FILE COPY

HONEYWELL INTEROFFICE CORRESPONDENCE EPCOVRDEF

Date: April 15, 1986

Subject: Approved DPS6 Stage 3 System and Central Subsystem Family EPS-1

To: LIST

From: V. Morganti
Organization: Systems Engineering
HED: MA30
MS: 852A
HVN: 671-2778

Attached is the approved copy (Rev. E) of the DPS6 Stage 3 System and Central Subsystem Family EPS-1 (i.e, the non MRX family). It describes the functionality, performance and configurations of the different Central Subsystems.

Any comments or questions should be directed to the author V. Morganti.

V. Morganti

V. Morganti

[illegible]

[illegible] 671-6000

Honeywell HONEYWELL INFORMATION SYSTEMS		SPEC. NO. 60149740	SHEET 1	REV. E
LOC.		PROJECTION	CODE	
PREPARED BY J. Curley	DATE 10/4/82	TITLE Engineering Product Specification, Part 1 DPS6 Stage 3 Central Subsystem Family		
APPROVED BY	DATE			

REVISION RECORD

REV.	AUTHORITY	DATE	SIGNATURE	SHEETS AFFECTED
A	DRAFT.	10/4/82		All
B	BLCE7883	11/2/84	[illegible]	All
C	BLCF7540	8/1/85	[illegible]	ALL
D		12/11/85	[illegible]	-Rev. D Pages
E		4/30/86	[illegible]	Rev. E Pages

NOTE

This EPS-1, if not revised within one year, should be considered obsolete and therefore reference should be made to the appropriate product manual.

HONEYWELL INFORMATION SYSTEMS
SYSTEMS ENGINEERING - BOSTON

ENGINEERING PRODUCT SPECIFICATION (EPS - 1)

SUBSYSTEM LEVEL

TITLE: DPS6 Stage 3 CSS

VERSION: APPROVED DATE: 4/30/86

PRODUCT CALENDAR REFERENCE: ______

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

PAGE

CONTENTS

CONTENTS

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS

TABLES

TABLES

This page is intentionally blank.

SECTION 1 INTRODUCTION

1.1 DOCUMENT DEFINITION

This document is the Engineering Product Specification for the DPS6 Stage 3 Central Subsystem Family, consisting of the following:

o CR41E	The extended physical memory version of the CR41,
o M5XE	The extended physical and logical memory version of the M5X, featuring both a Standard Menory Management Unit (SMMU) and an Extended Menory Management Unit (EMMU),
o M6XE	The extended logical memory version of the M6X, which will be available in two speed versions, normal and slow.
o Dual M6X	The dual processor version of the M6X and
o Dual M6XE	The dual processor version of the M6XE.

1.1.1 16-Bit Expanded Physical Memory Processor Subsystems

The CR41E and M5XE models are being introduced to replace the current CR41- and M5X-based Stage 2.1 DPS6 Systems. These new processor systems are instruction compatible with their predecessors and feature expanded physical memory beyond the current limitation of two megabytes (internal processor performance remains the same). The CR41E supports the SMMU for a logical memory addressability of 2 MB. The M5XE supports both the SMMU and the EMMU for a logical memory addressability of either 2 MB or 32 MB. Processor features are as follows:

o CR41E:

- Physical memory capacity of up to 4 MB maximum;

- SMMU support for a logical memory addressability of 2 MB;

- Internal CIP, external SIP functionality;

- Field upgradable from DPS6/40, 45 models (only the CSS needs replacement); and

- Fully supported in M400 Release 3.1 and beyond.

o M5XE:

- Physical memory capacity of up to 8 MB maximum;

- SMMU and EMMU support for a logical memory addressability of either 2 MB or 32 MB;

- External cache, CIP, and SIP functionality;

- Monoprocessor configuration only;

- Field upgradable from DPS6/75 models (only the cache and CPU need replacement); and

- Fully supported in M400 Release 4.0.

1.1.2 32-Bit Enhanced Processor Subsystem

Increased logical addressing through the use of the Extended Memory Management Unit (EMMU), in both monoprocessor and dual processor M6XE configurations and dual processor 6X models, with SMMU only, will be introduced to supplement the current Model 6X- based Stage 2.1 DPS6/95 System. These new processor subsystems feature increased processing power through the use of multiprocessor techniques and all provide for field upgradability. Specific features are:

o M6XE:

- Physical memory capacity of up to 16 MB maximum;

- EMMU support for a logical memory addressability of 32 MB;

- Field upgradable from selected DPS6/95 models; and

- Fully supported in M400 Release 4.0.

- Available in two speed versions, normal and slowed down.

- o Multiprocessor Support:
  - Dual Tightly Coupled configurations of M6X and M6XE systems;
  - Up to 16 MB of physical address space capacity;
  - Field Upgrade Kits will be provided for selected DPS6/95 models; and
  - Fully supported in MOD400 Release 4.0.

1.1.3 I/O Support

All of the new Stage 3 systems offer the following attachments:

- o Communications:
  - NMLC-based Standard Radial Communications with FLAPs and/or FLAPless RS232 and RS422 adapters; and
  - Local Area Network Connections including OMNINET, CSMA/CD (ETHERNET), and, later on, IEEE Token Bus and Ring.
- o Mass Storage:
  - 9" FSD family of Winchester disk drives;
  - 9" removable disk drives, if available; and
  - Future 5-1/4" and 8" Winchester disk drives, when available.
- o Tape/Unit Record:
  - New lower-cost GCR/PE reel-to-reel drives;
  - Card Reader/Punches;
  - Floppy disk;
  - Serial and line printers; and
  - Reader/sorters.

1.1.4 Documentation Tree Structure

Document Number	Title	Responsibility
	Product Functional Description	
No. TBD	DPS6 Stage 3 PFD	SCMO Market Planning
	Product Functional Specification	
60149834	DPS6 Stage 3 PFS	SCMO Market Planning

1.1.5 CSS Model Designators

This EPS-1 uses engineering model designators. Table 1-1 shows the relationship between the engineering and the marketing model designators.

Table 1-1 Model Designators

ENGINEERING	MARKETING
M37	DPS 6/3X, 48
CR41, CR41E	DPS 6/4X, (except 48)
5X	DPS 6/7X
5XE	DPS 6/7XE
6X(slow)	DPS 6/85
6X(normal)	DPS 6/9X
6XE(slow)	DPS 6/85E
6XE(normal)	DPS 6/9XE

1.2 SCOPE

1.2.1 General Requirements

This document specifies the DPS6 Stage 3 Central Sybsystem Family, consisting of the CR41E, M5XE and M6XE models. As evolutionary members of the Level 6 Product Family, they offer increased functionality while maintaining upward compatibility with the Level 6 models 3X, 4X, 5X and 6X CSSs.

In the interest of clarity, the functionality defined in this specification is for the M6XE CSS, the top of the line model. Whenever functionality does not apply to the other models, it is so stated. The term 'CSS' is used and is meant to refer to all models unless stated otherwise.

Although this document is specifically intended to describe the Stage 3 products, it is also intended for use as an up-to-date reference for the older DPS6 CSS products.

Configurations presented in Section 13 are not all inclusive but are the recommended set.

1.2.2 Key Features

Some of the Key features of this system are:

- o Physical memory extensions on the lower 16-bit members
- o Commercial instruction capability as standard
- o Scientific instruction capability (optional on CR41E and M5XE)
- o Memory Management Unit (MMU) - CR41E and M6X support the SMMU while the M5XE and M6XE support SMMU and EMMU
- o Extended MRX Megabus (M6X and M6XE only)
- o Full compatibility with selected Megabus elements.

1.2.3 Restrictions and Enhancements

- o The CSSs support Long Address Form Addressing only.
- o MMU enhancements have been made. (Refer to Section 4.)
- o Unavailable resource trap conditions detected during the execution of commercial and scientific instructions result in a trap TV23, as in a 6X environment, rather than a TV23 or TV15 in a 4X-5X environment. See Table 3-3.

1.3 REFERENCE DOCUMENTS

1.3.1 Governing Documents

1. 60149834 DPS6 Stage 3, PFS
2. 60139142, L6 System Control Facility, EPS-1
3. 60130080, L6 Model 43, EPS-1
4. 60126298, Extended Megabus, EPS-1
5. 60129876, L6 Maintainability, EPS-1
6. 60135091, 6X Processor, EPS-1
7. 60135295, 6X Commercial Instruction Processor, EPS-1
8. 60135281, 6X Scientific Instruction Processor, EPS-1
9. 60130079, L6 Memory Management Unit, EPS-1
10. 60138362, L6 Power Specification
11. 60149832, MRX Megabus EPS-1.

1.3.2 Standards

The subsystem referenced herein shall be designed to meet or exceed the standards listed below. Any deviations to standards for the device and device-oriented electronics is detailed in the appropriate section of this Product Specification.

Recent regulations, EMI rules pertaining to Class A computing devices, enacted by the Federal Communications Commission require that the subsystem described herein adhere to FCC rules part 2.805, 15.4 pp. N through Q, 15.804, 15.810 through 15.818 and 15.838 even sections only. It is expected that the European community, Canada, and so forth, will be enacting similar rulings in the near future.

1. General Design, Honeywell Standard:

   B03.07, Reliability - Standard Failure Rate Data Base

   B03.08, Reliability Failure Rate & MTBF Predictions

   B04.08, Selection and Qualification of Standard Finishes

   B01.08, Environment, Operating

   B01.09, Equipment Safety

   B01.10, Environment, Transportation, Storage & Installation

2. Electrica Design, Honeywell Standard:

   B01.48, Primary Power - Utility Supplied

   B04.06, System Grounding

3. Mechanical Design, Honeywell Standard:

   A00.10, Metrication

   B04.09, Application of HIS Standard Finishes

4. Industrial Design, Honeywell Standard:

   D01.00, Product Use and Appearance

   D01.01, Human Factors/Industrial Design

   D01.02, Enclosure and Structure Design

   D01.03, Materials and Finishes

   D01.04, Signals and Controls

   D01.05, Cabinet Hardware

   D01.06, Miscellaneous Hardware, Accessories and Other Consumables

5. Product Maintainability, Honeywell Standard:

   B07.11, Logic Nomenclature

   B07.12, Location Reference Designation

   B07.13, Identification Nomenclature for ICs, Printed Cards and Card Cages

   B07.38, Logic Symbology

   B07.39, Logic Block Diagrams

   G02.01, FE Tools and Test Equipment Catalog

   G02.05, FE Product Tools & Test Equipment

   G07.01, Field Product Maintenance Documentation

   G07.02, Product Manual Content Guide

   G07.03, Product Style Guide for Manuals

   G07.08, Major and Intermediate Block Diagrams

   G07.09, Repair Documentation, Draft

6. D.002.01, PWA/PWB Testability Design Rules

7. Manufacturing Testability Guidelines:

   MTG1, PWA Test equipment Connection Requirements

   MTG3, PWA Microdiagnostic Creation

8. 60129949 - Application Rules for Minicomputers & Terminals Products

<u>1.3.3 Reference Only</u>

1.	Q4.1	- PWA/PWB Testability Design Rules
2.	MG1	- Component Availability
3.	MTG2	- PWA Test Documentation Requirements
4.	MTG4	- PWA Test Monitor/Test Box Designs
5.	MTG5	- PWA Quality Logic Test Creation
6.	MTG6	- PWA Test & Verification Program Creation
7.	MTG7	- PWA IC Socket Utilization
8.	MTG8	- Design for Producibility, Installability, Maintainability and Replaceability

9. MPTG1 - PWB/PWA Producibility Guidelines

10. 58035052 - Wordwide Maintenance Requirements.

1.4 DEFINITIONS

A	- Address associated with a Trap
AAS	- I/O starting address AS
AL	- Active Level
ALV	- Active Level Interrupt Vector
AS	- Address Syllable
ASV	- Address Space Vector
Atom	- Single item of information; e.g., bit, digit, byte, word, etc.
B	- Bit test indicator of I register
BCD	- Binary Coded Decimal
BD	- A 32-bit signed displacement that follows the address syllable
Bn	- Base register n, $1 \leq n \leq 7$
C	- Carry indicator of I register
CAS	- Non-procedural I/O Control Word AS
CI	- Commercial instruction indicator register
CIP	- Commercial Instruction Processor
CL	- Current Level
CLV	- Current Level Interrupt Vector
CPU	- Central Processor Unit
CSS	- Central Subsystem
CW	- Current Stack Length in words
CZ	- Clean Zero
d	- Small displacement
D	- A 16-bit signed displacement that follows the address syllable
DAS	- I/O Data Address Syllable
DD	- Data Descriptor
DZ	- Divide by Zero
e	- Exponent
EA	- Effective Address
EDAC	- Error Detection and Correction
EII	- Extended Integer Instruction
EPS-1	- Engineering Product Specification, Part 1
EOF	- Exponent Overflow
EUF	- Exponent Underflow
EUM	- Exponent Underflow Trap Mask
f	- Fraction
FT	- Address of the top element of the current active frame in the stack
FZ	- Fuzzy Zero

G	- Greater than indicator of I register
i	- Integer
I	- Indicator register; or I/O indicator of I register
I'	- Part of TSA containing I register and Trap number
IA	- Intermediate Address
IL	- Interrupting Level
ILV	- IL's interrupt vector
IMA	- Immediate Address
IMO	- Immediate Operand
I/O	- Input/Ouput
ISA	- Interrupt Save Area
ISM	- Interrupt Save Mask
IV	- Interrupt Vector
Kn	- K register n, $1 \leq n \leq 7$
L	- Less than indicator of I register
LAF	- Long Address Form
LAS	- Logical Address Space
LSB	- Least Significant Bit(s)
MAS	- Memory Address Syllable
MB	- Megabyte
MBN	- Must Be NULL Address
MBZ	- Must Be Zero
MMPO	- Main Memory PROM Option
MMU	- Memory Management Unit
Mn	- Mode Register n, $1 \leq n \leq 7$
MSB	- Most Significant Bit(s)
MW	- Maximum Stack Area Length in words
M3X	- Model 37 CSS (and functionality)
M4X,5X	- Models 43/7 and 53/7 Central subsystems (and functionality)
M6X	- Model 6X CSS (and with enhanced functionality)
NATSAP 0-3	- Next Available Trap Save Area Pointers
NULL	- An Address of all Zeros
O	- Offset
OS	- Operating System
ORU	- Optimum Replaceable Unit
OV	- Overflow indicator of I register
P	- Program Counter
PAS	- Physical Address Space
PE	- Precision Error
PEM	- Precision Error Trap Mask
PFS	- Product Functional Specification
QLT	- Quality Logic Test

RAS	- Register Address Syllable
RDBR	- Remote Descriptor Base Register
RFU	- Reserved for Future Use (Zero expected but not checked)
RHU	- Reserved for Hardware Use
RMW	- Read Modify Write
Rn	- Register n, $1 \leq n \leq 7$
RSU	- Reserved for Software Use
RTC	- Real Time Clock
s	- Sign of Mantissa (operand)
S	- System Status Register
SA	- Scientific Accumulator
SAF	- Short Address Form
SB	- Significant Bit(s)
SBZ	- Should Be Zero. A SBZ field should be set by the software to Zero. The hardware does not check this field; if it is not Zero, unspecified results occur.
SD	- Segment Descriptor
SE	- Significance Error
SEM	- Significance Error Trap Mask
SI(EUF)	- Scientific Indicator Exponent Underflow
SI(G)	- Scientific Indicator Greater Than
SI(L)	- Scientific Indicator Less Than
SIP	- Scientific Instruction Processor
SI(PE)	- Scientific Indicator Precision Error
SI(QE)	- Scientific Indicator Quality Logic Test
SI(SE)	- Scientific Indicator Significance Error
T	- Stack Address Register
TBD	- To Be Defined
THP	- Trap Handling Procedure
TSA	- Trap Save Area
TSAL	- Trap Save Area Link
TSAP	- Trap Save Area Pointer
TV	- Trap Vector
T&V	- Test & Verification
U	- Signs unlike indicator of I register
WDT	- Watch Dog Timer
Z	- Miscellaneous Trap Information in TSA.

1.4.1 Flowchart Symbols

```
      *********          ++++++++++++              o o
     *         *       +    START/    +          o     o
    * START/END *     +     RETURN     +       ( CONTINUE )
     *         *       + SUBROUTINE +            o     o
      *********          ++++++++++++              o o


    _________________                         ______________
   |                 |                       /              \
   |   ACTION BOX    |                      < DECISION BOX  >
   |_________________|                       \______________/


    _________________                      ____________________
   (                 )                    <                    >
   (   COMMENT BOX   )                    <  GO TO SUBROUTINE  >
   (_________________)                    <____________________>
```

1.4.2 Instruction Convention Descriptions

Table 1-2 lists various conventions and descriptions used in describing instructions in this document.

Table 1-2 Conventions and Definitions Used in Instructions (Sheet 1 of 2)

SYMBOL	DEFINITIONS
B#	Base Address register selected by # field in instruction
R#	Word operand register selected by # field in instruction
K#	Double word operand register selected by # field in instruction
[ ]	Contents of; e.g., [R6] = contents of R6
EA	Effective Address
( ) or .	Bit field specification; e.g., R3(6) = bit 6 of R3; I(C) = bit C of I-register; S.RN = RN field of status register
:	Field operator; e.g., R3(1:7) = bits 1 through 7 of R3
>	Greater than
<	Less than
=	Equal to
≠	Not equal to
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
**	Raised to the power of; e.g., 2**n = 2 raised to the power n

Table 1-2 Conventions and Definitions Used in Instructions (Sheet 2 of 2)

SYMBOL	DEFINITIONS
⊕	Exclusive OR
\/	Inclusive OR
/\	Logical AND
—	One's complement; e.g., $[\overline{R6}]$ = complemented contents of register R6
<--	Is replaced by; e.g., [R1] <-- [R2] = Transfer contents of R2 to R1
<-->	Swap
TEMP	Temporary register
;	Separator for non-simultaneous operations; e.g., [R2] <-- [R1]; [R3] <-- [R2]
≡	Equivalent
P	Program counter. For the purpose of P Relative Addressing, the following definitions are used: o Pa: Points to the word containing the address syllable; o Pd: Points to the word containing the displacement

SECTION 2 ARCHITECTURE

2.1 OVERVIEW OF THE CSS

The CSS offers a contemporary and open ended architecture. The CSS supports variable sizes (model dependent) of Physical Address Space (PAS) as well as Logical Address Space (LAS). Word size is 16 bits (2 bytes). The CSS may consist of a monoprocessor (CR41E); or monoprocessor or tightly coupled dual processors (M5XE and M6XE).

The key processor architecture features are:

- o 38 (31 for CR41E and M5XE) program visible registers including multiple accumulators, address, index, and control registers
- o Bit, digit, byte, word, and multiword instructions
- o Bit test, set, and mask capability
- o Immediate, register to register and register to memory operations
- o 64 standard interrupt levels
- o Multiple vectored trap structure
- o Hardware executed scientific and commercial instructions
- o Hardware supported context save and restore

o Multiple addressing modes including indexing, indirect, base plus displacement, Interrupt Vector plus displacement, program counter relative, auto increment/ decrement, stack relative, base plus displacement plus offset, etc.

o Permanent bootstrap

o Power fail detection

o Real time clock and watch dog timer

o Automatic restart (optional)

o Stack and Queue management

o Extended Memory Management supporting 32 MB of LAS (available on the M5XE and M6XE only).

Note that in the CR41E, commercial instructions are executed synchronously rather than asynchronously as in an M6X, M6XE, M5X, and M5XE.

The instructions supported by the CR41E CSS are classified as follows:

o General
o Commercial (all M5X CIP instructions)
o Scientific (all M5X SIP instructions)

The instructions supported by the M5XE CSS are classified as follows:

o General
o Commercial (all M5X CIP instructions)
o Scientific (all M5X SIP instructions)
o Extended Memory Management.

The instructions supported by the M6XE CSS are classified as follows:

o General
o EII (all M6X EII instructions)
o Commercial (all M6X CIP instructions)
o Scientific (all M6X SIP instructions)
o Extended Memory Management.

NOTE

In the CR41E and M5XE, certain instructions are not supported. Refer to Sections 5 through 8 for more information about specific exceptions.

2.2 SYSTEM BUS

The CR41E and M5XE communicate with other system components via the standard Megabus; the M6XE does so via the Level 6 Extended megabus. Block diagrams of typical configurations are shown in Figures 2-1, 2-2 and 2-3.

