform the machine-to-programming function. It also provides a way of collecting the machine status associated with events that occur before a process is initiated. The attributes are called initial process definition template and machine initialization status record.

Initial Process Definition Template

The initial process definition template is used to save, within the machine, those parameters and encapsulated objects required for the machine to perform an IPL. The format of the initial process definition template is defined in Chapter 11. The Modify Machine Attribute instruction, used to save the initial process definition template in the machine, is defined in Chapter 19.

A program may materialize the initial process definition template by using the Materialize Machine Attribute instruction. This instruction is defined in Chapter 19.

Machine Initialization Status Record Machine Attribute

The machine initialization status record is a machine attribute that provides a means of passing to a program the status of the machine as collected during machine initialization. The status record contains the machine event related information that is normally passed to the machine interface by the events after an AIPL or IPL. A program may materialize the machine initialization status record at any time. The status record is created by the machine. A process may delete the status record by using the Modify Machine Attribute instruction defined in Chapter 19.

Appendix B. 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 and no carry, zero and carry, not zero and carry,
S, NS	Signaled, not signaled
DE, I	Exception deferred, exception ignored
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}

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}

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)}

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 either 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}

DE ::= DESC-{defer}

DQ ::= DESC-{dequeued}

NDQ ::= DESC-{not dequeued}

E ::= DESC-{equal}

H ::= DESC-{high}

I ::= DESC-{ignore}

L ::= DESC-{low}

M ::= DESC-{XED}

N ::= DESC-{negative}

NE ::= DESC-{not equal}

NS ::= DESC-{not signaled}

NZ ::= DESC-{not zero}

O ::= DESC-{ones}

P ::= DESC-{positive}

S ::= DESC-{signaled}

U ::= DESC-{unequal}

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
Branch	B	1011	1	–	BT	–	–	2-5
Compute Array Index	CAI	1044	4	–	BINV, BINS.3.	–	–	2-14
Call Internal	CALLI	0293	3	–	I-ENT PT, {ARG LIST|NULL}, I-PTR	–	–	9-7
Call External	CALLX	0283	3	–	PROGRAM, {ARG LIST|NULL}, {IDL|NULL}	–	–	9-4
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.	–	–	2-15
Compare Bytes Left-Adjusted	CMPBLA	10C2	2+B	BR|IND	{CHARS|NUMERICS}.2., {BT.3.|IT.3.}	H, L, E	{B|I}	2-6
Compare Bytes Left-Adjusted With Pad	CMPBLAP	10C3	3+B	BR|IND	{CHARS|NUMERICS}.3., {BT.3.|IT.3.}	H, L, E	{B|I}	2-8
Compare Bytes Right-Adjusted	CMPBRA	10C6	2+B	BR|IND	{CHARS|NUMERICS}.2., {BT.3.|IT.3.}	H, L, E	{B|I}	2-9
Compare Bytes Right-Adjusted With Pad	CMPBRAP	10C7	3+B	BR|IND	{CHARS|NUMERICS}.3., {BT.3.|IT.3.}	H, L, E	{B|I}	2-11
Compare Numeric Value	CMPNV	1046	2+B	BR|IND	NUMERICS.2., {BT.3.|IT.3.}	H, L, E	{B|I}	2-12
Compare Pointer for Space Addressability	CMPPSPAD	10E6	2+B	BR|IND	{SPP|D-PTR}, {NUMERICV|CHARV|C-ARRAY|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}	4-3
Copy Bytes Left-Adjusted	CPYBLA	10B2	2	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}	–	–	2-23
Copy Bytes Left-Adjusted With Pad	CPYBLAP	10B3	3	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}.2.	–	–	2-24
Copy Bytes Overlap Left-Adjusted	CPYBOLA	10BA	2	–	{NUMERICV|CHARV}.2.	–	–	2-25
Copy Bytes Overlap Left-Adjusted With Pad	CPYBOLAP	10BB	3	–	{NUMERICV|CHARV}.2., {NUMERICS|CHARS}	–	–	2-26
Copy Bytes Right-Adjusted	CPYBRA	10B6	2	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}	–	–	2-28
Copy Bytes Right-Adjusted With Pad	CPYBRAP	10B7	3	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}.2.	–	–	2-29
Copy Bytes Repeatedly	CPYBREP	10BE	2	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}	–	–	2-27
Copy Bytes With Pointers	CPYBWP	0132	2	–	{CHARV|PTR}, {CHARV|PTR|NULL}	–	–	3-4
Copy Data Space Entries	CPYDSE	048F	3	–	CURSOR,SPP,CURSOR	–	–	16-4
Copy Hex Digit Numeric to Numeric	CPYHEXNN	1092	2	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}	–	–	2-30
Copy Hex Digit Numeric to Zone	CPYHEXNZ	1096	2	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}	–	–	2-31
Copy Hex Digit Zone to Numeric	CPYHEXZN	109A	2	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}	–	–	2-32
Copy Hex Digit Zone to Zone	CPYHEXZZ	109E	2	–	{NUMERICV|CHARV}, {NUMERICS|CHARS}	–	–	2-33
Copy Numeric Value	CPYNV	1042	2+[B]	[BR|IND]	NUMERICV, NUMERICS, [BT.3.|IT.3.]	P, N, Z	[B|I, R]	2-34
Create Access Group	CRTAG	0366	2	–	AG, SPP	–	–	13-1
Create Controller Description	CRTCD	0496	2	–	CD, SPP	–	–	17-1
Create Cursor	CRTCR	044A	2	–	CURSOR, SPP	–	–	16-8
Create Context	CRTCTX	0112	2	–	CONTEXT, SPP	–	–	3-5
Create Duplicate Object	CRTDOBJ	0327	3	–	S-PTR, SPP, S-PTR	–	–	13-4