STANDARD 16-BIT MEGABUS

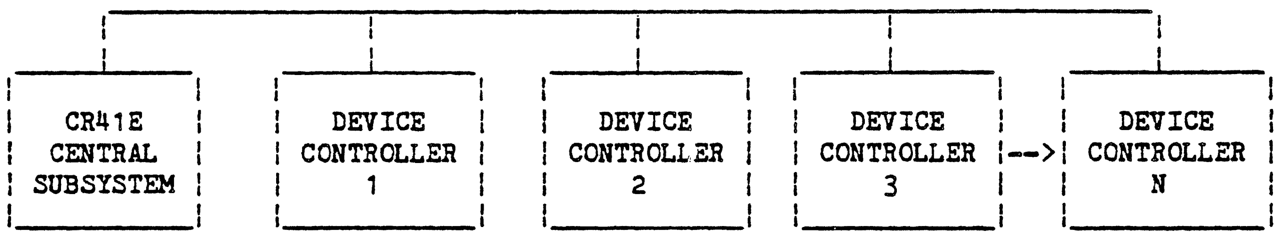


Figure 2-1 Standard 16-Bit Megabus Interconnection Block Diagram For CR41E

STANDARD 16-BIT MEGABUS

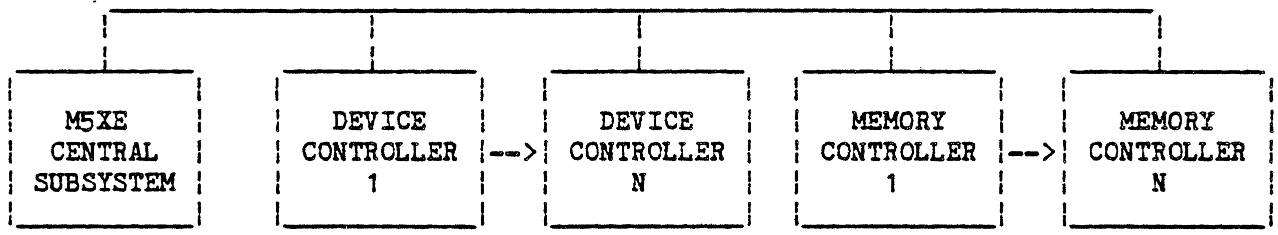


Figure 2-2 Standard 16-Bit Megabus Interconnection Block Diagram For M5XE

LEVEL 6 EXTENDED MEGABUS (M6XE)

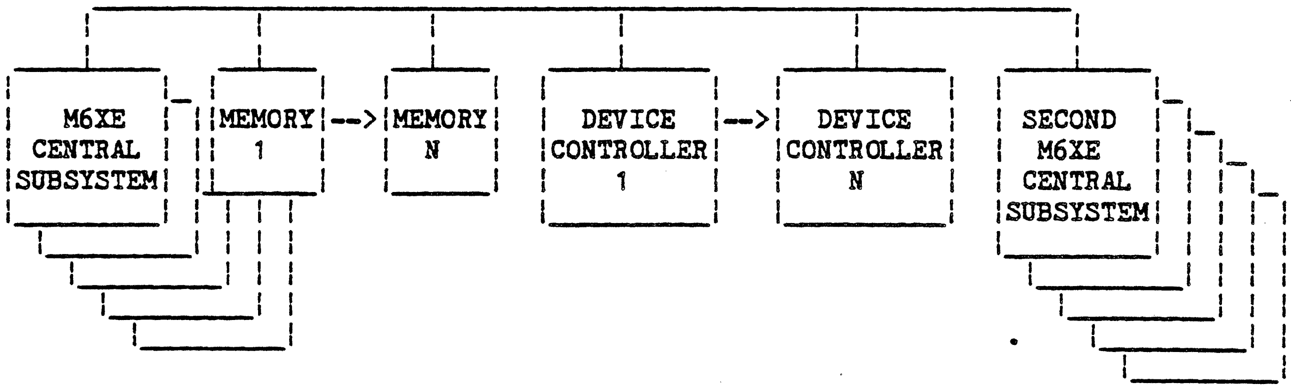


Figure 2-3 Extended 32-Bit Megabus Interconnection Block Diagram For M6XE

2.3 SYSTEM CONFIGURATIONS

For system configurations, refer to Section 14.

2.4 CSS ATTRIBUTES

2.4.1 CR41E

2.4.1 1 PROCESSOR

o Control Store Size	- 48X2K (processor), 56X4K (processor + CIP)
o Cycle Time	- 240 nsec.
o MIPS	- 0.4

2.4.1 2 MEMORY

o Size	- 4 MB (on CPU board)
o Cycle Time	- Read 350 / 1000 nsec. and Write 300 / 750 nsec. with times representing approximate memory controller and processor cycle times respectively.

2.4.1 3 CACHE

None

2.4.2 M5XE

2.4.2 1 PROCESSOR

o Control Store Size	- 72X2K
o Cycle Time	- 160 nsec.
o MIPS	- 0.7

2.4.2 2 MEMORY

o Size	- 8 MB (on separate board)
o Cycle Time	- Read 350 / 1000 nsec. and Write 300 / 750 nsec. with times representing approximate memory controller and processor cycle times respectively.

2.4.2 3 CACHE

o Size	- 4K X 16 bits.
o Organization	- Set associative 4 Levels.
o Addressability	- CPU only reads and writes through cache to memory.
o Cycle Time	- Read 160 nsec.
	- Write is write through to memory.

2.4.3 M6XE

2.4.3 1 PROCESSOR

o Control Store Size	- 96X2K
o Cycle Time	- 98 nsec.
o MIPS	- 1.8

2.4.3 2 MEMORY

o Size	- 16 MB (on two board)
o Cycle Time	- Read 350 / 1000 nsec. and Write 300 / 750 nsec. with times representing approximate memory controller and processor cycle times respectively.

2.4.3 3 CACHE

o Size	- 4K X 16 bits.
o Organization	- Set associative 2 Levels.
o Addressability	- CPU, CIP and SIP read and write through cache to memory.
o Cycle Time	- Read 200 nsec.
	- Write is write through to memory.

This page is intentionally blank.

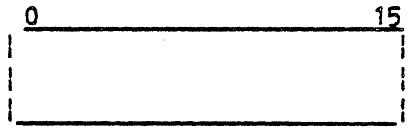
SECTION 3 FUNCTIONAL REQUIREMENTS

3.1 DATA FORMATS

3.1.1 Data Formats for Non-Commercial or Scientific Instructions

3.1.1.1 MEMORY WORDS

All data elements are based on 16-bit memory words. The format of each memory word is defined from left to right with the first bit numbered Zero.

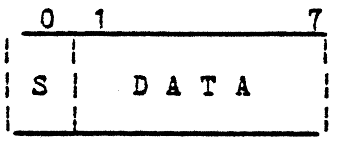


Memory may be accessed by instructions to the bit, byte, word or multiword data item level. In all cases, the leftmost element is the most significant element, such that bit 0 above is the first bit, bit 1 is the second bit, bits 0 through 7 are the first byte, bits 8 through 15 are the second byte, etc. Multiword items require successive word locations with the lowest address defined as the most significant or leftmost part of the data item.

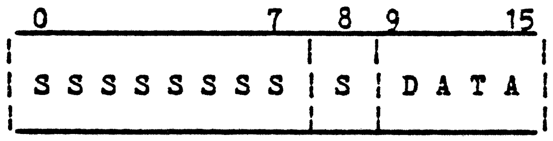
3.1.1.2 SIGNED INTEGER DATA

Signed integer data is in two's complement form. S represents the sign bit, where for S = 0, integer is positive or Zero; for S = 1, integer is negative. The following types are available:

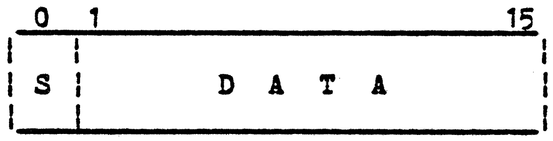
- Signed Integer Data Byte - Data is an 8-bit integer with the radix point to the right of bit 7, the least significant bit. S indicates the sign. Range (r) is $-2^7 \leq r \leq 2^7-1$.



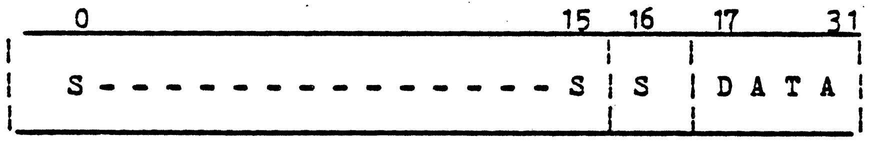
- Sign Extended Integer Byte in a Word - Data is a 16-bit integer with the radix point to the right of bit 15, the least significant bit. S indicates the sign. Range (r) is $-2^7 \leq r \leq 2^7-1$.



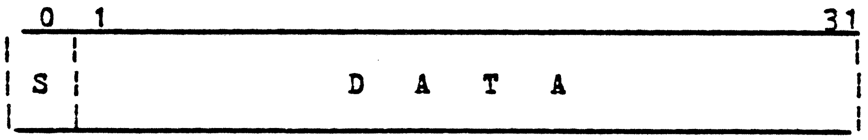
- Signed Integer Data Word - Data is a 16-bit integer with the radix point to the right of bit 15, the least significant bit. S indicates the sign. Range (r) is: $-2^{15} \leq r \leq 2^{15}-1$.



- Sign Extended Integer Word in a Double Word - Data is a 32-bit integer with the radix point to the right of bit 31, the least significant bit. S indicates the sign. Range (r) is: $-2^{15} \leq r \leq 2^{15}-1$.



- Signed Integer Data Double Word - Data is a 32-bit integer with the radix point to the right of bit 31, the least significant bit. S indicates the sign. Range (r) is: $-2^{31} \leq r \leq 2^{31}-1$.



3.1.1.3 UNSIGNED INTEGER DATA

The following unsigned integer data types are available:

o Unsigned Integer Byte - Data is an 8-bit integer with Range (r): $0 \leq r \leq 2^8-1$.

```
0        7
|         |
| D A T A |
|_________|
```

o Unsigned Integer Byte in a Word - Data is a 16-bit integer with Range (r): $0 \leq r \leq 2^8-1$.

```
0          7 8          15
|           |             |
| ALL ZEROs | D A T A     |
|___________|_____________|
```

o Unsigned Integer Word - Data is a 16-bit integer with Range (r): $0 \leq r \leq 2^{16}-1$.

```
0                       15
|                         |
|        D  A  T  A       |
|_________________________|
```

o Unsigned Integer Word in a Double Word - Data is a 32-bit integer with Range (r): $0 \leq r \leq 2^{16}-1$.

```
0                     15 16                     31
|                        |                        |
|    A L L   Z E R O s   |     D   A   T   A      |
|________________________|________________________|
```

o Unsigned Integer Double Word - Data is a 32-bit integer with Range (r): $0 \leq r \leq 2^{32}-1$.

```
0                                                31
|                                                 |
|                  D   A   T   A                  |
|_________________________________________________|
```

3.1.2 Data Formats for Commercial Instructions

3.1.2.1 OVERVIEW

Commercial instructions operate on three data types:

- o Decimal strings - binary-coded-decimal representation
- o Alphanumeric strings - ASCII-8 code characters
- o Binary numbers - 16 or 32 bit precision.

Each instruction is designed to assume a specific data type for each operand. Operations on each data type, therefore, are limited to specific mutually exclusive subsets of the commercial instruction repertoire. These are listed below:

- o Decimal data operations:
  - Arithmetic: Add, subtract, multiply, divide
  - Decimal comparison
  - Conversion between decimal data formats
  - Conversion to binary
  - Decimal shift
  - Numeric string edit
- o Alphanumeric operations:
  - Alphanumeric comparison
  - Translation by character
  - String search - identify equality
  - String verify - detect inequality
  - String move
  - Alphanumeric string edit
- o Binary:
  - Conversion to decimal string

The instructions are described in Section VII of this specification. Each of the data types, including its representation and format in memory is described in this subsection.

The operands for each instruction are specified by a set of descriptors which provide information regarding the location, size and format for each operand. These descriptors may be coded in line with the processing instructions or located in tables. The data type (i.e., decimal, alphanumeric or binary) is implicit according to the instruction so that mixed data-type operations are not performed.

3.1.2.2 DECIMAL DATA

Decimal data operands have the following general characteristics:

- o A decimal datum is assumed to be a real integer.
- o Digits of a decimal number occupy contiguous storage locations.

o They are referenced by the location of the most significant digit or leading sign if there is one.

o The most significant digit is assigned to the location with the lowest memory address. The most significant digit within a word is to the left of less significant digits in the same word.

The various representations of decimal data differ primarily with regard to:

o Whether there are one or two digits in a byte; namely string or packed decimal.

o Whether the value is signed or unsigned (assumed positive).

o The position of the sign, if any.

3.1.2.2.1 String Decimal

Each digit of a string decimal datum occupies one byte in storage. The datum can occupy an even or odd number of bytes and can start at an even or odd byte address.

Depending on the coding of the descriptor for the datum, any of four sign conventions are assumed:

o Unsigned - assumed positive

o Leading separate sign byte - before the most significant numeric digit

o Trailing separate sign byte - after the least significant numeric digit

o Trailing over-punched sign - in the same byte as the least significant digit.

All digit bytes, except the one containing an over-punched sign, must have valid pure binary-coded decimal (BCD) values in the least significant four bits. The most significant four bits of each non-sign digit byte is ignored in performing the operation. These bits (termed zone bits) are set to hexadecimal three in digit bytes stored as a result of an operation. The resulting digits are represented, therefore, as the ASCII codes for numeric characters 0 through 9.

Up to 31 bytes can be used to store a decimal string datum including sign. The numeric precision is therefore 30 digits for numbers with the leading or trailing separate sign bytes and 31 digits for over-punched sign or unsigned data.

Table 3-1 indicates the sign code conventions recognized and generated when processing commercial instructions. For the leading and trailing separate sign convention, the positive and negative signs are denoted by the ASCII plus and minus characters, respectively. In the trailing over-punched sign convention, the least significant four bits of the least significant digit byte do not adhere to BCD coding and undergo a transliteration when over-punched by a sign to form an ASCII character. The full 8 bits are examined to determine the sign and digit value for numeric operations on over-punched sign operands (The older models decode only the least significant 7 bits to determine sign and digit value).

3.1.2.2.2 Packed Decimal

Each byte of a packed decimal datum contains two BCD digits; i.e., each digit occupies a half-byte. Two sign conventions are provided; unsigned or trailing separate sign. Unsigned numbers are assumed positive. The coding of the sign for signed number strings is indicated in Table 3-2.

Up to 31 half-bytes of storage can be used to represent a packed decimal datum. The numeric precision is therefore 31 digits for an unsigned datum and 30 digits for a signed datum.

The first and last digits representing a number can be situated in any half-byte aligned position of a word. The length of the operand for this data type is expressed in half-bytes. Any half-byte position in memory can be specified as the location of the datum.

Except for the sign, all digits must be in the BCD code set for numeric 0 through 9; otherwise an illegal character trap TV27 occurs.
For a detailed description, refer to subsection 7.3.

3.1.2.3 ALPHANUMERIC DATA

Alphanumeric operands consist of 8-bit characters. Their maximum size is 255 characters except as specified otherwise. Alphanumeric strings in memory can start and/or end on either odd or even byte boundaries.

3.1.2.4 BINARY DATA

Binary operands can be either 16 bits long or 32 bits long. They are 2's-complement integers and thus the most significant bit is the sign bit and the binary point is assumed to be to the right of the least significant bit. The range of the value of the binary operand is:

- o For a 16-bit long operand: $-2^{15} \leq r \leq 2^{15}-1$.
- o For a 32 bit long operand: $-2^{31} \leq r \leq 2^{31}-1$.

Note that the length of binary data is specified in bytes; thus, the length of the operand should be specified as either two or four bytes, otherwise unspecified results will occur. Binary operands in memory can start on either odd or even byte boundaries.

3.1.3 Data Formats for Scientific Instructions

Scientific instructions operate on two data types:

- o Hexadecimal Floating Point
- o Signed Binary Integer

Table 3-1 Sign Conventions for String Decimal Operands

SEPARATE LEADING AND TRAILING SIGN		
SIGN VALUE	ASCII CHARACTER	HEXADECIMAL CODE
+	+	2B
-	-	2D

TRAILING OVERPUNCH SIGN				
SIGN VALUE	DIGIT VALUE	ASCII CHARACTER	HEXADECIMAL CODE RECOGNIZED AND GENERATED	HEXADECIMAL CODE RECOGNIZED ONLY
+	0		7B	30
+	1	A	41	31
+	2	B	42	32
+	3	C	43	33
+	4	D	44	34
+	5	E	45	35
+	6	F	46	36
+	7	G	47	37
+	8	H	48	38
+	9	I	49	39
-	0		7D	None
-	1	J	4A	None
-	2	K	4B	None
-	3	L	4C	None
-	4	M	4D	None
-	5	N	4E	None
-	6	O	4F	None
-	7	P	50	None
-	8	Q	51	None
-	9	R	52	None

Table 3-2 Packed Decimal Sign Conventions

SIGN DIGIT (HEXA-DECIMAL)	PACKED DECIMAL ASCII SIGN	
	GENERATED BY HARDWARE	RECOGNIZED BY HARDWARE
A		+
B	+	+
C		+
D	-	-
E&F		+

3.1.3.1 HEXADECIMAL FLOATING POINT

A hexadecimal floating point number can be 32 bits (a double word) or 64 bits (a quadruple word) in length. The format of the number is:

```
  0     6  7  8               31/63
  ---------------------------------
  :   e   : s :        f          :
  ---------------------------------
```

where: e - The exponent in excess 64 form. The range (r) of the true value of e is $-64 \leq r \leq +63$. The radix is 16;

s - The operand sign where s = 0 is positive and s = 1 is negative; and

f - The fractional mantissa in magnitude form. The range of f is:

$0 \leq f \leq (16^6-1)/16^6$ for double word

$0 \leq f \leq (16^{14}-1)/16^{14}$ for quadruple word.

Thus, the value (v) of the floating point number is defined to be:

$$v = (-1)^s \times f \times 16^{(e-64)}.$$

For normalized floating point numbers,

$$f = 0 \text{ or } f \geq 1/16$$

A floating point number in which all bits, including fraction (mantissa), sign, and exponent are Zero is defined as a floating point Clean Zero (CZ).

A floating point number in which all bits of the fraction (mantissa) are Zero and bits of the sign and/or exponent are not Zero is defined as a floating point Fuzzy Zero (FZ).

3.1.3.2 BINARY INTEGER FORMAT

A signed integer number can be 16 bits (a single word) or 32 bits (a double word) in length. The format of the number is:

```
  0                                  15/31
  ----------------------------------------
  |                 i                    |
  ----------------------------------------
```

where i - The integer in two's complement form. The radix point is to the right of bit 15/31, the least significant bit. The range of the signed integer is:

$-2^{15} \leq i \leq 2^{15}-1$ for a single word and

$-2^{31} \leq i \leq 2^{31}-1$ for a double word.

3.1.3.3 "PSEUDO DECIMAL" FLOATING POINT FORMAT (M6X AND M6XE ONLY)

A "pseudo decimal" floating point number can be three or five words in length. The length is specified by the accumulator length field of M4. The format of the number in memory is:

```
  -0-----------15-
 n|   |         |
  | S |         |   Word N
  |   |         |     .
  |-----        |     .
  |             |     .
  -      M      -     .
  -             -     .
                      .
  |             |     .
  |-------------|     .
  |             |     .
  |      EX     |     .
  |             |   Word N + 2 or 4
  ---------------
```

EX - The exponent in two's complement form. The useful range of EX is $-77 \leq EX \leq 77$. The radix is 10.

S - The operand sign where S = 0 is positive and S = 1 is negative. S is bit 0 of word N.

M and S form the mantissa in two's complement form. The range of the mantissa is:

$$-2^Y \leq \text{mantissa} \leq 2^Y-1$$

where y = 31 for double word mantissa, 63 for quad word mantissa.

The value (v) of the number is defined to be:

$$v = \text{mantissa} \times 10^{EX}$$

3.2 ADDRESSES

Addresses are used to point to an operand, specifically, to a data item (for example, bit, byte, word, and so forth) or to an instruction. Addresses are stored in memory or address registers. Word addresses are the native address form of the processor.

3.2.1 Address Types

The processor supports both physical addresses (PAs) and Logical Addresses (LAs). Most software visible addresses (for example, in base registers, memory, and so forth) are LAs. PAs are typically computed by the processor by translating LAs as a function of descriptors stored in the Memory Management Unit (MMU).

Segment descriptors use PAs.