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Create Data Space	CRTDS	045A	2	–	DATA SPACE, SPP	–	–	16-14
Create Data Space Index	CRTDSINX	046A	2	–	DS-INDEX, SPP	–	–	16-19
Create Independent Index	CRTINX	0446	2	–	INDEX, SPP	–	–	6-1
Create Logical Unit Description	CRTLUD	049A	2	–	LUD, SPP	–	–	17-8
Create Network Description	CRTND	049E	2	–	ND, SPP	–	–	17-14
Create Process Control Space	CRTPRCS	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 Character to Hex	CVTCH	1082	2	–	CHARV, CHARS	–	–	2-16
Convert Character to Numeric	CVTCN	1083	3	–	NUMERICV, CHARS, CHARS[7]	–	–	2-17
Convert External Form to Numeric Value	CVTEFN	1087	3	–	NUMERICV, CHARS, {CHARS[3]|NULL}	–	–	2-19
Convert Hex to Character	CVTHC	1086	2	–	CHARV, CHARS	–	–	2-21
Convert Numeric to Character	CVTNC	10A3	3	–	CHARV, NUMERICS, CHARS[7]	–	–	2-22
Disable Event Monitor	DBLEVTMN	0399	1	–	CHARS[48]	–	–	15-2
Data Base Maintenance	DBMAINT	0482	2	–	{DATA SPACE|DS-INDEX}, CHARS[1],BINS|NULL	–	–	16-26
De-activate Cursor	DEACTCR	0401	1	–	CURSOR	–	–	16-29
De-activate Program	DEACTPG	0225	1	–	PROGRAM|NULL	–	–	9-8
Delete Data Space Entry	DELDSEN	0481	1	–	CURSOR	–	–	16-30
Delete Program Observability	DELPGOBS	0211	1	–	PROGRAM	–	–	8-6
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 Controller Description	DESCD	04A1	1	–	CD	–	–	17-23
Destroy Cursor	DESCR	0429	1	–	CURSOR	–	–	16-32
Destroy Context	DESCTX	0121	1	–	CONTEXT	–	–	3-8
Destroy Data Space	DESDS	0421	1	–	DATA SPACE	–	–	16-33
Destroy Data Space Index	DESDSINX	0425	1	–	DS-INDEX	–	–	16-34
Destroy Independent Index	DESINX	0451	1	–	INDEX	–	–	6-5
Destroy Logical Unit Description	DESLUD	04A9	1	–	LUD	–	–	17-24
Destroy Network Description	DESND	04AD	1	–	ND	–	–	17-26
Destroy Process Control Space	DESPCS	0311	1	–	PCS	–	–	11-4
Destroy Program	DESPG	0221	1	–	PROGRAM	–	–	8-7
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]	2-36
Divide with Remainders	DIVREM	1074	4+[B]	[BR|IND]	NUMERICV, NUMERICS.2., NUMERICV	P, N, Z	[B|I, S, R]	2-38
Enable Event Monitor	EBLEVTMN	0369	1	–	CHARS[48]	–	–	15-4
Edit	EDIT	10E3	3	–	CHARV, NUMERICS, CHARS	–	–	2-40
End	END	0260	0	–	–	–	–	9-10
Enqueue	ENQ	036B	3	–	QUEUE, CHARS, SPP	–	–	12-9
Ensure Data Space Entries	ENSDSEN	0499	1	–	CURSOR	–	–	16-35
Ensure Object	ENSOBJ	0381	1	–	S-PTR	–	–	13-8
Exchange Bytes	EXCHBY	10CE	2	–	{CHARV|NUMERICV}.2.	–	–	2-48
Extract Magnitude	EXTRMAG	1052	2+[B]	[BR|IND]	NUMERICV, NUMERICS, [BT.3.|IT.3.]	P, Z	[B|I, S]	2-51