3.2.1.1 PHYSICAL ADDRESS

A Physical Address (PA) is used by the processor to address the main memory. Its format is:

```
 0                                                            *
 ______________________________________________________________
|                                                              |
|              P H Y S I C A L    A D D R E S S                |
|______________________________________________________________|
```

Where: * = 20 for a CR41E (4 MB addressability)

* = 21 for an M5XE (8 MB addressability)

* = 22 for an M6XE (16 MB addressability)

Whenever the processor addresses a subword data item or passes a buffer address to a device controller, the processor extends its word PA to a byte PA by appending a byte bit to it. Note that the PA supported by the Megabus is in bytes.

3.2.1.2 LOGICAL ADDRESS

A Logical Address (LA) is the address used by a process to address its logical memory. An LA is 20-bits or 24-bits as a function of the Memory Management Unit (MMU) mode used, when contained in a base register. An LA occupies 32 bits when stored as an address value in memory or when in IMA form. Whenever an address value or IMA is loaded into a base register or used as an address, then use only its low order n bits (where n = 20 for a CR41E and n = 24 for the M5XE, M6X and M6XE) and check the remaining high order bits for zero. If the high order bits are non zero, then post a Trap TV15.

3.3 VISIBLE REGISTERS

Thirtyeight (31 for a CR41E) program-visible registers can be loaded and read by various instructions in the Level 6 instruction set. There are:

- o Seven general word operand registers
- o Ten address registers
- o Seven general double-word operand registers (not available on a CR41E)
- o Twelve control registers
- o Three scientific accumulators, and
- o One descriptor segment base register.

The register names and functions are defined below, and are shown in Figure 3-1.

3.3.1 Word Operand Registers

Registers R1 through R7 are 16-bit word operand general registers and accumulators. They can also be used for post-indexing of addresses.

```
                          0            15
                          ----------------
          R1 through R7   |              |
                          ----------------
```

3.3.2 Address Registers

Address registers are 20 bits (or 24 bits if EMMU) in length. Registers B1 through B7 are base registers, P is the program counter, RDBR is the Remote Descriptor Base Register used whenever an AS specifies a remote descriptor, and T is the Stack Pointer Register.

```
                        0             19/23
       B1 through B7,   ---------------------
       T, P, and RDBR   |                   |
                        ---------------------
```

The seven base registers (B1 through B7) can be used for formulating addresses pointing to any word of procedure, data, or arbitrary location in the virtual memory space. Address registers will typically contain addresses, pointers or base references for use in generating effective addresses and referencing program and data relatively. Base registers have auto increment and auto decrement capability to allow easy use of these registers for stack, queues, and program loops.

The contents of P or the program counter is the address of the current instruction. Normally, P is incremented to point to the next instruction, except as noted for branches and jumps.

```
    WORD OPERAND REGISTERS      GENERAL REGISTERS     INDEX REGISTERS
     0                  15      AND ACCUMULATORS
    :         R1         :            <---                 <---
    :         R2         :            <---                 <---
    :         R3         :            <---                 <---
    :         R4         :            <---                 <---
    :         R5         :            <---                 <---
    :         R6         :            <---                 <---
    :         R7         :            <---                 <---

    ADDRESS REGISTERS           PROGRAM         BASE              STACK
     0               19/23      COUNTER         REGISTERS         POINTER
    :        P         :          <---
    :        B1        :                          <---
    :        B2        :                          <---
    :        B3        :                          <---
    :        B4        :                          <---
    :        B5        :                          <---
    :        B6        :                          <---
             B7        :                          <---
    :       RDBR       :                          <---
    :        T         :                                            <---

    DOUBLE-WORD REGISTERS*      GENERAL REGISTERS     INDEX REGISTERS
     0                31        AND ACCUMULATORS
    :       K1         :.         <---                  <---
    :       K2         :          <---                  <---
    :       K3         :          <---                  <---
    :       K4         :          <---
    :       K5         :          <---
    :       K6         :          <---
    :       K7         :          <---

    CONTROL REGISTERS           SYSTEM AND      INDICATORS        TRAP ENABLE/
      0     7        15         SECURITY KEYS                     MODE CONTROL
  S :                 :           <---
  I :          :                                  <---
 CI :          :                                  <---
 SI :          :                                  <---
 M1 :          :                                                    <---
 M2 :          :                                                    <---
 M3 :          :                                                    <---
 M4 :          :                                                    <---
 M5 :          :  _                                                 <---
 M6 :          :  :_  used by CR41E-based designs as hardware
 M7 :          : _:   support for integrated CIP

    SCIENTIFIC ACCUMULATORS                     GENERAL SCIENTIFIC REGISTERS
     0             31           63              AND ACCUMULATORS
SA1 :               :            :                 <---
SA2 :               :            :                 <---
SA3 :               :            :                 <---
```

Figure 3-1 Visible Registers

*Available only in the M6X and M6XE models.

3.3.3 Double Word Operand Registers (M6X and M6XE only)

Registers K1 through K7 are used with Extended Integer Instructions (EII) and are 32 bits in length. K1, K2, and K3 can also be used for indexing of addresses.

```
                    0                                31
                    ----------------------------------
      K1 through K7 |                                |
                    ----------------------------------
```

3.3.4 Control Registers

3.3.4.1 S REGISTER

The S register contains the process status security keys. The contents of the S register can be read via the Store S Register instruction and are also saved at context switch time. At context load time, the processor only uses the RN field, all other fields/bits are ignored. The bit fields are defined as follows:

```
         0  1    2  3   4           7  8    9 10            15
        -------------------------------------------------------
    S = | C |  RN  | SH |      RFU     | CH# |        CL      |
        -------------------------------------------------------
```

where C = Check indicator:

- o For C = 1 then one or more ORUs (for example, processor, Device Controller) have not passed their QLTs
- o For C = 0 then all QLTs have been completed successfully.

RN = Ring Number. The number of rings supported by the processor is four, numbered from 0 to 3. Ring 0 is the most privileged and ring 3 the least. The current ring of execution of the processor (RCR) is stored in the S register (S.RN). The ring number codes are as follows:

	Level 6 Code
Ring 0	11
Ring 1	10
Ring 2	01
Ring 3	00

SH = Super Halt indicator (M6X and M6XE only):

SH=1 = The CSS is in the super halt state. A super halt is posted during Trap, RTT, LEV or interrupt processing upon detection of any of the following: a NULL TV, NULL NATSAP, UAR, RED, parity Error or protection violation. The super halt does not cause a ring change, but it disables the EXECUTE function. To restart, push the STOP function, enter a new P and depress EXECUTE or alternatively, depress MASTER CLEAR.

SH=0 = The CSS is not in the super halt state.

RFU = Reserved for Future Use.

CH# = Channel Number. This two-bit field defines the processor channel number. The channel numbers reserved for the processor are 0 - 3. The CH# provides the least significant two bits of the channel number, the remaining bits of the processor channel number are Zero. Circuitry on the processor determines the CH# as a function of its position on the Megabus.

CL = Current Interrupt Level number where $0 \leq CL \leq 63$. "CL = 0" is the highest priority level and "CL = 63" is the lowest priority level. See subsection 3.5.

3.3.4.2 INDICATOR (I) REGISTER

The I register contains the program status indicators for general and extended integer instructions.

	0	1	2	3	4	5	6	7
I =	OV	RFU	C	B	I	G	L	U

where OV = Overflow indicator

C = Carry of latest operation designated to affect this bit

B = Bit Test indicator, representing the state of last bit tested

I = Input/Output (I/O) indicator representing the status of the last peripheral interrogated. The meaning of the bit is peripheral dependent, but generally I is set to One if the device accepted the I/O command sent to it.

G, L, U = Greater than , Less than, and signs Unlike bits representing the results of the latest compare.

3.3.4.3 COMMERCIAL INDICATOR (CI) REGISTER

This eight-bit indicator register is cleared at initialization time and gets updated during the execution of commercial instructions.

Only the bits pertinent to the instruction are changed (set or cleared); all other bits remain unchanged. These bits are testable via the commercial branch instructions. Note that when testing any one or more of the bits, all the indicators, including the one(s) being tested, are left unchanged.

	0	1	2	3 4	5	6	7
CI	OV	TR	SF	RFU	G	L	RFU

- Bit 0 (OV) - Overflow Indicator - This bit is set during decimal operations when either:

  - The receiving field cannot contain all significant digits of the result, or

  - A divide-by-Zero condition is detected.

- Bit 1 (TR) - Truncation Indicator - This bit is set during alphanumeric operations when the receiving field cannot contain all characters of the result.

- Bit 2 (SF) - Sign Fault - This bit is set during decimal operations when a negative result is to be stored in an unsigned field.

- Bits 3 and 4 (RFU) - Reserved for Future Use.

- Bit 5 (G) - Greater Than - This bit is set during the executing of certain decimal and alphanumeric instructions. The following conditions are reported:

  - Result is greater than Zero for decimal arithmetic instructions.

  - The first operand is greater than the second operand for either decimal or alphanumeric comparisons.

- Bit 6 (L) - Less Than - This bit is set during the execution of certain decimal and alphanumeric instructions. The following conditions are reported:

  - Result is less than Zero for decimal arithmetic instructions.

  - The first operand is less than the second operand for either decimal or alphanumeric comparisons.

It should be noted that the Commercial Indicator Register is part of the system context and will be saved/restored as a function of the mask bits in the Interrupt Save Area.

3.3.4.4 SCIENTIFIC INDICATOR (SI) REGISTER

This 8-bit indicator register is cleared at initialization time and gets updated during the execution of scientific instructions. Only the bits pertinent to the instruction are changed (set or cleared); all other bits remain unchanged. These bits are testable via the scientific branch instructions.

0	1	2	3	4	5	6	7
EUF	RFU	SE	PE	RFU	G	L	RFU

- Bit 0 (EUF) - Exponent Underflow - Set when the result of a scientific operation has an exponent value which is smaller than the allowed range for an exponent. Refer to subsection 3.1.3.1.

- Bits 1, 4 and 7 (RFU) - Reserved for Future Use (MBZ)
- Bit 2 (SE) - Significance Error - Set if an integer is truncated during a floating point to integer conversion operation.
- Bit 3 (PE) - Precision Error - Set when a nonzero portion of a fraction is truncated during a floating point to integer conversion operation. See the SST instruction for specific interpretation of SI(PE).
- Bit 5 (G) - Greater Than - May only be changed during a compare operation.
- Bit 6 (L) - Less Than - May only be changed during a compare operation.

The scientific indicator register is part of a system context and is saved/restored as a function of the interrupt save mask in the ISA.

3.3.4.5 M1 REGISTER

The M1 register contains the trap enable mode control keys for jump and branch instructions as well as any instruction that sets the OV indicator because of an Rn register overflow. The fields are defined as follows:

```
          0 1     7
          ---------
    M1 =  | J | T#  |
          ---------
```

where J = Trace Trap Enable for jumps and branches:
0 = Trace Trap disabled
1 = Trace Trap enabled (TV02)

T# = T1 through T7, Overflow Trap Enable controls for registers R1 through R7 respectively:
0 = Trap disabled
1 = Trap enabled (TV06).

A trap can be requested during the execution of the following jump and branch instructions: JUMP, LNJ, ENTER and all branch instructions.

A trap can be requested when an overflow occurs during the execution of the following instructions: ADD, SUB, MUL, DIV, ADV, MLV, SAL, DAL, AID and SID.

3.3.4.6 M2 REGISTER

In the CR41E and M5XE models, the M2 register is present but not used (RFU). In the M6X and M6XE models, the M2 register contains trap enable, mode control keys, for any extended integer instruction that sets the OV indicator because of a Kn register overflow. The fields are defined as follows:

```
          0   1     7
          -----------
    M2 =  | RFU | T#  |
          -----------
```

where T# = T1 through T7, Overflow Trap Enable controls for registers K1 through K7 respectively:
0 = Trap disabled
1 = Trap enabled (TV06).

A trap can be requested when an overflow occurs during the execution of the following instructions: KADD, KSUB, KMUL, KDIV, KAL, KADV and KMLV.

3.3.4.7 M3 REGISTERS

The M3 Register contains trap enable mode control keys for commercial instructions. The fields are defined as follows:

```
          0    1    2       7
          -------------------
   M3 =   | OV | TR | R F U |
          -------------------
```

where OV = Overflow Trap Mask:
0 = Trap disabled
1 = Trap enabled (TV29)

TR = Truncation Trap Mask:
0 = Trap disabled
1 = Trap enabled (TV28).

3.3.4.8 M4 REGISTER

The M4 Register contains control information for scientific instructions. The fields are defined as follows:

```
           0     1     2     3     4     5     6     7
         -------------------------------------------------
  M4 =   | R/T | RFU | ML1 | AL1 | ML2 | AL2 | ML3 | AL3 |
         -------------------------------------------------
                     |___________|___________|___________|
                           |           |           |
                         SA #1       SA #2       SA #3
```

where R/T = Round/Truncate Mode:
0 = Truncate
1 = Round

SA #1 = Scientific Accumulator #1

SA #2 = Scientific Accumulator #2

SA #3 = Scientific Accumulator #3

ML = Memory Length. Length of Main Memory data field associated with this SA:
0 = Two words
1 = Four words

AL = Accumulator Length. Length of the value in the SA:
0 = Two words
1 = Four words.

3.3.4.9 M5 REGISTER

The M5 Register contains trap enable mode control keys for scientific instructions. The fields are defined as follows:

```
                  0     1     2     3   4     7
                 -------------------------------
        M5 =     | EUM | RFU | SEM | PEM | R F U |
                 -------------------------------
```

where EUM = Exponent Underflow Trap Mask:
0 = Trap disabled
1 = Trap enabled (TV19)

SEM = Significance Error Trap Mask:
0 = Trap disabled
1 = Trap enabled (TV21)

PEM = Precision Error Trap Mask:
0 = Trap disabled
1 = Trap enabled (TV22).

3.3.4.10 M6 AND M7 REGISTERS

In the CR41(E) model, the M6 and M7 registers cannot be used else post a trap TV05. In all other models, the M6 and M7 Register are reserved for future use.

3.3.5 Scientific Accumulators (SA)

Three variable length scientific accumulators are supported. These contain hexadecimal floating point values two or four words in length. M4 contains control bits for each SA (1, 2, or 3) which define both the accumulator length and the length of memory operands directed to the accumulator.

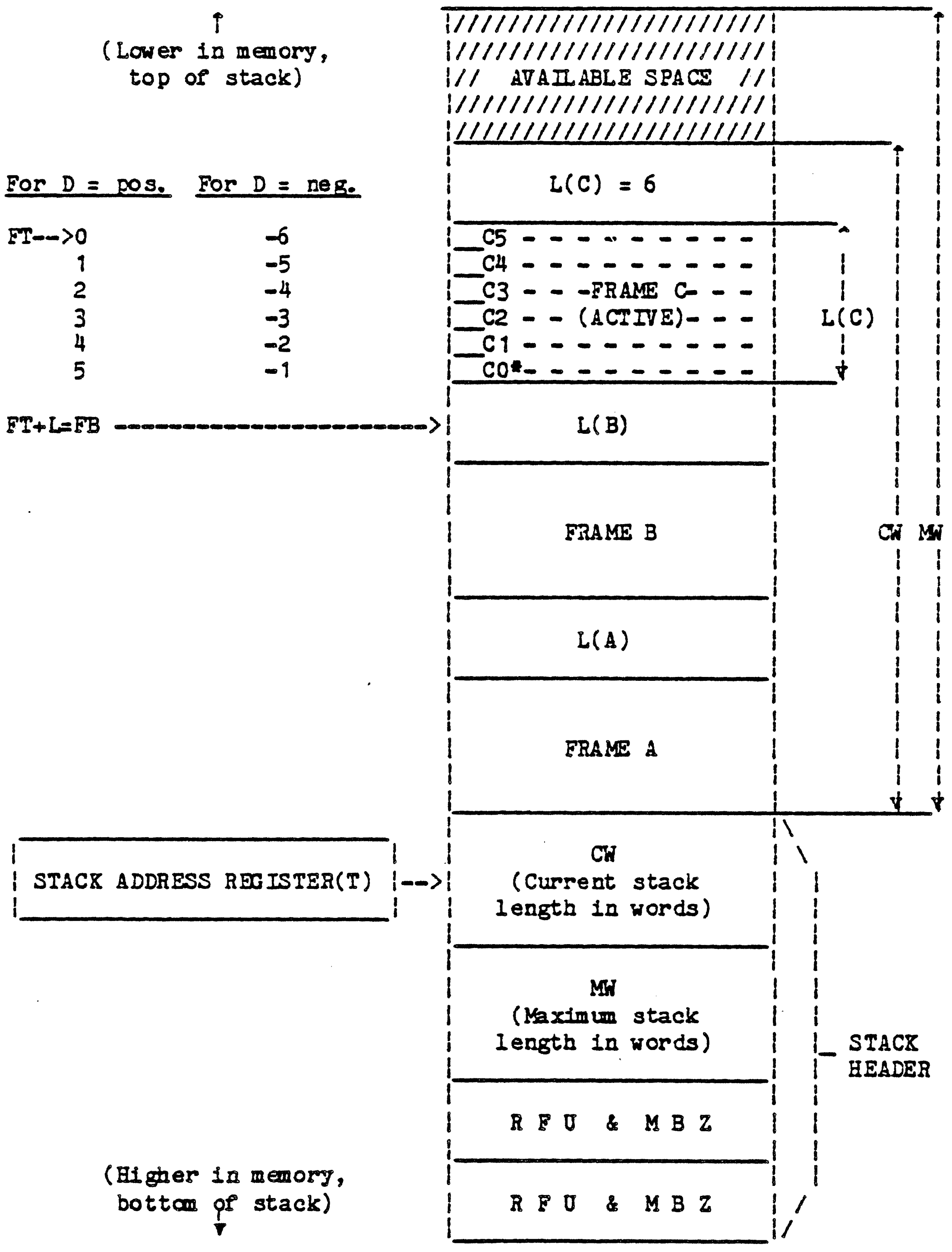
3.4 STACK MANAGEMENT

The CS functionality provides a simple stack capability for each interrupt level. The Stack Address Pointer (T), points to the first word of the stack header (See Figure 3-2).

3.4.1 Stack Header

The stack header contains four entries; two are not used by the processor and must contain null pointers.

MW is a 16-bit positive integer that defines the number of words allocated to the stack. MW is set by software when the header is created and referenced (but not altered) by hardware.



* where Cn = Word n of Frame C

Figure 3-2 Stack Structure

CW is a 16-bit positive integer that defines the number of words currently consumed by the stack. CW is set by software when the header is created; thereafter, the value of CW is updated by the hardware.

3.4.2 Stack-Related Instructions

The following generic instructions are provided to manipulate the stack:

- o Load Stack Address Register (LDT)
- o Store Stack Address Register (STT)
- o Acquire Stack Space (ACQ)
- o Relinquish Stack Space (RLQ).
- o Modify Frame Length (MFL). (M6X and M6XE only)

These instructions all contain two words and have a common first word. Appropriate checks are made for stack/overflow conditions. For a description of the above instructions, refer to Section 5.

3.4.3 Stack-Related Address Syllable (M6X and M6XE only)

A set of Stack related Address Syllables is defined to perform limit checking whenever data in the active frame is being accessed. These ASs are relative to the the top of the current active frame (FT) and assure that the data being referenced is within the active frame. If an indirect stack related AS is used (e.g., @[FT+D]+O), then the CSS assures that the pointer to the data is within the active frame. Any limit check violation results in a Trap 16 (program error).

In Figure 3-2 examples of FT+D addressing are given. Refer to subsection 3.11.3.5 for a description of these entries.

3.4.4 Stack Management Restrictions (M6X and M6XE only)

The stack functionality supported by the CSS is fully compatible with that supported by the previous members of the DPS6 family. The following restrictions however apply when using the Modify Frame length instruction and stack related address syllables.

- o The stack header in memory must not be accessed by software following the execution of a Modify Frame Length (MFL) instruction or usage of a stack-related address syllable since the CSS does not update the header in memory but rather hidden CSS registers only. Note that as long as an MFL instruction or stack-related address syllable is not used, the stack header is maintained current in main memory as in previous members of the DPS6 family.

- o The stack header in main memory may be accessed by the software after it is made current by the CSS following a reload of the T register.

3.5 INTERRUPTS

3.5.1 Concept

Interrupts are events generally unrelated to the current process. They can be generated in four ways:

- o Externally by:
  - A peripheral device requiring service;
  - A watchdog timer runout;
  - A real time clock runout;
  - CSS-CSS dialog;
- o Internally by:
  - An LEV instruction execution or
  - The exhaustion of a trap save area pool
- o By an incipient power failure.

Typically, external interrupts and power fail interrupts are honored at the end of the current instruction and internal interrupts are honored during the execution of an instruction. Regardless of the type, interrupts, when honored, typically result in a level and context change. The context of the currently executing process is stored in main memory and then the context of the interrupting process is fetched from main memory.

3.5.2 Interrupt Handling

Every process executes at a priority level defined by the CL field in the S register. CL is the Current Interrupt Level number ($0 \leq CL \leq 63$). Level 0 is defined to have the highest priority while Level 63 has the lowest priority. Level assignments for various events are shown in Table 3-3.

Any event having a priority greater than the currently executing process causes the latter to be interrupted; that is, the currently executing process is interrupted if $IL < CL$, where IL = Interrupting Interrupt Level Number.

When an interrupt is honored, a context switch is performed by using the following constructs:

- o Level Activity Flags (AFs),
- o A set of Interrupt Vectors (IVs),
- o Interrupt Save Areas (ISAs), and
- o Hardware Context Areas (HCA) (M6X and M6XE only)

The CSS requires that the above constructs be in main memory. The CSS (M6X and M6XE only) also can not tolerate a trap (i.e., TV15 or 17) when accessing any of the above constructs. If a trap is encountered, then the CSS will post a super halt. In a CR41E and M5XE a trap will cause unspecified results.

The CSS does not perform any access right checks when it accesses any of the above constructs. Specifically, it assumes to have ring 0 privilege and ignores the access permit flags.

Table 3-3 Interrupt Level Assignments

EVENT CAUSING INTERRUPT	LEVEL ASSIGNMENT	DEV WORD SETING	LEVEL ASSIGNED BY
Power Failure	0	No change	Hardware
Watchdog Timer	1	No change	Hardware
Exhaustion of Trap Save Area Pool	2	No change	Hardware
INHIBIT	n	No change	Software
Real Time Clock	n	No change	Software
Peripheral Device	n	Device Channel #	Software
CSS-CSS Dialog	n	Interrup-ing CSS Channel #	Software

3.5.2.1 LEVEL ACTIVITY FLAGS

64 Level Activity Flag (AF) bits in four dedicated locations (20 - 23 hex) are maintained to indicate the processor levels that are currently active (that is, ready for execution - see Figure 3-3). Usually, the level currently executing corresponds to the most significant bit set, and its number is stored in S.CL. Activity flag bits are set by interrupt requests and are set and/or cleared by the LEV instruction.

3.5.2.2 INTERRUPT VECTORS

64 Interrupt Vectors (IVs) in 128 dedicated locations (80 - FF) are maintained. The IVs have a one-to-one relationship with the AFs (refer to Figure 3-3 and 3-4). They point to Interrupt Save Areas (ISA). An IV points to the DEV word of the ISA.

3.5.2.3 INTERRUPT SAVE AREA

The context for each level is stored in main memory in an area called the Interrupt Save Area (ISA). See Figures 3-3 and 3-4 for the format of the ISA. The ISA is pointed to by an IV associated with each specific level. Thus, the currently executing process running at level n, as well as the interrupting process running at a higher priority level m, will each have a unique IV pointing to its respective ISA. If the two IVs point to the same ISA, no context swap takes place, but a change to the higher priority level takes place. Once the context swap and level change have occurred, the interrupting process starts executing at its assigned level. Definitions for the entries in an ISA are listed after Figures 3-7 and 3-8.

- o HARDWARE CONTEXT POINTER (HCP) (M6X and M6XE only) - A two-word entry that points to a 32-word Hardware Context Area (HCA) in memory, if the H bit in ISM2 equals one.

- o ADDRESS SPACE VECTOR (ASV) - A two-word entry that points to the base of a segment table in memory. Refer to subsection 4.3.1.3.

- o TASK SEGMENT TABLE LIMIT (TSTL) (EMMU only) - A one word entry that defines the maximum addressable segment available to the task. Refer to subsection 4.3.1.4.

- o TRAP SAVE AREA POINTER (TSAP) - A two-word entry that points to a Trap Save Area (TSA) during trap processing. As a pointer, its two high-order bits consist of a ring number and should be set to zero.

- o INTERRUPTING DEVICE ID (DEV) - A one-word entry used to identify the interrupting source. The format of DEV for external interrupts is:

  - For device and CSS-CSS interrupts, set DEV to:

```
               0                   9 10          15
               |                    |              |
     DEV =     | CHANNEL NUMBER OF  | INTERRUPTING |
               | INTERRUPTING UNIT  | LEVEL NUMBER |
               |____________________|______________|
```

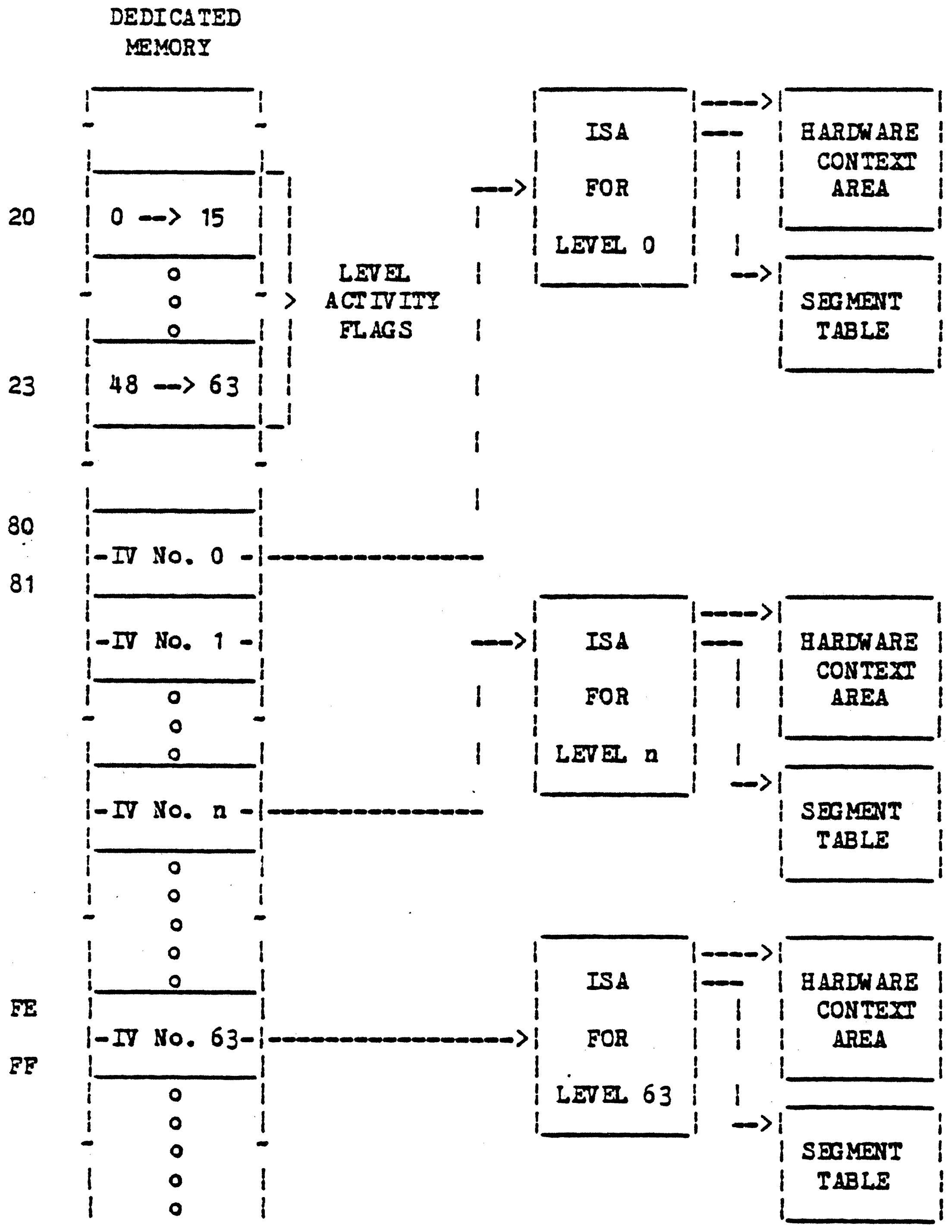
  - For Real Time Clock interrupts, DEV is not changed.

  - For Watch Dog Timer interrupts, DEV is not changed.

  - For internal interrupts, DEV is not changed.

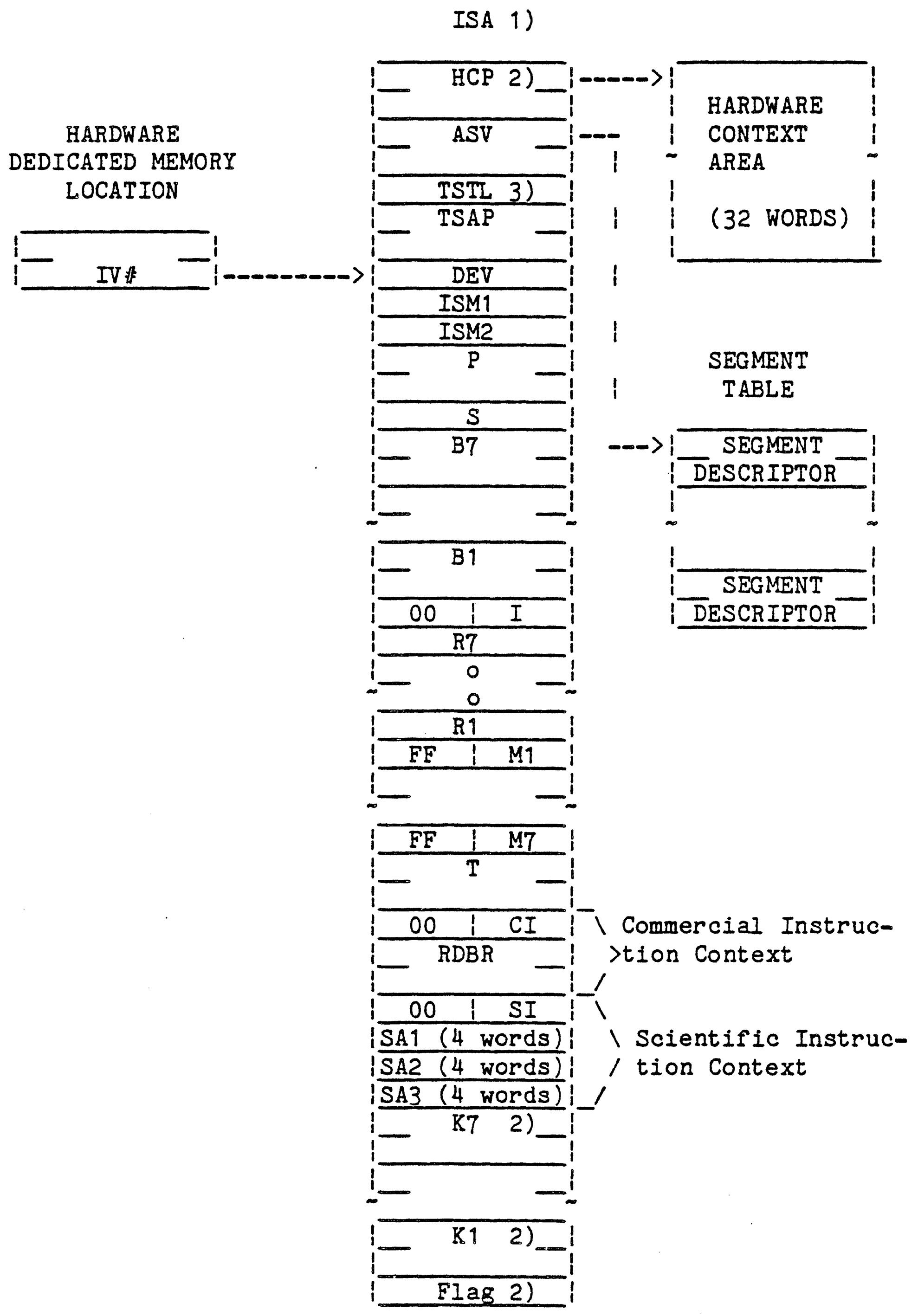
  - For power fail interrupts, DEV is not changed.

  Note that the IV points to DEV and IV relative addressing is relative to this entry.



where IV No. n = Interrupt Vector for Level No. n
ISA = Interrupt Save Area

Figure 3-3 Interrupt Constructs



NOTES

1)ISA shown above assumes all defined bits of ISM1 and ISM2 equal to One.
2) Applicable to M6X and M6XE only.
3) Applicable to EMMU models only, else RSU.

Figure 3-4 Interrupt Save Area (Sheet 1 of 2)

where IV	=	Interrupt Vector
HCP	=	Hardware Context Pointer
ASV	=	Address Space Vector
TSTL	=	Task Segment Table Limit
TSAP	=	Trap Save Area Pointer
DEV	=	Interrupting Device ID
ISM1, ISM2	=	Interrupt Save Mask
T	=	Stack Address Register
CI	=	Commercial Instruction Indicator Register
RDBR	=	Remote Descriptor Base Register
SI	=	Scientific Instruction Indicator Register
SA	=	Scientific Accumulator

Figure 3-4 Interrupt Save Area (Sheet 2 of 2)

o INTERRUPT SAVE MASK 1 (ISM1) - A one-word entry used in the save/restore hardware process. Its format is:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ISM1	M 1	R 1	R 2	R 3	R 4	R 5	R 6	R 7	I	B 1	B 2	B 3	B 4	B 5	B 6	B 7

Processor registers selected by the mask bits (bit = 1) are stored in memory in consecutive locations starting at the specified location. The mask bits are scanned from right (bit 15) to left (bit 0); ISM1 is scanned first, then ISM2. If the mask bit is a one, the corresponding register is saved or restored as applicable. If the mask bit is a zero, the corresponding register is not saved or restored and, in addition, the word or words required by the register is/are assumed not to be in the ISA.

o INTERRUPT SAVE MASK 2 (ISM2) - A one-word entry used in the save/restore process. Its format is:

	0	1 3	4	5	6	7	8	9	13	14	15
ISM2	MU	NATSAP	RFU	K *	S	C	H *	LA	RFU	T	M

* Applicable to M6X and M6XE only.

The following defines the ISM2 mask bits.

- MU = MMU Context. If set, then an Inrush operation is requested at context load time. Refer to subsection 4.3.4.

- NATSAP = NATSAP Select. The NATSAP Select field is used to select which NATSAP, and therefore which TSA pool, will be used during Trap processing. The field is encoded as follows:

NATSAP Select	NATSAP Used	Memory Location
0 0 0	0	00010
0 0 1	1	0000E
0 1 0	2	0000C
0 1 1	3	0000A
1 X X	RFU	RFU

- K = K1 - K7 (M6X and M6XE only). If set, then save/restore K1 - K7 in/from the specified locations. If not set, do not save/restore K1 - K7, and, the words required by the registers are assumed not to be in the ISA.

- S = Scientific Context. 'If set, then save/restore the scientific indicator (SI) and the scientific accumulators (SAs) in/from the specified locations. If not set, then do not save/restore the scientific context; the words required by the registers are assumed not to be in the ISA.

- C = Commercial Context and RDBR. If set, then save/restore the commercial indicator (CI) register and the RDBR in/from the specified locations. If not set, then do not save/restore the commercial context and RDBR. The words required by the registers are assumed not to be in the ISA.

- H = Hardware Context Area (HCA) Indicator (M6X and M6XE only). If H is set then make certain long instructions interruptable. The CSS uses the HCA to store any intermediate instruction context.

- LA = Level Active (M6X and M6XE only). This bit is set by the software to request the CSS to write a Flag word (at the end of the ISA), after having saved all the process context in the ISA. Flag is written only if the context change is caused by an LEV instruction.

- T = Stack Address Register. If set, then save/restore the stack address register in the specified locations. If not set, then do not save/restore the register. The words required by the register are assumed not to be in the ISA.

- M = M2 through M7. If set, then save/restore M2 through M7 (M2 through M5 for CR41E) in the specified locations. Six words are assumed allocated in all models. If not set, do not save/restore the registers. The six words are assumed not to be in the ISA.

o PROCEDURE COUNTER (P) - A two-word entry used to save/restore P.

- S REGISTER (S) - A one-word entry used to save/restore S. Refer to subsection 3.3.4.1.

- BASE REGISTERS (B) - Up to seven two-word entries used to save/restore B1 - B7. Mask bits in ISM1 determine which entries are to be saved/restored.

- INDICATOR REGISTER (I) - A one-word entry used to save/restore I. A mask bit in ISM1 determines if the entry is to be saved/restored. Since I is eight bits, it is stored in the right half of the entry with the left half set to 00.

- WORD OPERAND REGISTERS (R) - Up to seven one-word entries used to save/restore R1 - R7. Mask bits in ISM1 determine which entries are to be saved/restored.

- MODE REGISTER (M) - Up to seven one-word entries used to save/restore M1 - M7. Mask bits in ISM1 and ISM2 determine which entries are to be saved/restored. Since an M register is eight bits, it is stored in the right half of ther entry, with the left half set to FF.

- STACK REGISTER (T) - A two-word entry used to save/restore T. A mask bit in ISM2 determines if the entry is to be saved/restored.

- COMMERCIAL INDICATOR REGISTER (CI) - A one-word entry used to save/restore CI. The C mask bit in ISM2 determines if the entry is to be saved/restored. Since CI is eight bits, it is stored in the right half of the entry with the left half set to 00.

- REMOTE DESCRIPTOR BASE REGISTER (RDBR) - A two-word entry used to save/ restore the RDBR. It is saved/restored along with CI when so specified by the C bit in ISM2. The RDBR ring number is saved/restored as is.

- SCIENTIFIC INDICATOR REGISTER (SI) - A one-word entry used to save/restore SI. The S mask bit in ISM2 determines if the entry is to be saved/restored. Since SI is eight bits, it is stored in the right half of the entry, with the left half set to 00.

- SCIENTIFIC ACCUMULATORS (SAs) - Three four-word entries used to save/restore the scientific accumulators. These entries are saved/restored along with SI when so specified by the S bit in ISM2. The format of this area is as follows:

```
        _ ____________________
       | |      |   |         |
       | | (MSB)e | S | (MSB)f |
       | |--------------------|
       | |          f         |
       | |--------------------|
SA1  - | |          f         |
       | |--------------------|
       | |          f         |
       |-|--------------------|
SA2      |    Same as SA1     |
         |--------------------|
SA3      |    Same as SA1     |
         |____________________|
```

where SA1, SA2,and SA3 = Scientific Accumulator Registers for Registers 1,2 and 3; e= Exponent; s = Sign; and f = Fractional mantissa.

- DOUBLE WORD OPERAND REGISTERS (K) (M6X and M6XE only) - Up to seven two-word entries used to save/restore K1 - K7. The K mask bit in ISM2 determines if the entries are to be saved/restored.

- FLAG (M6X and M6XE only) - A one word entry which is set by the processor to FFFF after having saved all process context in the ISA. Flag is written only if the LA mask bit in ISM2 = 1 and the context change is caused by an LEV instruction. Flag is not updated if context change is caused by an interrupt.

3.5.2.4 HARDWARE CONTEXT AREA (M6X and M6XE only)

The Hardware Context Area (HCA) consists of 32 words and is used by the CSS to store intermediate instruction context when certain long instructions are specified to be interruptable (ISM2.H = 1).

The HCA is pointed to by the HCP contained in the ISA.

3.5.3 External Interrupt Handling

External interrupts (EIs) are typically sensed between instructions. In a few cases (for example, when executing interruptable instructions), EIs are also sensed during instruction execution.

When an EI is sensed, the CSS compares the current interrupt level number (CL) with the interrupting interrupt level number (IL). If CL $\leq$ IL, then the EI is not honored and the interrupt source is told to stack the interrupt. If CL > IL, the EI is honored. Context is saved in the ISA of the interrupted level and a level change to the interrupting level is performed and its context loaded. The DEV word in the ISA of the interrupting level is updated to indicate the interrupt source. The AF of the interrupted level is left on and the AF of the interrupting level is set.

If during a level change operation the IV of the interrupting level is found to be NULL, then the interrupt is ignored and a scan and dispatch function performed. (Refer to Figure 3-5.)