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
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
Initiate Process	INITPR	0324	4	–	PCS, SPP, {ARG-LIST|NULL}, {SPP|NULL}	–	–	11-5
Insert Data Space Entry	INSDSEN	0483	3	–	CURSOR, CHARV[7], SPP	–	–	16-36
Insert Independent Index Entry	INSINXEN	04A3	3	–	INDEX, SPP.2.	–	–	6-8
Insert Sequential Data Space Entries	INSSDSE	0487	3	–	CURSOR,SPP,SPP	–	–	16-38
Lock Object	LOCK	03F5	1	–	SPP	–	–	14-1
Lock Space Location	LOCKSL	03F6	2	–	SPP,CHARS[1]	–	–	14-4.1
Materialize Access Group Attributes	MATAGAT	03A2	2	–	SPP, AG	–	–	13-9
Materialize Authority	MATAU	0153	3	–	SPP, S-PTR, {USER PROFILE|NULL}	–	–	7-8
Materialize Authorized Objects	MATAUOBJ	013B	3	–	SPP, USER PROFILE, CHARS[1]	–	–	7-10
Materialize Authorized Users	MATAUU	0143	3	–	SPP, S-PTR, CHARS[1]	–	–	7-12
Materialize Controller Description	MATCD	04B3	3	–	SPP, CD, CHARS[2]	–	–	17-27
Materialize Cursor Attributes	MATCRAT	043B	3	–	SPP, CURSOR, CHARS[1]	–	–	16-40
Materialize Context	MATCTX	0133	3	–	SPP, {CONTEXT|NULL}, CHARS	–	–	3-9
Materialize Data Space Attributes	MATDSAT	0437	3	–	SPP, DATA SPACE, CHARS[1]	–	–	16-43
Materialize Data Space Index Attributes	MATDSIAT	0433	3	–	SPP, DS-INDEX, CHARS[1]	–	–	16-45
Materialize Exception Description	MATEXCPD	03D7	3	–	SPP, EXCP-DESC,CHARS[1]	–	–	10-1
Materialize Invocation	MATINV	0516	2	–	SPP.2.	–	–	18-3
Materialize Independent Index Attributes	MATINXAT	0462	2	–	SPP, INDEX	–	–	6-10
Materialize Logical Unit Description	MATLUD	04BB	3	–	SPP, LUD, CHARS[2]	–	–	17-31
Materialize Machine Attributes	MATMATR	0636	2	–	SPP, CHARS[2]	–	–	19-2
Materialize Network Description	MATND	04BF	3	–	SPP, ND, CHARS[2]	–	–	17-35
Materialize Object Locks	MATOBJLK	033A	2	–	SPP, S-PTR	–	–	14-4
Materialize Program	MATPG	0232	2	–	SPP, PROGRAM	–	–	8-8
Materialize Process Attributes	MATPRATR	0333	3	–	SPP, {PCS|NULL}, CHARS [1]	–	–	11-13
Materialize Process Locks	MATPRLK	0312	2	–	SPP, {PCS|NULL}	–	–	14-6
Materialize Pointer	MATPTR	0512	2	–	SPP, {S-PTR|D-PTR|SPP|I-PTR}	–	–	18-5
Materialize Pointer Locations	MATPTRL	0513	3	–	SPP.2., BINS	–	–	18-7
Materialize Queue Attributes	MATQAT	0336	2	–	SPP, QUEUE	–	–	12-11
Materialize Resource Management Data	MATRMD	0352	2	–	SPP, CHARS[8]	–	–	13-11
Materialize Space Attributes	MATS	0036	2	–	SPP, S-PTR	–	–	5-5
Materialize Selected Locks	MATSELLK	033E	2	–	SPP, {S-PTR|SPP}	–	–	14-7.1
Materialize System Object	MATSOBJ	053E	2	–	SPP, S-PTR	–	–	18-9
Materialize User Profile	MATUP	013E	2	–	SPP, USER PROFILE	–	–	7-15
Monitor Event	MNEVT	0371	1	–	SPP	–	–	15-6
Modify Addressability	MODADR	0192	2	–	{CONTEXT|NULL}, S-PTR	–	–	3-12
Modify Automatic Storage Allocation	MODASA	02F2	2	–	{SPP|NULL}, BINS	–	–	9-10
Modify Controller Description	MODCD	04C3	3	–	CD, SPP, CHARS[2]	–	–	17-38
Modify Exception Description	MODEXCPD	03EF	3	–	EXCP-DESC, SPP, CHARS[1]	–	–	10-4
Modify Logical Unit Description	MODLUD	04CB	3	–	LUD, SPP, CHARS[2]	–	–	17-44
Modify Machine Attributes	MODMATR	0646	2	–	SPP, CHARS[2]	–	–	19-8