3.5.4 Internal Interrupt Handling

Internal Interupts (IIs) occur during the execution of an instruction. In the case of the LEV instruction, the instruction specifies that an II be generated. In the case of a TSA-related interrupt, an interrupt to level 2 is generated because during instruction excution a trap was detected and the Trap Save Area (TSA) being used is the last one in the pool. Refer to Figure 3-5.

```
                    *******
                   * START *
                    *******
                       |
                       v
                      o o
                    o( A )o
                      o o
                       |
                       v
              ____________________
             ( DETERMINE CSS STATE )
              ____________________
                       |
                       v
              ______________________
             /    IS CSS IN  THE    \  YES        o o
            <        HARDWARE        >--------->o( D )o
             \__ERROR HALT STATE ?__/              o o
                       | NO                    (Sheet 4)
                       v
              ______________________
             /     IS CSS IN        \  YES        o o
            <   THE OPERATOR HALT    >--------->o( E )o
             \______STATE ?_________/              o o
                       | NO                    (Sheet 4)
                       v
              ______________________
             /     IS CSS IN        \  YES        o o
            <     THE PROCEDURE      >--------->o( F )o
             \____HALT STATE ?______/              o o
                       | NO                    (Sheet 5)
                       v
              ______________________
             / IS CSS IN EITHER THE \  YES        o o
            <   LEVEL 63 , HALT OR   >--------->o( F1)o
             \__SUPER HALT STATE ?__/              o o
                       | NO                    (Sheet 5)
                       v
            _________________________
           (  THE CSS IS IN THE RUN  )
           ( STATE.  THE FLOW STARTS )
           (  WITH THE FETCHING OF   )
           (__THE NEXT INSTRUCTION___)
                       |
                       v
              ______________________
             | FETCH NEXT INSTRUC- |
             |________TION_________|
                       |
                       v
                      o o
                    o( B )o
                      o o
```

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 1 of 10)

```
                          o o
                        o( B )o
                          o o
                           |
                           v
                 ______________________
                /          IS          \ YES         o o
               <   INSTRUCTION AN LEV   >-------->o( L )o
                \____INSTRUCTION ?_____/             o o
                           | NO                   (Sheet 9)
                           v
                  ____________________
                 | START INSTRUCTION |
                 |_____EXECUTION_____|
                           |
----------------------->|
|                          v
|                 ______________________
|         NO    /   IS INSTRUCTION      \
|        -----<     INTERRUPTABLE        >
|        |      \__________?____________/
|        |                 | YES
|        |                 v
|        |        ______________________
|        |      /         IS AN          \  YES         o o
|        |     <        INTERRUPT         >-------->o( C )o
|        v      \_______PENDING ?_______/              o o
|     o   o                | NO                   (Sheet 3)
|   o( B1 )o------------->|
|     o   o                v
|                 ______________________
|               /        HAS A TSA       \  YES         o o
|              <   (LEVEL 2) INTERRUPT    >-------->o( T )o
|               \_______BEEN POSTED ?___/              o o
|                          | NO                   (Sheet 10)
|                          v
|                 ______________________
|      NO       /           IS           \
-------------<  INSTRUCTION EXECUTION  >
                \_______COMPLETE ?______/
                           | YES
                           v
                  ______________________
                /         IS AN          \  NO          o o
               <        INTERRUPT         >-------->o( A )o
                \_______PENDING ?_______/              o o
                           | YES                  (Sheet 1)
                           v
                          o o
                        o( C )o
                          o o
```

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 2 of 10)

```
                              o o
                            o( C )o
                              o o
                               |
                               v
                      ______________________
                     (   INTERRUPT IS AN    )
                     (__EXTERNAL INTERRUPT__)
                               |
                               v
                       ____________________
                      | IL <-- LEVEL # FROM |
                      |__INTERRUPT SOURCE___|
                               |
                               v           YES          o o
                          < IS IL < CL ? >--------->o( I )o
        o  o                   | NO                     o o
      o( C2 )o---------------->|                    (Sheet 6)
        o  o                   v
                     _________________________
                    | INFORM INTERRUPT SOURCE |
                    |_TO STACK THE INTERRUPT__|
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               v
                          /  IS THE   \  NO          o o
                         < CSS IN THE RUN >------->o( A )o
                          \__STATE ?__/              o o
                               | YES              (Sheet 1)
                               v
         o o      YES    / IS INSTRUCTION \  NO         o  o
       o( A )o<-------< EXECUTION COMPLETE >------->o( B1 )o
         o o             \_______?________/             o  o
      (Sheet 1)                                      (Sheet 2)
```

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 3 of 10)

```
                              o o
                            o( D )o
                              o o
                               |
                               v
                        _________________
                       (   CSS IS IN THE    )
                       (      HARDWARE      )
                       (__ERROR HALT STATE__)
                               |
                               v
                          ______________
                         | CSS IS FROZEN |              o o
                         | AND WAITS FOR |------->o( A )o
                         |_AN INITIALIZE_|              o o
                                                      (Sheet 1)


                              o o
                            o( E )o
                              o o
                               |
                               v
                          _____________
                         (   CSS IS IN   )
                         ( THE OPERATOR  )
                         (__HALT STATE___)
                               |
                               v
                          ____________
                  YES    /   IS AN    \
               -------<   INTERRUPT    >
               |         \_PENDING ?_/
               |               | NO
               |               v
               |          ____________
               |         |  WAIT FOR  |             o o
               |         |  OPERATOR  |----->o( A )o
               |         |INTERVENTION|             o o
               |                                   (Sheet 1)
               ---------------
                               |
                               v
                         _____________
                        / IS INTERRUPT \  NO         o  o
                       <  A POWER FAIL  >-------->o( C2 )o
                        \_INTERRUPT ?__/               o  o
                               | YES               (Sheet 3)
                               v
                              o o
                            o( C )o
                              o o
                            (Sheet 3)
```

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 4 of 10)

```
                 o o                          o o
               o( F )o                      o( F1)o
                 o o                          o o
                  |                            |
                  v                            v
        ______________________      _________________________
       (    CSS IS IN THE    )     (    CSS IS IN EITHER    )
       (      PROCEDURE      )     ( THE LEVEL 63, HALT OR  )
       (_____HALT STATE______)     (____SUPER HALT STATE____)
                  |                            |
                  ------------------------------
                                |
                                v
                           ____________
                          /   IS AN   \  YES           o o
                         <  INTERRUPT  >------------>o( C )o
                          \_PENDING ?_/                o o
                                | NO                (Sheet 3)
                                v
                      _____________________
                 NO  /  HAS THE OPERATOR   \
   ----------------< DEPRESSED THE EXECUTE >
   |                 \________KEY ?________/
   |                            | YES
   |                            v
   |            ___________________________________
   | LEVEL 63, /                                  \
   |<---------< IN WHICH HALT STATE IS THE CSS IN? >
   | HALT OR   \__________________________________/
   | SUPER HALT                 |
   |                            | PROCEDURE
   |                            v
   |                 ________________________
   |                | SKIP HALT INSTRUCTION  |------
   |                |_AND ENTER THE RUN STATE|     |
   |                                               |
   |                      _____________            |
   --------------------->| WAIT FOR AN |           |
                         |__INTERRUPT__|           |
                                |                  |
                                v                  |
                               o o                 |
                             o( A )o<---------------
                               o o
                            (Sheet 1)
```

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 5 of 10)

```
                          o o
SEE NOTE ON SHEET 10    o( I )o
                          o o
                           |
                           v
                  ______________________
                 | SET AF CORRESPONDING |
                 |  TO IL.  UPDATE DEV  |
                 |__WORD IF APPLICABLE__|
                           |                 o  o
                           |<-------------o( I1 )o
                           v                 o  o
                  ______________________      ^
                 | SCAN FOR AL STARTING |     |
                 |___FROM AF 0 --> 63___|     |
                           |                  |
                YES        v                  |
   -----------------<_IS AL = CL ?_>          |
   |                       | NO               |
   |    o o     NO         v                  |
   |  o( J )o<----<_IS ALV = NULL ?_>         |
   |    o o                | YES              |
   | (Sheet 7)             |                  |
   |           YES         v                  |
   |         -------<_IS AL = 63 ?_>          |
   |         |             | NO               |
   |         |             v                  |
   |         |      |_RESET AL AF_|           |
   |         |             |                  |
   |         |             --------------------
   |         ---------------
   |                       |
   |                       v
   |               ___________________
   |              | ENTER LEVEL  63   |
   |              |___HALT STATE______|
   |                       |
   |                       v
   |                      o o
   -------------------->o( A )o
                          o o
                       (Sheet 1)
```

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 6 of 10)

```
                                  o o
        SEE NOTE ON SHEET 10    o( J )o
                                  o o
                                    |
             c o       YES          v
                            _________________
           o( A )o<--------< IS ALV = CLV ? >
             o o                    | NO                o  o
          (Sheet 1)                 |<---------------o( J1 )o
                                    v                   o  o
                       _________________________
                      ( PERFORM A  LEVEL CHANGE )
                                    |
                       YES          v
                            ________________
                 ----------< IS CLV = NULL? >
                 |                  | NO
                 |                  v
                 |        ______________________
                 |       | STORE CSS CONTEXT    |
                 |       |____INTO CL ISA_______|
                 |                  |
                 |   NO             v
                 |      _________________________
                 |<----< IS THIS AN M6X OR M6XE?>
                 |                  | YES
                 |                  v              NO
                 |      ________________________
                 |     < IS THIS AN LEV INSTR.? >-------
                 |                  | YES              |
                 |                  v                  |
                 |        ____________________         |
                 |       /        IS          \    NO  |
                 |      < LA BIT IN ISM2 = 1? >------->|
                 |       \____________________/        |
                 |                  | YES              |
                 |                  v                  |
                 |          __________________         |
                 |         | SET FFFF--> FLAG |        |
                 |                  |                  |
                 |                  |<------------------
                 |                  v
                 |        ____________________
                 |   NO  /     IS THERE       \
                 |<-----< CONTEXT TO BE SAVED >
                 |       \___IN THE CL HCA ?__/
                 |                  | YES
                 |                  v
                 |          __________________
                 |         | STORE CONTEXT    |
                 |         |__INTO CL HCA_____|
                 |                  |
                 ------------------>|
                                    |
                                    v
                                  o o
                                o( K )o
                                  o o
                               (Sheet 8)
```

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 7 of 10)

```
                                      o o
           SEE NOTE ON SHEET 10     o( K )o
                                      o o
                                       |
                                       v
                     ____________________________________
                    | LOAD CONTEXT FROM AL ISA INTO CSS  |
                    ------------------------------------
                                       |
                                       v
                               _____________________
                    NO   /          IS THERE          \
                 --------< CONTEXT IN THE AL HCA >
                 |        \___________?___________/
                 |                     | YES
                 |                     v
                 |              ___________________
                 |             | LOAD CONTEXT FROM |
                 |             |  AL HCA INTO CSS  |
                 |              -------------------
                 |                     |
                 |                     v
                 |        _______________________________
                 |       (    USE CONTEXT IN THE AL HCA    )
                 |       (     TO RESUME INSTRUCTION       )
                 |       (_________EXECUTION_______________)
                 |                     |
                 ------------------>|
                                       v
                  ___________________________________________
                 | BROADCAST ON BUS FOR PENDING INTERRUPTS |
                  -------------------------------------------
                                       |
                                       v
                                      o o
                                    o( A )o
                                      o o
                                   (Sheet 1)
```

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 8 of 10)

```
                                            o o
           SEE NOTE ON SHEET 10           o( L )o
                                            o o
                                             |
                                             v                  ***********
                        ____________________________                           
                       / DOES INSTRUCTION MEET THE \  NO     * TRAP # 13 *
                      <    PRIVILEGE REQUIREMENTS   >---->* PRIVILEGE *
                       \____________?______________/       * VIOLATION *
                                     | YES                  ***********
                                     v
                        | IL <-- IL FIELD FROM   |
                        |_LEV INSTRUCTION OPERAND_|
                                     |
         ------------------------------
         |
         v
  ____________________          ____________________
 /       S = 1        \  YES   /       Q = 1        \  NO
<    IS CL TO BE       >------><  IS A QUICK LEVEL    >-----------------
 \___SUSPENDED?_______/        \ CHANGE REQUESTED? /                  |
         | NO                          | YES                          |
         |                             v                              v
         |                  | PERFORM QUICK LEVEL |       | SCHEDULE LEVEL IL   |
         |                  | CHANGE BY           |       |  BY SETTING THE AF  |
         |                  | - SETTING AF FOR IL |       |_CORRESPONDING TO IL_|
         |                  |_- SET ILV <-- CLV___|                 |
         |                             |                            |
         |                             v                            v
         |                  |   SUSPEND CL BY     |       |   SUSPEND CL BY     |
         |                  |  CLEARING ITS AF,   |       |  CLEARING ITS AF,   |
         |                  |__GO TO LEVEL IL_____|       |_SCAN AND DISPATCH___|
         |                             |                            |
         |                             ------------------           |
         v                                              |           |
                            | PERFORM QUICK LEVEL |     |           |
  ____________________      | CHANGE BY           |     |           |
 /       Q = 1        \  YES| - SETTING AF FOR IL |---->|           |
<   IS A QUICK LEVEL   >--->| - SET ILV <-- CLV   |     |           |
 \ CHANGE REQUIRED ? /      |_- GO TO LEVEL IL____|     |           |
         | NO                                           |           |
         |                                              |           |
         v                  | SCHEDULE LEVEL IL   |     v           |
  ____________________      |  BY SETTING THE AF  |    o o          |
 /       D = 1        \  YES| CORRESPONDING TO IL,|-->o( A )o       |
<  IS INTERRUPT TO BE  >--->|  NO LEVEL CHANGE    |    o o          |
 \____DEFERED ?_______/     |__IS PERFORMED_______|  (Sheet 1)      |
         | NO                                                       |
         |                                                          |
         v                                                          |
| SCHEDULE LEVEL IL   |                o o                          |
|  BY SETTING THE AF  |-------------->o( I1 )o<----------------------
| CORRESPONDING TO IL,|                o o
|_SCAN AND DISPATCH___|             (Sheet 6)
```

Figure 3-5 Interrupt Flow Chart (Sheet 9 of 10)

```
                         o o
                       o( T )o
                         o o
                          |
                          v
                 ______________________
                | SET AF = 2 SINCE     |
                | THIS INTERRUPT TYPE  |
                |   IS PROCESSED AT    |
                |______LEVEL 2_________|
                          |
                          v
                 ___________________
                |_SET AL TO LEVEL 2_|
                          |
                          v          YES     **********
                  < IS ALV = NULL >------->*   SUPER   *
                          | NO              *   HALT    *
                          v                  **********
                        o   o
                       o( J1 )o
                        o   o
                       (Sheet 7)
```

Where:

IL	-	Interrupt Level No.
AL	-	Activate Level No.
CL	-	Current Level No.
AF	-	Activity Flag
ALV	-	AL's Interrupt Vector
CLV	-	CL's Interrupt Vector

NOTE

The CSS (M6X and M6XE only) can not tolerate a trap (i.e., TV15 or 17) when accessing any of the above constructs. If a trap is encountered, then the CSS will post a super halt. In a CR41E and M5XE a trap will cause unspecified results.

Figure 3-5 Interrupt Sequence Flow Chart (Sheet 10 of 10)

3.5.4.1 LEV INSTRUCTION

The LEV instruction is used to set/clear "Level Activity Flags" (AFs), inhibit interrupts, enable interrupts, and so forth. It uses a 16-bit operand to control the various sequences of actions. The format of the operand is as follows:

```
  0   1   2          7   8   9  10          15
 |   |   |            |   |   |              |
 | S | D | 0 0 0 0 0 0 | Q | 0 |      IL      |
 |___|___|____________|___|___|______________|
```

where: S = Suspend current level
D = Defer Interrupt
Q = Quick level change (inhibit)
IL = Interrupting Level #
CL = Current Level

The actions performed are summarized as follows:

S	Q	D	A C T I O N
0	0	0	Schedule level IL, scan and dispatch
0	0	1	Schedule level IL, defer interrupt
0	1	X	Inhibit to level IL
1	0	X	Schedule level IL, suspend CL, scan and dispatch
1	1	X	Inhibit to level IL and Suspend CL

where Schedule = Set activity flag for IL;

Scan and dispatch = Scan AF, find highest priority active level, save context of current running level, and restore context of the highest priority active level;

Defer Interrupt = Do not scan nor dispatch. Continue executing at CL;

Inhibit = Set AF for IL. Change level of running process to IL. Set IV of IL (ILV) to IV of CL (CLV). Do not perform context save/ restore or change the process address space. (Used for quick level change to higher priority level.); and

Suspend = Clear AF for CL.

The above functionality and interrupt related functions are described in Figure 3-5. The hardware guarantees that a process is always active at level 63; that is, the AF corresponding to 63 is treated as if it is always set. If the IV of level 63 equals NULL, then the CSS performs the equivalent of a Halt instruction, entering the instruction halt state. In this state, the CSS minimizes the use of bus cycles while waiting for an interrupt. If the IV of level 63 is not NULL, the processor treats this level like any other.

3.5.4.2 TSA RELATED INTERRUPT

This interrupt type is posted by the processor if during Trap execution the TSA being used is the last one in the pool. An interrupt to level 2 is generated to report this condition to the software. Refer to Figure 3-5.

3.5.5 Power Fail Interrupt Handling

Power fail interrupts (PFIs) are, like EIs, typically sensed between instructions. In a few cases (for example, when executing interruptable instructions), PFIs are also sensed during instruction execution.

When a PFI is sensed, the CSS determines the value of the current interrupt level number (CL):

- o If CL = 0, then the CSS enters the operator halt state.
- o If CL > 0, the PFI is honored. Context is saved in the ISA of the interrupted level and a level change to level zero is performed and its context loaded. The DEV word in the ISA of level zero is not changed. The AF of the interrupted level is left on and the AF of level zero is set. Refer to Figure 3-5.

Software reaction to a PFI is expected to be as follows:

- o A minimum of Level zero context is requested to be loaded;
- o An instruction is executed to update an indicator to log the fact that a PFI has occured; and
- o A halt instruction is executed.

Note that the above defined functionality applies to all CSSs whether the CSS is the master or a slave in the case of dual processor systems.

3.6 TRAPS

3.6.1 Concept

A trap is an event that can occur during the execution of an instruction and indicates that software intervention is required. A set of trap vectors and a Trap Save Area (to hold the trap context) are used to pass control to the software and to provide the software with the necessary trap context.

As a function of the trap type, the appropriate trap vector is used to identify the trap handler. The trap handler, using the trap context, can then process the trap condition. Following successful resolution of the trap condition, the trap handler performs a Return to resume execution of the trapped procedure. Precisely which instruction is executed following the Return is a function of the trap type and is defined by the trap handler, by updating the trap context in the trap save area, if needed.

Since trap handling can require a change in the process' execute privilege, the trap vector is used to define the execute privilege of the trap handler.

3.6.2 Trap Handling

Each type of trap is associated with a Trap Vector (that is, a Trap handler pointer) that is stored in a dedicated main memory location. The various events that can cause a trap and their associated vector numbers are given in Table 3-4.

The following constructs are used (refer to Figure 3-6):

- o Interrupt Vectors (IVs)
- o Interrupt Save Area (ISA)
- o Interrupt Save Mask Word 2 (ISM2)
- o Trap Save Area (TSA)
- o Trap Vectors (TVs)
- o Next Available TSA Pointer (NATSAP).
- o Trap Save Area Link (TSAL)
- o Trap Save Area Pointer (TSAP)

The CSS requires that the above constructs be in main memory. The CSS (M6X and M6XE only) can not tolerate a trap (i.e., TV15 or 17) when accessing any of the above constructs. If a trap is encountered, then the CSS will post a super halt. In a CR41E and M5XE a trap will cause unspecified results.

The CSS does not perform any access right checks when it accesses any of the above constructs. Specifically, it assumes to have ring 0 priviledge and ignores the access permit flags.

Trap handling is performed using Trap Save Areas (TSA). Detection of a trap condition causes the CSS to save the trap context into a TSA and then to transfer control to the appropriate trap handler. Figure 3-7 shows the trap context.

The CSS uses the NATSAP selected by ISM2 (1-3) to access one of the four TSA pools. If the selected TSA pool is empty (that is, the selected NATSAP contains a NULL link), the CSS will enter the super halt state; otherwise, the first available TSA stack entry is unlinked from the selected pool and linked to the ISA of the current level. The TV# identifies the trap handler procedure, its execution ring and entry point.

The execute privilege of the trap handler is determined as a function of the TV. For TV = even, no ring change takes place and the trap handler uses the same privilege as the trapped prodecure. For TV = odd, a ring change to ring 0 occurs and the trap handler runs in ring 0.

Before entering the trap handler, B3, whose content was saved in the TSA, is loaded with a pointer to the TSA's A field. In addition, TSAL is checked to insure that it is not equal to NULL (indicating that this TSA is the last stack frame in the pool). If NULL, a level 2 interrupt is generated before the trap handler is entered.

The generic instruction RTT must be used to return from a trap which uses a TSA. The RTT uses the current level ISA to access the TSA, from which it restores the values of all visible registers saved during the trap. The TSA frame is returned to the head of the appropriate memory pool (as defined by ISM2 (1-3)).

Table 3-4 Trap Vectors and Events

VECTOR #	EVENT	
1	Monitor call (MCL instruction)	
2	Trace trap 1) (debug) or BRK instruction	
3	Scientific Opcode not supported	
4	RSU	
5	Address syllable or opcode (not Scientific) not supported	
6	Integer register overflow	1)
7	Scientific divide by zero	
8	Scientific exponent overflow	
9	Stack underflow	
10	Stack overflow	
11	RFU	
12	Recursive remote descriptor usage	
13	Unprivileged use of privileged operation	
14	Unauthorized reference to protected memory	
15	Reference to unavailable resource	
16	Program error	
17	Memory or bus error	
18	Segment descriptor access error	2)
19	Scientific exponent underflow	1)
20	Scientific program error	
21	Scientific significance error	1)
22	Scientific precision error 1)	
23	Reference to unavailable resource by CIP or SIP	3)
24	Memory or bus error detected by CIP or SIP	3)
25	Commercial divide by zero	
26	Commercial specification error	
27	Commercial illegal character (data code)	
28	Commercial truncation error	1)
29	Commercial overflow	1)
30	CIP QLT error	4)
31	SIP QLT error	4)
32	Unauthorized reference to protected memory by CIP or SIP	3)
33	Illegal Argument detected during a scientific instruction	4)
34 thru 46	RFU	

NOTES

1) This trap occurs only if enabled.
2) This trap occurs only when in EMMU mode.
3) This trap is applicable only on systems that have a separate processor for CIP and/or SIP. For systems with no separate processor(s), TV23, 24 and 32 are mapped into TV15, 17 and 14 respectively.
4) This trap is applicable only on a M6X or M6XE.

During the execution of a trap or an RTT instruction, the CSS does not tolerate a NULL pointer. Detection of a NULL pointer results in the action(s) shown in Table 3-5.

Figure 3-8 gives a flow chart of the trap sequence. For information on the Return sequence, refer to the RTT instruction in Section 5.

3.6.3 Trap Vector Descriptions

Defined below are the trap types given in Table 3-4. For more information also refer to Figure 3-7 for a description of the trap context.

TV01 Monitor Call - This trap is posted whenever an MCL instruction is executed.

TV02 Trace or BRK Instruction - This trap is posted whenever either a BRK instruction is executed or whenever a trace trap is requested in M1 and the branch condition is true during the execution of an LNJ, JMP, ENT or any branch instruction.

TV03 Scientific Opcode not supported - This trap is posted whenever a scientific instruction is encountered by the CSS but no SIP is configured.

TV05 Address Syllable or Opcode (not scientific) not supported - This trap is posted whenever an unassigned Address Syllable is used or an unspecified opcode is encountered by the CSS.

This trap is also posted, in a CR41E, if the MTM or STM instructions specify M5 or M7.

TV06 Integer Register Overflow - This trap is posted whenever an overflow trap is requested in M1(#) and an overflow occurs during the execution of any of the following general (R register) instructions:

ADD, SUB, MUL, DIV, ADV, MLV, SAL, DAL, AID and SID.

This trap is also posted whenever an overflow trap is requested in M2(#) and an overflow occurs during the execution of any of the following EII (K register) instructions:

KADD, KSUB, KMUL, KDIV, KAL, KADV and KMLV.

The trap is posted upon completion of the instruction. The register remains unchanged, on an overflow, for the following instructions:

MUL, DIV, MLV, KDIV, KMUL and KMLV.

For all the other instructions, the register is modified, on an overflow.

TV07 Scientific Divide by Zero - The Divide by Zero (DZ) trap is generated whenever the divisor of a scientific divide (SDV) instruction is equal to Zero. Operands remain unchanged.

TV08 Scientific Exponent Overflow - This trap is posted whenever the exponent (e) of the floating point result is greater than +63. The scientific instructions that can cause this trap are:

SLD, SAD, SSB, SML, SDV, SST and SSW.

The trap is posted upon completion of the instruction and the result field(s) are altered.

TV09 Stack Underflow - This trap is posted when a RLQ instruction returns the last frame on the stack.

TV10 Stack Overflow - This trap is posted when an ACQ instruction requests a frame that is larger in size than the available space on the stack.

TV12 Recursive Remote Descriptor Usage - This trap is posted when the remote descriptor of a commercial instruction points to another remote descriptor.

TV13 Unprivileged Use of Privileged Operation - A privileged instruction was executed in a ring other than zero or one. Refer to subsection 4.3.1.1.

TV14 Unauthorized Reference to Protected Memory - During instruction execution one of the following conditions was detected:

- o Process did not have the required privilege to execute the instruction,
- o Process did not have the required privilege to read the operand,
- o Process did not have the required privilege to write the operand.

TV15 Reference to Unavailable Resource -

- o During a Megabus operation (e.g., to memory, I/O controller, CSS, etc.), one of the following conditions occured:
  - - A deadman timeout condition was detected. Specifically, the memory address or I/O channel number does not exist or required parity(s) of address was incorrect.
  - - A Megabus operation timeout was detected. Specifically, an excess prinet delay was detected or a Megabus unit responds with excess waits or a Megabus unit accepts a read command but does not deliver requested data.
- o During the address translation process one of the following conditions was detected:
  - - Segment descriptor not valid,
  - - Segment size violation or
  - - Segment number > TSTL (EMMU Mode only).

o The high order bits of an LA where non zero. Refer to subection 3.2.1.2.

o During the execution of the VLD or CVP instructions when in SMMU Mode (applicable to M5XE or M6XE only), the Segment number in B5 is greater than 0F i.e., ([B5] > 0FXXXX).

o During the execution of the ASST instruction:

- The SST did not start in a valid segment or

- The SST was not contained within the physical memory space..

o During the execution of the ATST instruction, the TST did not start in a valid segment.

TV16 Program Error - During instruction decoding, the AS used is found to be illegal as follows:

o AS = REG and the instruction is a LAB, JMP, LNJ, ENT, SAVE, RSTR, LXA or IOLD-AAS;

o AS = IMO from Map 1 and the instruction is an LXA, or any EII using a data type with atom size less than a word;

o AS = IMO from Map 2 and the instruction is not a commercial instr;

o AS = AS3 from Map 2 and the instruction is not a commercial instr;

o AS = AS3 from Map 2 is being used by a remote descriptor;

o AS = =Kn and the instruction is a LAB, JMP, LNJ, ENT, SAVE, RSTR, LXA, IO, IOH or IOLD;

o AS = AS23 from Map 1 and the instruction is an EII instruction;

o The AS chosen by the escape AS EII1 is = ASN, EII1 or EII23 (refer to Figure 3-14);

o The AS chosen by the escape AS EII23 is = AS3 (see Figure 3-14);

During RTT instruction execution, [TSAP] = Null (there is no TSA);

During MMM or BSRCH instruction execution, [R6] < zero;

During LDT, STT, ACQ, or RLQ instruction decoding, an encoding error is detected in the second word of the instruction;

During LXA or EII instruction decoding, the data type used is found to be illegal.

During MMUD instruction decoding, an encoding error is detected in [R5];

During the execution of the RLQ instruction, CW = 0 at the start of RLQ execution or RLQ returns more space than exists on the stack.

During the execution of the RLQ or ACQ instructions, [T] = NULL.

Usage of a stack related AS resulted in a limit check violation.

During the execution of the ASD instruction:

- o [B5] > 0FXXXX and processor is in the SMMU mode (applicable only to M5XE and M6XE).
- o An attempt was made to invalidate Segment 0 in an M6X or M6XE in any Mode or in an M5XE in EMMU mode.

During the execution of the ATST instruction:

- o CSS was in SMMU mode.
- o The TST (M5XE only) was not double word aligned.

TV17 Memory or Bus Error - During a Megabus read operation (e.g., from memory, I/O controller, etc.) one of the following conditions was detected:

- o Memory controller indicated a memory read error via the RED Megabus lines; or
- o CSS detected a parity error in data received from cache or Megabus.

TV18 Segment Descriptor Access Error (EMMU Mode only)-

- o During the fetching, by the EMMU, of a segment descriptor from the EMMU Storage Array, a parity error was detected (M6XE only).
- o During the demand fetching, by the EMMU, of a segment descriptor from the Task Segment Table, one of the following conditions was detected:
  - A deadman timeout condition was detected. Specifically, the memory address does not exist or required parity(s) of address was incorrect; or
  - A Megabus operation timeout was detected. Specifically, an excess prinet delay was detected or the memory controller accepts a read command but does not deliver requested data.
  - A memory controller indicated a memory read error via the RED Megabus lines.
  - A segment descriptor is found not to be double word aligned (M5XE only). This implies that the TST is not double word aligned. Refer to subsection 4.3.4.

TV19 Scientific Exponent Underflow - This trap is posted whenever an underflow trap is requested in M5 and the exponent (e) of the floating point result is less than -64. The scientific instructions that can cause this trap are:

SLD, SAD, SSB, SML, SDV and SSW.

The trap is posted upon completion of the instruction and the result field(s) are altered.

TV20 Scientific Program Error - During scientific instruction decoding, the AS used is found to be illegal as follows:

- o AS = IMO and the instruction is an SST or SSW;
- o AS = REG or =Kn and the instruction is an SCZD or SCZQ;
- o AS = REG (specifying R4, R5, R6, R6/R7) or =Kn and the instruction is an SNGD or SNGQ;

TV21 Scientific Significance Error - This trap is posted during the conversion of a floating point number into an integer value if all of the following conditions are true:

- a significance error trap is requested in M5 and
- the integer does not fit in the destination register.

The trap is posted upon completion of the scientific instruction. Refer to subsection 8.1.3.

TV22 Scientific Precision Error - This trap is posted during the conversion of a floating point number into an integer value if all of the following conditions are true:

- a precision error trap is requested in M5,
- M4(R/T) is a zero and
- the integer fits in the destination register but there are non zero fractional digits.

The trap is posted upon completion of the scientific instruction. Refer to subsection 8.1.3.

TV25 Commercial Divide by Zero - The Divide by Zero (DZ) trap is generated whenever the divisor of a decimal divide instruction is equal to Zero. Note that the CI(OV) is also set.

TV26 Commercial Specification Error - The Illegal Specification (IS) trap is generated whenever any of the following conditions is detected:

- o An undefined Commercial op code is detected;

- o Any DD of an alphanumeric instruction is a packed decimal DD;
- o A decimal operand has zero length;
- o An operand in an Edit instruction has zero length;
- o A separate signed decimal operand consists of only a sign byte or half byte;
- o In a Move and Edit instruction, the receiving field length has not been exhausted but either there are no more micro-ops or the sending field length is exhausted;
- o Attempt to execute an illegal Move and Edit micro-op;
- o A DD2 specifies an IMO for other than DCM and ACM instructions;
- o In a DSH, DD1 specifies an IMO;
- o A DD3 specifies an IMO;

TV27 Commercial Illegal Character (data code) - The Illegal Character (IC) trap is generated whenever any of the following conditions is detected:

- o An illegal decimal digit is detected (i.e., low order four bits of digit are not 0-9);
- o An illegal sign digit is detected (i.e., a digit that is not one of the recognized sign values); or
- o An illegal overpunch digit is detected.

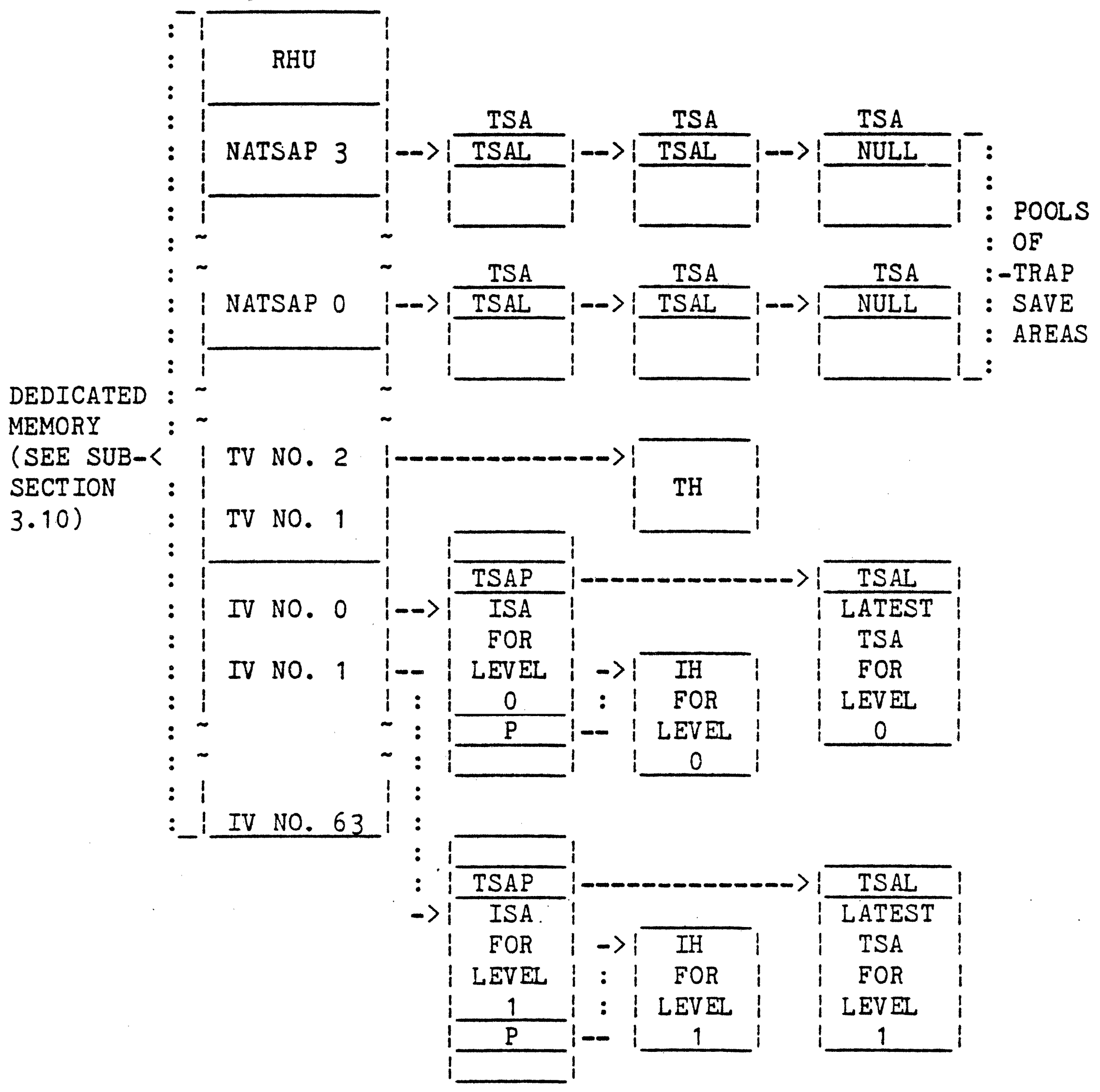
TV28 Commercial Truncation Error - The Truncation (TR) trap is generated conditionally [as a function of the setting of M3(TR)] whenever the receiving field of an alphanumeric instruction cannot contain all characters of the result.

Whether a trap occurs or not, the receiving field is altered [it will contain the leftmost part of the result and the CI(TR) will be set].

TV29 Commercial Overflow - The Overflow (OF) trap is generated conditionally [as a function of the setting of M3(OV)] whenever the receiving field of a decimal instruction cannot contain all the significant digits of the result (leading zeros are not considered significant) or, if during a shift left instruction a non-Zero digit is shifted out.

If a trap occurs [M3(OV) = 1], then instruction execution is suspended and the original operands remain unmodified. The CI(OV) indicator is set.

If no trap occurs [M3(OV) = 0], the receiving field is altered (it receives the least significant part of the result). The CI(OV) indicator is set.



where

IH	=	Interrupt Handler
IV NO. n	=	Interrupt Vector for Level No. n
NATSAP 0 - 3	=	Next Available TSA Pointers
RHU	=	Reserved for Hardware Use
TH	=	Trap Handler
TSA	=	Trap Save Area
TSAL	=	Trap Save Area Link
TSAP	=	Trap Save Area Pointer
TV NO. n	=	Trap Vector for Class No. n
ISA	=	Interrupt Save Area

Figure 3-6 Trap and Interrupt Constructs

TSA FORMAT

```
     ISA
   ________
  :________:    0                       15
  |_TSAP__|-->|_          TSAL          _|  n
  :________:  |           7 8            |  n+1
              |__TRAP #___|_COPY OF I____|  n+2
              |___________R3_____________|  n+3
              |__INSTRUCTION 1ST WORD____|  n+4
   ________   |___________Z______________|  n+5
  |__B3___|-->|_          A             _|  n+6
              |__________________________|  n+7
              |_          P             _|  n+8
              |__________________________|  n+9
              |_          B3            _|  n+10
              |__________________________|  n+11
              :        OPTIONAL          :
              :        SOFTWARE          :
              -          WORK            -
              :_ _ _ _ SPACE _ _ _ _     :
```

Figure 3-7 Trap Context Format When Using TSA Functionality

3.6.4 TSA Entries Descriptions

Descriptions for the TSA entries are:

1. TSAL = Trap Save Area Link, used to link TSA stack entries to either a pool of TSAs or to the ISA.

2. TRAP # = An eight-bit number that identifies the trap. That is, TRAP # = 40(Hex) - Vector # as shown in Table 3-4.

3. COPY OF I = A copy of [I] at trap time.

4. R3 = A copy of [R3] at trap time.

5. INSTRUCTION 1ST WORD = A copy of the instruction's first word. This entry is undefined whenever the instruction's first word can not be fetched (e.g., if a TV 15 or 17 is detected).

6. Z = Miscellaneous information. Its format is:

```
       0  1        3 4            7 8    9 10  11  12         15
 Z =  |  |          |               |      |    |    |            |
      | A| 0   0   0|      B   I    |  RN  | 0  | NT |    I   S   |
      |__|__________|_______________|______|____|____|____________|
```

where:

a. A defines whether the A field in words 6 and 7 of the Trap Context Format is meaningful or not: for A = 0, the A field is meaningful; for A = 1, the A field is meaningless.

b. BI is a four-bit index field that is meaningful when subword indexing is specified in the AS. BI is meaningless on word and multiword operations and when A = 1. BI can be viewed as being the extension of the A field: together [A] and [BI] form an atom address.

For byte operand addresses, BI = X000, where X is the low-order bit of the byte address followed by three Zeros.

For digit (packed decimal) operand addresses, BI = XX00, where XX are the two low-order bits of the digit address followed by two zeros.

For bit operand addresses, BI = four low-order bits of the bit address.

c. RN is the Saved Ring Number from S.RN of the Trapped procedure.

d. NT is the Nested Trap indicator (M6X or M6XE only).

e. IS is the Instruction Size (up to the instant of trap detection).

7. A = The word Address in the Trap Context. The content of A depends on the trap type:

For TV02 trace traps, [A] points to the instruction following the jump or branch instruction, causing the trap.

For recursive remote descriptor usage, [A] points to the second word of the remote descriptor that attempted to point to another remote descriptor.

For an unauthorized reference to protected memory, reference to an unavailable resource, and memory or bus error traps, [A] contains the address causing the trap.

For CIP and SIP traps, on systems that have a separate processor for CIP and/or SIP, A points to the CIP or SIP instruction.

NOTE

1. The above descriptions define the typical setting of A for the specified trap types. If in certain situations A cannot be defined, then the CSS sets bit 0 of Z to one to indicate that A is meaningless.

2. For the other trap types, the address saved is unspecified and bit 0 of Z is set to one to indicate that A is meaningless.

8. P = A copy of the program counter. The content of P is always a word address and depends on the trap type:

   For TV02 trace traps, [P] points to the next instruction to be executed as a result of the jump or branch.

   For CIP or SIP traps, on systems that have a separate processor for CIP and/or SIP, P points to the next instruction to be executed, which is beyond the CIP or SIP instruction which caused the trap.

   For all other trap types, [P] points somewhere beyond the first word of the instruction that caused the trap. If the instruction is to be retried, subtract from [P] the content of Z.IS.