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Modify Network Description	MODND	04CF	3	–	ND, SPP, CHARS[2]	–	–	17-50
Modify Process Event Mask	MODPEVTM	0372	2	–	{BINV[2]|NULL}, {BINS[2]|NULL}	–	–	15-5
Modify Process Attributes	MODPRATR	0337	3	–	{PCS|NULL}, SPP, CHARS[1]	–	–	11-20
Modify Resource Management Control	MODRMC	0326	2	–	SPP, CHARS[8]	–	–	13-16
Modify Space Attributes	MODS	0062	2	–	S-PTR, BINS	–	–	5-8
Modify User Profile	MODUP	0142	2	–	USER PROFILE, SPP	–	–	7-17
Multiply	MULT	104B	3+[B]	[BR|IND]	NUMERICV, NUMERICS.2., [BT.3.|IT.3.]	P, N, Z	[B|I, S, R]	2-52
Negate	NEG	1056	2+[B]	[BR|IND]	NUMERICV, NUMERICS, [BT.3.|IT.3.]	P, N, Z	[B|I, S]	2-54
No Operation	NOOP	0000	0	–	–	–	–	2-56
Not	NOT	108A	2+[B]	[BR|IND]	CHARV, CHARS, [BT.3.|IT.3.]	Z, NZ	[B|I, S]	2-56
Or	OR	1097	3+[B]	[BR|IND]	CHARV,CHARS.2.,[BT.3.|IT.3.]	Z, NZ	[B|I, S]	2-58
Reclaim Lost Objects	RECLAIM	0686	2	–	SPP, CHARS[2]	–	–	19-10
Remainder	REM	1073	3+[B]	[BR|IND]	NUMERICV, NUMERICS.2., [BT.3.|IT.3.]	P, N, Z	[B|I, S]	2-59
Rename Object	RENAME	0162	2	–	S-PTR, CHARS	–	–	3-14
Request I/O	REQIO	0471	1	–	SPP	–	–	17-56
Resume Process	RESPR	0386	2	–	{PCS|NULL}, CHARS[1]	–	–	11-25
Retrieve Data Space Entry	RETDSEN	048A	2	–	SPP, CURSOR	–	–	16-49
Retrieve Event Data	RETEVTD	0375	1	–	SPP	–	–	15-10
Retrieve Exception Data	RETEXCPD	03E2	2	–	SPP, CHARS[1]	–	–	10-6
Retract Authority	RETRACT	0193	3	–	{USER PROFILE|NULL}, S-PTR, CHARS[2]	–	–	7-19
Retrieve Sequential Data Space Entries	RETSDSE	048B	3	–	SPP, CURSOR, SPP	–	–	16-50
Release Data Space Entries	RLSDSEN	048E	2	–	CURSOR, CHARS[1]	–	–	16-47