9. B3 = A copy of [B3] at trap time.

10. OPTIONAL SOFTWARE WORK SPACE = n words that software may allocate in the TSA for its own use.

Table 3-5 RTT and Trap

RTT				
CASE	[IV]	[TSAP]	A C T I O N	C O M M E N T
A	≠NULL	≠NULL	Execute RTT	
B	≠NULL	=NULL	Generate TV16 (Program error)	
C	=NULL	X	P <-- Address of IV	CSS posts a halt or super halt

TRAP					
CASE	[NATSAP]	[TV]	[IV]	A C T I O N	C O M M E N T
D	=NULL	X	X	P <-- NATSAP	Same comment as C
E	≠NULL	=NULL	X	P <-- Address of TV	Same comment as C
F	≠NULL	≠NULL	=NULL	P <-- Address of IV	Same comment as C
G	≠NULL (Not last trap block)	≠NULL	≠NULL	Execute Normal Trap	
H	≠NULL (Last trap block)	≠NULL	≠NULL	Level 2 interrupt (after trap context is saved)	

```
                 *******
                 * START *
                 *******
                    |
                    |
                    v
         ___________________________
        | ASSIGN THE APPROPRIATE    |
        |___TRAP # TO THE TRAP______|
                    |
                    v
           _______________________
          | PROCESSOR PROCESSING  |
          | THE TRAP DETERMINES   |
          |_____ITS CPU #_________|
                    |
                    v
            ___________________
           ( STORE TRAP FRAME  )
           (______IN TSA_______)
                    |
                    v
         _________________________
        | USING CPU # AND CURRENT |
        | LEVEL #, COMPUTE THE VA |
        |_OF THE IV AND FETCH IT__|
                    |
                    |
                    v
            __________________
           /                  \  YES        ************
          <    WAS  A TRAP     >-------->*  HALT  OR  *
           \___ENCOUNTERED ?__/          * SUPER HALT *
                    | NO                  ************
                    |                          ^
                    v               YES        |
           <_IS [IV] = NULL ?_>--------------->|
                    | NO                       |
                    |                          |
                    v                          |
         _________________________             |
        | DETERMINE THE NATSAP #  |            |
        |____BY READING ISM2______|            |
                    |                          |
                    |                          |
                    v                          |
            __________________                 |
           /                  \  YES           |
          <    WAS  A TRAP     >----------------
           \___ENCOUNTERED ?__/
                    | NO
                    |
                    v
                   o o
                  o( A )o
                   o o
                 (Sheet 2)
```

Figure 3-8 Trap Sequence Flow Chart (Sheet 1 of 4)

```
                  o o
                 o( A )o
                  o o
                   |
                   |
                   v
           __________________________
          | USING CPU # AND NATSAP #, |
          |   COMPUTE THE VA OF THE   |
          |___NATSAP AND FETCH IT_____|
                   |
                   |
                   v
             _________________
            /                 \  YES        ************
           <   WAS  A TRAP     >-------->*  HALT  OR  *
            \__ENCOUNTERED ?__/          * SUPER HALT *
                   | NO                  ************
                   |                           ^
                   v_______________  YES       |
            <_IS [NATSAP] = NULL ?_>---------->|
                   | NO                        |
                   |                           |
                   v______________             |
             | USING NATSAP, STORE |           |
             |  THE TRAP CONTEXT*  |           |
             |___IN THE TSA________|           |
                   |                           |
                   |                           |
                   v                           |
             _________________                 |
            /                 \  YES           |
           <   WAS  A TRAP     >----------------
            \__ENCOUNTERED ?__/
                   | NO
                   |
                   v
                  o o
                 o( B )o
                  o o
                (Sheet 3)
```

*Refer to Figure 3-7

Figure 3-8 Trap Sequence Flow Chart (Sheet 2 of 4)

```
                 o o
                o( B )o
                 o o
                  |
                  v
        __________________________
       | USING CPU # AND TRAP #,  |
       |  COMPUTE THE VA OF THE   |
       |_____TV AND FETCH IT______|
                  |
                  |
                  v
          ___________________
         /                   \   YES          ************
        <    WAS   A TRAP     >---------->*  HALT   OR  *
         \___ENCOUNTERED ?___/               * SUPER HALT *
                  | NO                        ************
                  |                                ^
                  v                  YES           |
           <_IS [TV] = NULL ?_>------------------>|
                  | NO                             |
                  v                                |
         ______________________                    |
        | UNLINK TSA FROM TSA  |                   |
        | POOL AND LINK IT TO  |                   |
        | THE ISA AS FOLLOWS:  |                   |
        |                      |                   |
        | [TSAP]    --> TEMP   |                   |
        | [NATSAP]  --> TSAP   |                   |
        | [TSAL]    --> NATSAP |                   |
        |_[TEMP]____--> TSAL___|                   |
                  |                                |
                  v                                |
          ___________________                      |
         /                   \   YES               |
        <    WAS   A TRAP     >--------------------
         \___ENCOUNTERED ?___/
                  | NO
                  v
         ______________________
        | COMPUTE ADDRESS OF   |
        | THE A FIELD IN THE   |
        |  TSA, [TSAP] + 6     |
        |______________________|
                  |
                  v
                 o o
                o( C )o
                 o o
              (Sheet 4)
```

Figure 3-8 Trap Sequence Flow Chart (Sheet 3 of 4)

```
                              o o
                             o( C )o
                              o o
                               |
                               v
                    ___________________________
                   | ADDRESS OF THE A FIELD    |
                   |   IN THE TSA --> B3       |
                   |___________________________|
                               |
                               v
                        _________________
                       | TV.SN => P.SN   |
                       |_TV.DSP => P.DSP_|
                               |
                               v              NO
                     ______________________
                   < IS TV.DSP = ODD ? >-----
                               | YES             |
                               v                 |
                         ______________          |
                        |              |         |
                        |_11 --> S.RN _|         |
                               |                 |
                               |<-----------------
                               v
                  NO  /    IS TSA USED THE      \  YES
            --------< LAST ONE IN THE TSA POOL >---------
            |         \___([TSAL] = NULL) ?____/          |
            v                                             v
   _____________________                           ____________
  | START TRAP HANDLER  |                         | GENERATE A |
  |   EXECUTION AT      |                         |  LEVEL 2   |
  |___CURRENT LEVEL_____|                         |_INTERRUPT__|
            |                                             |
            |                  *****                      |
            ------------------>* END *<--------------------
                               *****
```

Figure 3-8 Trap Sequence Flow Chart (Sheet 4 of 4)

3.7 INTERNAL TIMERS

A system timer that ticks at 120 Hz (every 8.33 milliseconds) is a standard CSS feature. This timer is used to implement real time clock and watch dog timer functionality. For this purpose, a set of words in memory is defined as follows:

LOCATION	MNEMONIC	DESCRIPTION
000014	RTCI	Real time clock initial value. Format is 16-bit unsigned integer.
000015	RTCC	Real time clock current value. Format is 16-bit unsigned integer.
000016	RTCL	Real time clock interrupt level. Format is:

		` 0                 9   10      15`
		`| 0 0 0 0 0 0 0 0 0 0 |  L V L |`

000017	WDTC	Watchdog timer current value. Format is 16-bit unsigned integer.

3.7.1 REAL TIME CLOCK

When RTC is turned on by the RTCN generic instruction each tick of the internal timer will cause the following to occur:

1. [RTCC] <— [RTCC]-1

2. If [RTCC] = 0, then [RTCC] <— [RTCI] (i.e., initialize) and generate an interrupt to level specified by [RTCL]. If the interrupt is of lower priority, i.e., [RTCL] > [S.CL], it is scheduled for service when the level becomes low enough.

When the RTC is turned off via the RTCF generic instruction, no further decrementation of the counter takes place.

3.7.2 WATCHDOG TIMER (WDT)

When the WDT is turned on by the WDTN generic instruction, each tick of the internal timer will cause the [WDTC] to be decremented by 1. If [WDTC] = 0 and the CSS running level is greater than 1, i.e., [S(10:15)] > 1, then a level 1 interrupt is generated. If [WDTC] = 0 and [S(10:15)] = 0, then the WDT interrupt is scheduled for service when the level is low enough. After WDT runout, the [WDTC] continues to be decremented.

When the WDT is turned off via the WDTF generic instruction, no further decrementation of the [WDTC] takes place.

3.8 CSS INITIALIZE OPERATIONS

The following Initialize operations are supported by the CSS:

- START - Action taken following a power-up when the content of main memory is not valid.

- RESTART - Action taken following a power-up when the content of main memory is valid.

- SCF/SYSTEM INITIALIZE - Action taken following the activation of Dump and Control-I from the SCF keyboard. This operation is the equivalent of a START with no change in the state of the power supplies. All the system units, including the SCF, get initialized.

- SYSTEM INITIALIZE - Action taken following the activation of Master Clear from the SCF keyboard. All the system units, except the SCF, get initialized.

- CSS INITIALIZE - Action taken following the activation of CSS Clear from the SCF keyboard. This initialize operation affects only the selected CSS.

In all cases the above operations are directed by the SCF. Refer to the SCF EPS-1 for a description of these operations.

3.9 BOOTLOAD DEVICES

The bootload devices supported and the boot record formats are given in Table 3-6.

Table 3-6 Bootload Devices and Record Formats

DEVICES	R E C O R D F O R M A T
Diskette	Data portion (128 bytes) of track 0, sector 0 (first sector).
Disk Devices	Data portion (256 bytes) of cylinder 0, track 0, sector 0
Magnetic Tape - Streamer - 9 track PE 1600 cpi - 9 track GCR 6250 cpi	One record or less. The record must be the first after BOT.
L64 Front End Processor Coupler	Refer to the Level 64 FEP Coupler EPS-1.
L66 Front End Processor Coupler	Refer to Level 66 FEP Coupler EPS-1, Document No. 60132445.
Communication Boot Adapter	One record 8K bytes or less.

3.10 CSS DEDICATED MEMORY AREAS

Each configured CSS requires an area of 256 dedicated locations. These are allocated as shown in Figure 3-9.

```
                                   ------------------------
LOCATION SEGMENT 0 WORD # 00000  |  DEDICATED MEMORY  | |
                          000FF  | FOR PROCESSOR ZERO | |- 256 words
                                 |--------------------|--
                          00100  |  DEDICATED MEMORY  |
                          001FF  | FOR PROCESSOR ONE  |
                                 |--------------------|
                          00200  |  DEDICATED MEMORY  |
                          002FF  | FOR PROCESSOR TWO  |
                                 |--------------------|
                          00300  |  DEDICATED MEMORY  |
                          003FF  | FOR PROCESSOR THREE|
                                 ----------------------
```

Figure 3-9 CSS Dedicated Memory Areas

The dedicated memory areas are stored in segment zero and require that their VA = PA. This is necessary to satisfy the SCF which in certain cases needs to access the Display Address location, in dedicated memory, without going through an address translation process.

At Start time, the CSS QLTs must insure that PAs 0 - 128 Kbytes are error free by reconfiguring main memory if needed (M6X or M6XE only).

Figure 3-10 shows the entries in a dedicated memory area. All entries are processor number relative except for the memory error count. The memory error count is updated by all CSSs but is maintained only in processor zero's dedicated memory area..

```
                          ADDRESS
                                     ________________
                            00      |_  POWER FAIL  _|
                            01      |_   RESTART    _|
                            02      |______AREA______|
                            03      |                |
                                    ~      RHU       ~
                            05      |________________|
                            06      |                |
                                    ~      RSU       ~
                            09      |________________|
                            0A      |_              _|
                            0B      |____NATSAP 3____|
                            0C      |_              _|
                            0D      |____NATSAP 2____|
                            0E      |_              _|
                            0F      |____NATSAP 1____|
                            10      |_              _|
                            11      |____NATSAP 0____|
                            12      |_              _|
                            13      |______FLBP____**|
                            14      |__RTC INITIAL___|
                            15      |__RTC CURRENT___|
                            16      |_RTC INT. LVL. .|
                            17      |__WDT CURRENT___|
                            18      |_              _|
                            19      |______RSU_______|
** M6X and M6XE only        1A      |_   DISPLAY   **|
   else RFU                 1B      |____ADDRESS_____|
                            1C      |_              _|
                            1D      |_     RSU      _|
*  Supported only in        1E      |________________|
   processor zero's         1F      |_MEM. ER. COUNT *|__
   dedicated memory         20      | 0  -------> 15 |  |  LEVEL
   area.                    21      | 16 -------> 31 |  |- ACTIVITY
                            22      | 32 -------> 47 |  |  FLAGS
                            23      |_48 -------> 63_|__|
                            24      |_              _|
                            25      |_____TV #46_____|
                            26      |       .        |
                                    ~       .        ~
                            7D      |_______.________|
                            7E      |                |
                            7F      |______TV #1_____|
                            80      |_              _|
                            81      |______IV #0_____|
                            82      |_              _|
                            83      |______IV #1_____|
                            84      |       .        |
                                    ~       .        ~
                            FD      |_______.________|
                            FE      |_              _|
                            FF      |______IV #63____|
```

Figure 3-10 CSS Dedicated Memory Area Content (Sheet 1 of 2)

where:

NATSAP 0 - 3	= Next Available TSA Pointers (1 - 3)
FLBP	= Firmware Load Buffer Pointer (M6X and M6XE only). This pointer (consisting of a PA) points to a 2K buffer which is used by the software to load firmware, into I/O controllers which have a loadable control store, during a restart operation.
RTC INITIAL	= Real Time Clock Initial Value.
RTC CURRENT	= Real Time Clock Current Value.
RTC INT. LVL.	= Real Time Clock Interrupt Level.
WDT CURRENT	= Watch Dog Timer Current Value.
DISPLAY ADDRESS	= Contains the PA of a memory location whose content is displayed, by the SCF, when so requested by the operator.
MEM. ER. COUNT	= Memory yellow Error Count. This count is updated by the CSS at every TIC of the Real Time Clock if a Megabus yellow condition was detected since the last TIC..

Figure 3-10 CSS Dedicated Memory Area Content (Sheet 2 of 2)

3.11 ADDRESS SYLLABLE

The Address Syllable (AS) field of an instruction is used to compute an operand address. The AS maps are identified as follows:

- o AS Map 1 - The AS map available to the general instruction set (refer to Figure 3-11). The map as shown is as defined in a M6X and M6XE. Note that CR41E and M5XE models support a subset, as indicated.

- o AS Map 2 - The AS map available to the commercial instruction set (refer to Figure 3-12). This map may also be used by non commercial instructions in a M6X and M6XE. The map as shown is as defined in a M6X and M6XE. Note that CR41E and M5XE models support a subset, as indicated.

- o AS Map 3 - The AS map available only to a M6X and M6XE that allows access (with checking) to the active frame of a stack, and provides for indexing operations using a 32-bit index register (refer to Figure 3-13).

The format of any AS is as follows:

```
           9  11 12    15
           ----------------
     AS    | m  |    n    |
           ----------------
```

where m, n = coordinates of AS map.

The AS can take one of the following forms:

- o Register AS (RAS)
- o Immediate Operand (IMO)
- o Memory AS (MAS)
- o Remote Descriptor Specifier.

Figures 3-11 through 3-13 give the representations of AS MAP 1 - 3 respectively. Table 3-7 lists a set of definitions to facilitate these AS descriptions.

Entries in Figures 3-11 to 3-13 are mnemonics for the various address forms available and are described in subsections 3.11.1 through 3.11.4.

\ n \ m \	0	1-7	8	9-B	C	D-F
0	IMA	Bn (Note 1)	@IMA	@B(n-8)		
1	IMA+R1	Bn+R1	@IMA+R1	@B(n-8)+R1		
2	IMA+R2	Bn+R2	@IMA+R2	@B(n-8)+R2		
3	IMA+R3	Bn+R3	@IMA+R3	@B(n-8)+R3		
4	P+D	Bn+D	@[P+D]	@[B(n-8)+D]		
5	RFU (TV05)	REG (Note 2)	RFU (TV05)	B(n-8) + ↓R1	AS23* (Note 3)	B(n-C) + R1↑
6	↓FT or FT↑*	↓Bn	FT+D*	B(n-8) + ↓R2	EII1* (Note 4)	B(n-C) + R2↑
7	IMO (Note 5)	Bn↑	IV+D	B(n-8) + ↓R3	EII23* (Note 6)	B(n-C) + R3↑

*Not supported in CR41E & M5XE (TV05)

NOTES

1. Map 1 entries, except for Bn and REG (SA1, 2, 3), may not be used by intrinsic scientific instructions, else TV16.

2. REG = Rn except as noted in subsection 3.11.1.1.

3. AS23 may be used by general or non-intrinsic scientific instructions and specifies that AS Map 2 or 3 is to be used.

4. EII1 identifies an extended integer instruction and specifies that AS map 1 is to be used.

5. IMO may not be used by the LXA, SAVE or RSTR instructions, the AAS of an IOLD instruction (except CR41E), nor any extended integer instruction whose data type is less than a word. If IMO is used in any of these cases, then a trap, TV16 is posted.

6. EII23 identifies an extended integer instruction and specifies that AS Map 2 or 3 is to be used.

Figure 3-11 AS Map 1 Format

\ n \ m \	0	1-7	8	9-F
0		Bn+D+0 (Note 1)	P+D+0	@[B(n-8)+D]+0
1		Bn+D+R1+0	P+D+R1+0	@[B(n-8)+D]+R1+0
2	REMOTE	Bn+D+R2+0	P+D+R2+0	@[B(n-8)+D]+R2+0
3	DESCRIPTOR	Bn+D+R3+0	P+D+R3+0	@[B(n-8)+D]+R3+0
4		Bn+D+R4+0	@[P+D]+0	@[B(n-8)+D]+R4+0
5		Bn+D+R5+0	AS3 * (Note 2)	@[B(n-8)+D]+R5+0
6		Bn+D+R6+0	P+BD+0 *	@[B(n-8)+D]+R6+0
7		Bn+D+R7+0	IMO (Note 3)	@[B(n-8)+D]+R7+0

*Not supported in CR41E & M5XE (TV05)

NOTES

Map 2 entries are not applicable to non CIP instructions in a CR41E and M5XE.

Map 2 entries may also not be used by M6X and M6XE intrinsic scientific instructions, else TV16.

1. 0 = offset (applicable only to M6X and M6XE subword instructions, see Table 3-7 for further clarification).
2. In a M6X and M6XE this entry is available only to commercial instructions, else TV16.
3. In a M6X and M6XE this entry is available only to commercial instructions, else TV16.

Figure 3-12 AS Map 2 Format

\ n \ m \	0	1-7	8	9-F
0		FT+D+Rn+O (Notes 1 & 2)	@FT+O↑	
1		Bn+D+K1+O		
2	REMOTE	Bn+D+K2+O	RFU (TV05)	
3	DESCRIPTOR	Bn+D+K3+O		RFU
4		@[FT+D]+Rn+O	@[FT+D]+O	(TV05)
5		= Kn (Notes 2 & 3)		
6		RFU (TV05)	RFU (TV05)	
7				

NOTES

Map 3 entries are not applicable to a CR41E and M5XE.

Map 3 entries may also not be used by M6X and M6XE intrinsic scientific instructions, else TV16.

1. O = offset (applicable only to M6X and M6XE subword instructions, see Table 3-7 for further clarification).

2. This AS may not be used by M6X and M6XE commercial instructions, else TV16.

3. = Kn specifies a Kn except as noted in subsection 3.11.1.2.