Remove Independent Index Entry	RMVINXEN	0484	4	–	{SPP|NULL}, INDEX, SPP.2.	–	–	6-13
Resolve Data Pointer	RSLVDP	0163	3	–	D-PTR, {CHARS[32]|NULL}, {S-PTR|NULL}	–	–	3-15
Resolve System Pointer	RSLVSP	0164	4	–	S-PTR, {CHARS[34]|NULL}, {S-PTR|NULL}, {CHARS[2]|NULL}	–	–	3-17
Return From Exception	RTNEXCP	03E1	1	–	SPP	–	–	10-8
Return External	RTX	02A1	1	–	{BINS|NULL}	–	–	9-12
Scale	SCALE	1063	3+[B]	[BR|IND]	NUMERICV, NUMERICS, BINS, [BT.3.|IT.3.]	P, N, Z	[B|I, S]	2-61
Scan	SCAN	10D3	3+[B]	[BR|IND]	{BINV|B-ARRAY}, CHARS.2., [BT.3.|IT.3.]	P, Z	[B|I]	2-63
Search	SEARCH	1084	4+[B]	[BR|IND]	{BINV|B-ARRAY}, {N-ARRAY|C-ARRAY}, CHARS|NUMERICS,BINS	P, Z	[B|I]	2-65
Set Access State	SETACST	0341	1	–	SPP	–	–	13-19
Set Argument List Length	SETALLEN	0242	2	–	ARG-LIST, BINS	–	–	9-13
Set Cursor	SETCR	048C	4	–	CURSOR, SPP, CHARV[16], {CHARV|NULL}, {CHARS|NULL}	–	–	16-53
Set Data Pointer	SETDP	0096	2	–	D-PTR, {NUMERICV|N-ARRAY|CHARV|C-ARRAY}	–	–	4-5
Set Data Pointer Addressability	SETDPADR	0046	2	–	D-PTR, {NUMERICV|N-ARRAY|CHARV|C-ARRAY}	–	–	4-6
Set Data Pointer Attributes	SETDPAT	004A	2	–	D-PTR, CHARS[7]	–	–	4-7
Set Instruction Pointer	SETIP	1022	2	–	I-PTR, F-BT	–	–	2-66
Set System Pointer From Pointer	SETSPFP	0032	2	–	S-PTR, {D-PTR|SPP|S-PTR|I-PTR}	–	–	4-12
Set Space Pointer	SETSPP	0082	2	–	SPP, {CHARV|C-ARRAY|NUMERICV|N-ARRAY|PTR|P-ARRAY	–	–	4-8
Set Space Pointer With Displacement	SETSPPD	0093	3	–	SSP, {CHARV|C-ARRAY|NUMERICV|N-ARRAY|PTR|P-ARRAY	–	–	4-9
Set Space Pointer From Pointer	SETSPPFP	0022	2	–	SPP, {S-PTR|D-PTR|SPP}	–	–	4-10
Set Space Pointer Offset	SETSPPO	0092	2	–	SPP, BINS	–	–	4-11

Operation Description	Mnemonic	Op Code	No. Opnds	Extender	Operand Syntax	Resulting Conditions	Optional Forms	Page
Signal Event	SIGEVT	0345	1	–	SPP	–	–	15-12
Signal Exception	SIGEXCP	10CA	2+[B]	[BR|IND]	SPP.2., [BT.2.|IT.2.]	I, DE	[B|I]	10-10
Sense Exception Description	SNSEXCPD	03E3	3	–	SPP.3.	–	–	10-9.1
Store Parameter List Length	STPLLEN	0241	1	–	BINV	–	–	9-14
Store Space Pointer Offset	STSPPO	00A2	2	–	BINV, SPP	–	–	4-14
Subtract Logical Character	SUBLC	1027	3+[B]	[BR|IND]	CHARV, CHARS.2., [BT.3.|IT.3.]	[N]Z[N]C	[B|I, S]	2-67
Subtract Numeric	SUBN	1047	3+[B]	[BR|IND]	NUMERICV, NUMERICS.2., [BT.3.|IT.3.]	P, N, Z	[B|I, S, R]	2-69
Subtract Space Pointer Offset	SUBSPP	0087	3	–	SPP.2., BINS	–	–	4-15
Suspend Object	SUSOBJ	0361	1	–	S-PTR	–	–	13-22
Suspend Process	SUSPR	0392	2	–	{PCS|NULL}, CHARS[1]	–	–	11-26
Terminate Machine Processing	TERMMPR	0622	2	–	CHAR[2], {SPP|NULL}	–	–	19-12
Terminate Process	TERMPR	0332	2	–	{PCS|NULL}, CHARS[3]	–	–	11-28
Test Authority	TESTAU	10F7	3	–	{CHARV[2]|NULL}, S-PTR, CHARS[2]	–	–	7-21
Test Event	TESTEVT	10FA	2+[B]	[BR|IND]	SPP, {CHARS[48]|NULL}, [BT.2.|IT.2.]	S, NS	[B|I]	15-14
Test Exception	TESTEXCP	104A	2+[B]	[BR|IND]	SPP, EXCP-DESC, [BT.2.|IT.2.]	S, NS	[B|I]	10-13
Trace Instructions	TRINS	0552	2	–	PROGRAM, {SPP|NULL}	–	–	18-11
Trace Invocations	TRINV	0551	1	–	CHARS{4}	–	–	18-12
Test Bits under Mask	TSTBUM	102A	2+B	BR|IND	{CHARS|NUMERICS}.2., {BT.3.|IT.3.}	Z, O, M	{B|I}	2-72
Test and Replace Characters	TSTRPLC	10A2	2	–	CHARV, CHARS	–	–	2-71
Unlock Object	UNLOCK	03F1	1	–	SPP	–	–	14-10
Unlock Space Location	UNLOCKSL	03F2	2	–	SPP,CHARS[1]	–	–	14-11.1
Update Data Space Entry	UPDSEN	0492	2	–	CURSOR, SPP	–	–	16-63
Verify	VERIFY	10D7	3+[B]	[BR|IND]	{BINV|B-ARRAY}, CHARS.2., [BT.3.|IT.3.]	P, Z	[B|I]	2-75
Wait On Event	WAITEVT	0344	4	–	SPP, CHARS, CHARS[8], CHARS[3]	–	–	15-16
Transfer Control	XCTL	0282	2	–	PROGRAM, {ARG LIST|NULL}	–	–	9-15
Transfer Object Lock	XFRLOCK	0382	2	–	PCS, SPP	–	–	14-8
Transfer Ownership	XFRO	01A2	2	–	USER PROFILE, S-PTR	–	–	7-24
Translate	XLATE	1094	4	–	CHARV, CHARS, {CHARS|NULL}, CHARS	–	–	2-73
Exclusive Or	XOR	109B	3+[B]	[BR|IND]	CHARV, CHARS.2., [BT.3.|IT.3.]	Z, NZ	[B|I, S]	2-49

Index

READER'S COMMENT FORM

GA21-9331-1

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.

☐ 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

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

No postage necessary if mailed in the U.S.A.

GA21-9331-1

Cut Along Line

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N. Y.

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold and tape | Please do not staple | Fold and tape

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM System/38 Functional Reference Manual (File No. S38-01) Printed in U.S.A. GA21-9331-1

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM System/38 Functional Reference Manual (File No. S38-01) Printed in U.S.A. GA21-9331-1