Figure 3-13 AS Map 3 Format

Table 3-7 Address Syllable Notation For Word Address Forms (Sheet 1 of 2)

NOTATION	DESCRIPTION
D	D indicates a 16-bit <u>signed</u> displacement in words that follows the address syllable, where $-2^{15} \leq D \leq 2^{15}-1$
BD	BD indicates a 32-bit signed displacement in words that follows the address syllable, where $-2^{31} \leq BD \leq 2^{31}-1$.
@	Indirect operator
+R	Specifies indexing in atoms where $-2^{15} \leq R \leq +2^{15}-1$
+K	Specifies indexing in atoms where $-2^{31} \leq K \leq 2^{31}-1$
FB	FT + L
FT	Address of the top element of the current active frame in the stack
L	Length in words of the current active frame in the stack
↑	Auto increment (B↑, R↑, or FT↑ indicates post-incrementation)
↓	Auto decrement (↓B, ↓R, or ↓FT indicates pre-decrementation)
IMA	Immediate Word Address
IA	Intermediate word Address
B	Base Register
K	Double Word Operand Register
R	Word Operand Register
P	Program Counter. For the purpose of P Relative addressing, the following definition is used: Pd - Points to the leftmost word of the displacement. The displacement maybe in line or in a remote descriptor. (At the completion of an instruction, P points to the first word of the succeeding instruction)
()	Logical Binding
[]	Contents of
+	Addition operation
IMO	Immediate Operand
IV	Interrupt Vector (a word address)

Table 3-7 Address Syllable Notation For Word Address Forms (Sheet 2 of 2)

NOTATION	DESCRIPTION
O	Specifies an offset in bits. O is recognized only when executing subword instructions (Figure 3-14): - For bit instructions (LB, LBF, LBT, LBC and LBS), O specifies an offset in bits of $0 \leq O \leq 15$. - For digit instructions (packed decimal operands), O (C1, C2) specifies an offset of either 0, 4, 8 or 12 bits. - For byte instructions LDH, STH, CMH, ORH, ANH, LLH, IOH, IOLD, and commercial string operands (where O = C1), only the high order bit of the offset field is used. Thus, O specifies an offset of either 0 or 8 bits. - For all other instructions, O is ignored.
<--	Is replaced by
EA	Effective Address

In order to assure upward instruction compatibility, the three AS maps are referenced as follows:

o For general instructions:

The AS field of a general instruction can directly reference an entry in AS Map 1, with the exception of the escape entries EII1 and EII23. The instruction can also use entries from AS Map 2 or 3, by specifying entry AS23 (not applicable to CR41E and M5XE models). See Figure 3-14. AS23 adds a word to the instruction that specifies:

- An offset field, if applicable,

- The AS map number (AS Map 2 or 3),and

- The entry in the selected AS map (2 or 3).

```
                  0                 8 9        15
INSTRUCTION       -------------------------------
WORD 1            |                 |    AS     |
                  -------------------------------
```

If AS does not equal escape codes EII1, EI123 or AS23, then the AS is used to select an entry in AS Map 1.

```
                  0      3 4    7   8 9        15
INSTRUCTION       -------------------------------
WORD 1            |                 |   AS23    |
                  |------------------------------
WORD 2            | OFFSET | RFU | MAP# | AS(2, 3) |
                  -------------------------------
```

If AS = AS23, then AS specifies that the following word be interpreted as shown.

where MAP# = 0 --> AS Map 2; MAP# = 1 --> AS Map 3. AS(2,3) - word 2, bits 9-15 may not specify AS3 or IMO from Map 2. If it does, then TV16 will result.

Figure 3-14 AS Interpretation for General Instructions

o For Extended integer instructions (not applicable to CR41E and M5XE models):

The AS field values, EII1 and EII23 identify the opcode as belonging to the extended integer instruction subset of the instruction set. The next word contains the AS to be used for the instruction, and is interpreted as shown in Figure 3-13.

```
                 0  1  3 4          8 9       15
                 ------------------------------
INSTRUCTION      | 1 | K# |  OP CODE | EII1 or |
WORD 1           |   |    |          |  EII23  |
                 |-----------------------------|
WORD 2           | INSTRUCTION | MAP |   AS    |
                 |  SPECIFIC   |  #  | (1,2,3) |
                 ------------------------------
                 0           7   8   9       15
```

The escape AS (EII1, EII23) and the Map number in Word 2, bit 8, specifies which of the three AS maps is to be used.

ESCAPE AS (Word 1, bits 9-15)	MAP# (Word 2, bit 8)
EII1	0 = RFU (TV05)
EII1	1 = AS Map 1
EII23	0 = AS Map 2
EII23	1 = AS Map 3

AS(1, 2, 3 - Word 2, bits 9-15) may not specify AS23, EII1 or EII23 from Map 1 or AS3 or IMO from Map 2. If it does, then TV16 will result.

Figure 3-15 AS Interpretation for Extended Integer Instructions

o For Commercial Instructions:

The AS field of a data descriptor can directly reference an entry in AS MAP 2 (note the CR41E and M5XE exceptions). A data descriptor may also use entries from AS MAP 3 by specifying entry AS3 (M6X and M6XE only). AS3 adds a word to the data descriptor which is interpreted as shown in Figure 3-16. Refer to subsection 7.2 for more specific information.

```
                0   1   2   3   7  8  9   15
               ------------------------------
      WORD 1   | COMMERCIAL INSTRUCTION OPCODE |
               ------------------------------ -
      WORD 2   | C1 | C2 | C3 |  L  | T |  AS  |  :  DATA
               ------------------------------    :- DESCRIPTOR
      WORD 3   |      DISPLACEMENT or IMO      |  :  (EXAMPLE)
               ------------------------------ -
               |                               |  :
                                                  :_ DATA
               |                               |  :  DESCRIPTOR
               ------------------------------ -:
```

If AS does not equal code AS3, then the AS is used to select an entry in AS MAP 2.

```
                0   1   2   3   7  8  9   15
               ------------------------------
      WORD 1   | COMMERCIAL INSTRUCTION OPCODE |
               ------------------------------ -
      WORD 2   |          R F U         |  AS3 |  :
               ------------------------------    :  DATA
      WORD 3   | C1 | C2 | C3 |  L  | T |  AS  |  :- DESCRIPTOR
               ------------------------------    :  (EXAMPLE)
      WORD 4   |         DISPLACEMENT          |  :
               ------------------------------ -:
               |                               |  :
                                                  :_ DATA
               |                               |  :  DESCRIPTOR
               ------------------------------ -:
```

If AS = AS3, then the rest of the word containing AS3 is RFU and the following word is treated as the descriptor. All entries in AS Map 3 may be used except for FT+D+Rn+0 and = Kn which, if used, results in a TV05.

Figure 3-16 AS Interpretation for Commercial Instructions

- For Scientific Instructions

  - Non-intrinsic scientific instructions

    The AS field of a non-intrinsic scientific instruction can directly reference any entry in AS Map 1, with the exception of the escape entries EII1 and EII23. The instruction may also use entries from AS Map 2 or 3, by specifying entry AS23. AS23 adds a word to the instruction which is interpreted as shown in Figure 3-14. The offset field does not apply to scientific instructions and should be considered as RFU. In CR41E and M5XE models, only AS Map 1 is applicable.

  - Intrinsic scientific instructions (M6X and M6XE only)

    The AS field of an intrinsic scientific instruction may only specify the entries Bn and REG (SA1, 2, 3) of AS Map 1. AS Map 2 and Map 3 may not be used else TV05 results.

3.11.1 Register AS (RAS)

RAS addresses a register which is the source or destination for the operand. Two entries specify this form:

- AS Map 1 entry REG (m = 5, n = 1-7)
- AS Map 3 entry = Kn (m = 5, n = 1-7).

Interpretation of these entries depends on the instruction type.

3.11.1.1 AS MAP 1 ENTRY REG

The entry REG = Rn except as follows:

- REG is illegal (TV16) for instructions LAB, JMP, LNJ, ENT, SAVE, RSTR, LXA and the ICLD-AAS.

- REG is illegal (TV20) for scientific instructions SCZD and SCZQ.

- REG = Bn for the instructions LDB, STB, SWB, CMB, CMN and extended integer instructions specifying ADDRESS for data type.

- REG = R2/R3, R4/R5, R6/R7, for n = 3, 5, 7 respectively for the double word instructions AID, LDI, SID, SDI and extended integer instructions specifying double word for data type. For n = 1, 2, 4 and 6, results are undefined.

- REG selects SA1, 2, 3 for n = 1, 2, 3 for all scientific instructions.

- REG selects R4, R5, R6 for n = 4, 5, 6; and R6/R7 for n = 7 for non-intrinsic scientific instructions. R4, R5, R6, R6/R7 cannot be selected by the scientific instructions SNGD and SNGQ else post a Trap TV20.

Information transferred between an SA and an integer register (R) is converted as appropriate from floating to fixed form or vice versa.

3.11.1.2 AS MAP 3 ENTRY = Kn

The AS selects a K register. It is illegal (TV16) in the following cases:

- o The instructions LAB, JMP, LNJ, ENT, SAVE, RSTR, LXA and all I/O instructions;
- o Any instruction whose operand is less than a word;
- o Any extended integer instruction whose data type is other than double-word or address; and
- o Commercial instructions.

It is illegal (TV20) for the scientific instructions SCZD, SNGD, SCZQ and SNGQ.

Information transferred between an SA and an integer register (K) is converted as appropriate from floating to fixed form or vice versa.

3.11.2 Immediate Operand

The IMO form specifies an immediate operand, of the appropriate size, which follows the address syllable. AS Map 1 and 2 may specify IMO (Map 2 only for commercial instructions; Map 1 only for general, scientific, and EII instructions, else post a TV16).

In general instructions, the size of the IMO is determined by the instruction opcode. In Extended Integer and scientific instructions, the size of the IMO is determined by the data type field. In commercial instructions, the size of the IMO is always one word.

If IMO is interpreted as an address, then it uses a word address form.

The IMO form may not be used by the LXA and LSO instructions or any EII instruction using a data type with atom size less than a word else a TV16 is posted.

The IMO form may also not be used by the SST and SSW scientific instructions else a TV20 is posted.

The IMO form when used in a store operation will cause alteration of procedure.

3.11.3 Memory AS (MAS)

These forms specify an Effective Address (EA) of a memory location. MAS can have the following forms:

- o P Relative
- o Immediate Address (IMA)
- o B Relative
- o IV Relative.
- o Stack Relative.

3.11.3.1 P RELATIVE MAS

The following AS entries specify P Relative MAS address forms:

o From AS Map 1:

P+D	- EA is formed by adding D to Pd.
@[P+D]	- The EA is contained in the location pointed to by Pd+D.

o From AS Map 2:

P+D+O	- EA is formed by first adding D to Pd, and then concatenating to this word address, the offset O.
P+D+Rm+O	- EA is formed by first adding D to Pd, then concatenating to this word address the offset O, and lastly adding the atom index specified in Rm.
@[P+D]+O	- IA is a pointer read from the location defined by Pd+D. The offset O is then concatenated to IA to obtain the EA.
P+BD+O	- EA is formed by first adding BD to Pd, and then concatenating to this word address the offset O.

3.11.3.2 IMMEDIATE ADDRESS MAS (IMA)

The following AS entries in AS Map 1 specify IMA MAS address forms:

o	IMA	- Immediate address. The EA is contained in the location following the instruction.
o	@IMA	- @ is the indirection operator. The EA is contained in the location pointed to by IMA.
o	IMA+Rm	- The EA is IMA indexed by the scaled contents of Rm.
o	@IMA+Rm	- The EA is obtained by adding the scaled contents of Rm to the contents of the location pointed to by IMA (indirect post indexing).

3.11.3.3 B RELATIVE MAS

The following AS entries specify B Relative MAS address forms:

o From AS Map 1:

Bn	- The EA is contained in register Bn.
@B(n-8)	- The EA is contained in the memory location pointed to by B(n-8).
Bn+Rm	- The EA is obtained by adding the scaled contents of the index register Rm to the contents of Bn.

@B(n-8)+Rm	- The EA is obtained by adding the scaled contents of the index register Rm to the contents of the location pointed to by B(n-8).
Bn+D	- The EA is formed by adding D to the contents of Bn.
@[B(n-8)+D]	- The EA is contained in the location pointed to by B(n-8)+D.
↓Bn	- The EA is contained in Bn <u>after</u> the contents of Bn is decremented by One*.
Bn↑	- The EA is <u>contained in Bn</u>. The contents of Bn is incremented by One*. The incrementation takes place after EA formation and prior to execution of the opcode.
B(n-C)+R(m-4)↑	- The EA is obtained by adding the contents of B(n-C) with the scaled contents of index register R(m-4). After EA formation and prior to execution of the opcode, the index register is incremented by One.
B(n-8)+↓R(m-4)	- The contents of the index register R(m-4) is decremented by One and then scaled and added to the contents of B(n-8) to form the EA.

o From AS Map 2:

Bn+D+O	- EA is formed by first adding D to Bn and then concatenating to this word address the offset O.
@[B(n-8)+D]+O	- IA is a pointer read from the location defined by B(n-8)+D. The offset O is then concatenated to IA to obtain the EA.
Bn+D+Rm+O	- EA is formed by first adding D to Bn, then concatenating to this word address the offset O, and lastly adding the atom index specified by Rm.
@[B(n-8)+D]+Rm+O	- IA is a pointer read from the location defined by B(n-8)+D. The offset O is then concatenated to IA, and lastly the atom index specified by Rm is added to obtain the EA.

o From AS Map 3:

Bn+D+Km+O	- EA is formed by adding D to Bn, then concatenating to this word address the offset O, and lastly adding the atom index specified by register Km.

*Or more, depending on operand size.

3.11.3.4 IV RELATIVE MAS

The following entry in AS Map 1 specifies the IV relative MAS address form:

IV+D - The IA is the content of the location defined by the Interrupt Vector for the current level. D is added to IA to obtain the EA.

3.11.3.5 STACK RELATIVE MAS (M6X and M6XE only)

The following entries specify stack relative MAS address forms. For checking conditions, see subsection 3.4.

o From AS Map 1:

↓FT↑: The ↓FT↑ AS is opcode dependent. As a function of the opcode used, the active frame will either be pushed (↓), popped(↑), or a Trap 16 (program error) will result. The criteria for determining what operation applies are as follows:

a. All store instructions imply PUSH.

b. All read-write instructions and the following instructions result in a Trap 16:

- LINK JUMP (LNJ)
- JUMP (JMP)
- ENTER (ENT)
- LAB
- LXA
- IO (DAS)

c. All other instructions imply POP.

See Table 3-8 for the effect of the ↓FT↑ AS on all the the pertinent general, EII and scientific instructions.

A PUSH (↓) operation implies that an operand is to be be stored in the active frame. Consequently, the active frame must be enlarged accordingly. Following the enlargement of the frame, the new "acquired" space is used to store the operand.

The frame enlargement is in words. Consequently, if a byte is to be stored, then the frame is enlarged by one word, and the data is left-justified within the word. If an address is to be stored, then the frame is enlarged by two words.

A POP (↑) opperation implies that the operand pointed to by FT is to be used as defined by the instruction and then subsequently removed from the frame.

The frame size reduction is in words. Consequently, if the operand is a bit or a byte, the frame will be reduced in size by one word.

FT+D: This AS is used to access data within the active frame. See Figure 3-2.

- o If D $\geq$ 0 (positive), then FT+D applies. In Figure 3-2, FT points to C5.
- o If D < 0 (negative), then FB+D applies where FB = FT+L. In Figure 3-2, L = L(C) and FB points to L(B).

<u>Examples from Figure 3-2:</u>

```
For D =  0:  EA --> C5

For D =  5:  EA --> C0

For D = -1:  EA --> C0

For D = -4:  EA --> C3
```

The referenced operand must start and end within the active frame, else Trap 16 (program error).

Note that if the active frame $\geq$ 32K, D is incapable of reaching from one end of the frame to the other.

o From AS Map 3:

@FT+0↑:	IA is a pointer read from the location defined by FT. The offset 0 is concatenated to IA to form the EA. IA is then removed from the frame.
FT+D+Rn+0:	This AS is used to access data within the active frame. The word address IA is formed following the rules defined for the FT+D AS. The offset 0 is then concatenated to IA and lastly the atom index specified by Rn is added to obtain the EA.
@[FT+D]+0:	IA is a pointer read from the location defined by FT+D. (The same rules defined for the FT+D AS apply.) The offset 0 is concatenated to IA to form the EA.
@[FT+D]+Rn+0:	IA is a pointer read from the location defined by FT+D. (The same rules as defined for the FT+D AS apply.) The offset 0 is then concatenated to IA and lastly the atom index specified by Rn is added to obtain the EA.

Table 3-8 (↓)FT(↑) AS Functions (Sheet 1 of 6)

CPU INSTRUCTIONS

(↓)FT(↑)	MNEMONIC	INSTRUCTION DEFINITION
		WORD OPERAND INSTRUCTIONS
↑	LDR	Load Register R
↓	STR	Store Register R
X	SWR	Swap Register R
↑	CMR	Compare with Register R
↑	CMZ	Compare to Zero
↑	ADD	Add to Register R
↑	SUB	Subtract from register R
↑	MUL	Multiply Register R
↑	DIV	Divide Register R
↑	OR	OR with Register R
↑	XOR	Exclusive OR with Register R
X	SRM	Store R through Mask
↑	AND	AND with Register R
		BYTE INSTRUCTIONS
↑	LDH	Halfword (byte) Load Register R
X	STH	Halfword (byte) Store Register R
↑	CMH	Halfword (byte) Compare Register R
↑	ORH	Halfword (byte) OR w/ Register R
↑	XOH	Halfword (byte) Exclusive OR w/ Register R

↑ = POP ↓ = PUSH X = Trap 16

Table 3-8 (↓)FT(↑) AS Functions (Sheet 2 of 6)

CPU INSTRUCTIONS (Continued)		
(↓)FT(↑)	MNEMONIC	INSTRUCTION DEFINITION
		BYTE INSTRUCTIONS (Continued)
↑	ANH	Halfword (byte) AND w/ Register R
↑	LLH	Halfword (byte) Load Logical Register R
		MODE REGISTER INSTRUCTIONS
↑	MTM	Modify and/or Test Register M
↓	STM	Store Register M
		ADDRESS INSTRUCTIONS
↑	LDB	Load Register B
↓	STB	Store Register B
↑	CMB	Compare with Register B
↑	CMN	Compare Address to NULL
X	SWB	Swap Register B
X	LAB	Load Effective Address into B
X	LXA	Load Index and Address
		MODIFY INSTRUCTIONS
X	INC	Increment
X	DEC	Decrement
X	CAD	Add Carry

↑ = POP ↓ = PUSH X = Trap 16

Table 3-8 (↓)FT(↑) AS Functions (Sheet 3 of 6)

CPU INSTRUCTIONS (Continued)

(↓)FT(↑)	MNEMONIC	INSTRUCTION DEFINITION
		MODIFY INSTRUCTIONS (Continued)
X	NEG	Negate
X	CPL	Complement
↓	CL	Clear
X	CLH	Clear Halfword (byte)
		CONTROL INSTRUCTIONS
↓	STS	Store S Register
X	JMP	Jump
X	LNJ	Link Jump
X	ENT	Enter
↑	LEV	Level
↓	SAVE	Save Context (FT <-- FT-23)
↑	RSTR	Restore Context (FT <-- FT+23)
		BIT INSTRUCTIONS
↑	LB	Load Bit
X	LBF	Load Bit and Set False
X	LBT	Load Bit and Set True
X	LBC	Load Bit and Complement
X	LBS	Load Bit and Swap

↑ = POP ↓ = PUSH X = Trap 16

Table 3-8 (↓)FT(↑) AS Functions (Sheet 4 of 6)

DOUBLE WORD		
(↓)FT(↑)	MNEMONIC	INSTRUCTION DEFINITION
↑	LDI	Load Double Word Integer
↓	SDI	Store Double word Integer
↑	AID	Add Integer Double
↑	SID	Subtract Integer Double
		INPUT/OUTPUT INSTRUCTIONS
	IOLD	Output Address and Range:
X	AAS	
↑	CAS	
↑	RAS	(If CAS also specifies POP, then POP CAS first, then RAS)
	IOH	Halfword Input/Output
X	DAS	
↑	CAS	
	IO	Word Input/Output
X	DAS	
↑	CAS	
EII SINGLE OPERAND INSTRUCTIONS		
X	KINC	Increment
X	KDEC	Decrement
X	KNEG	Negate
X	KCPL	Complement

↑ = POP ↓ = PUSH X = Trap 16

Table 3-8 (↓)FT(↑) AS Functions (Sheet 5 of 6)

CPU INSTRUCTIONS (Continued)		
(↓)FT(↑)	MNEMONIC	INSTRUCTION DEFINITION
		<u>EII DOUBLE OPERAND INSTRUCTIONS</u>
↑	KLD	Load
↓	KST	Store
↑	KCM	Compare
X	KSW	Swap
↑	KADD	Add
↑	KSUB	Subtract
↑	KMUL	Multiply
↑	KDIV	Divide
↑	KOR	OR
↑	KXOR	Exclusive OR
↑	KAND	AND
		<u>SIP DOUBLE OPERAND INSTRUCTIONS</u>
↑	SLD	Scientific Load
↓	SST	Scientific Store
X	SSW	Scientific Swap
↑	SCM	Scientific Compare
↑	SAD	Scientific Add
↑	SSB	Scientific Subtract

↑ = POP ↓ = PUSH X = Trap 16

Table 3-8 (↓)FT(↑) AS Functions (Sheet 6 of 6)

CPU INSTRUCTIONS (Conrinued)		
(↓)FT(↑)	MNEMONIC	INSTRUCTION DEFINITION
		SIP DOUBLE OPERAND INSTRUCTIONS (Continued)
↑	SML	Scientific Multiply
↑	SDV	Scientific Divide
		SIP SINGLE OPERAND INSTRUCTIONS
↑	SCZD	Scientific Compare to Zero - Two Words
X	SNGD	Scientific Negate - Two Words
↑	SCZQ	Scientific Compare to Zero - Four Words
X	SNGQ	Scientific Negate - Four Words

↑ = POP ↓ = PUSH X = Trap 16

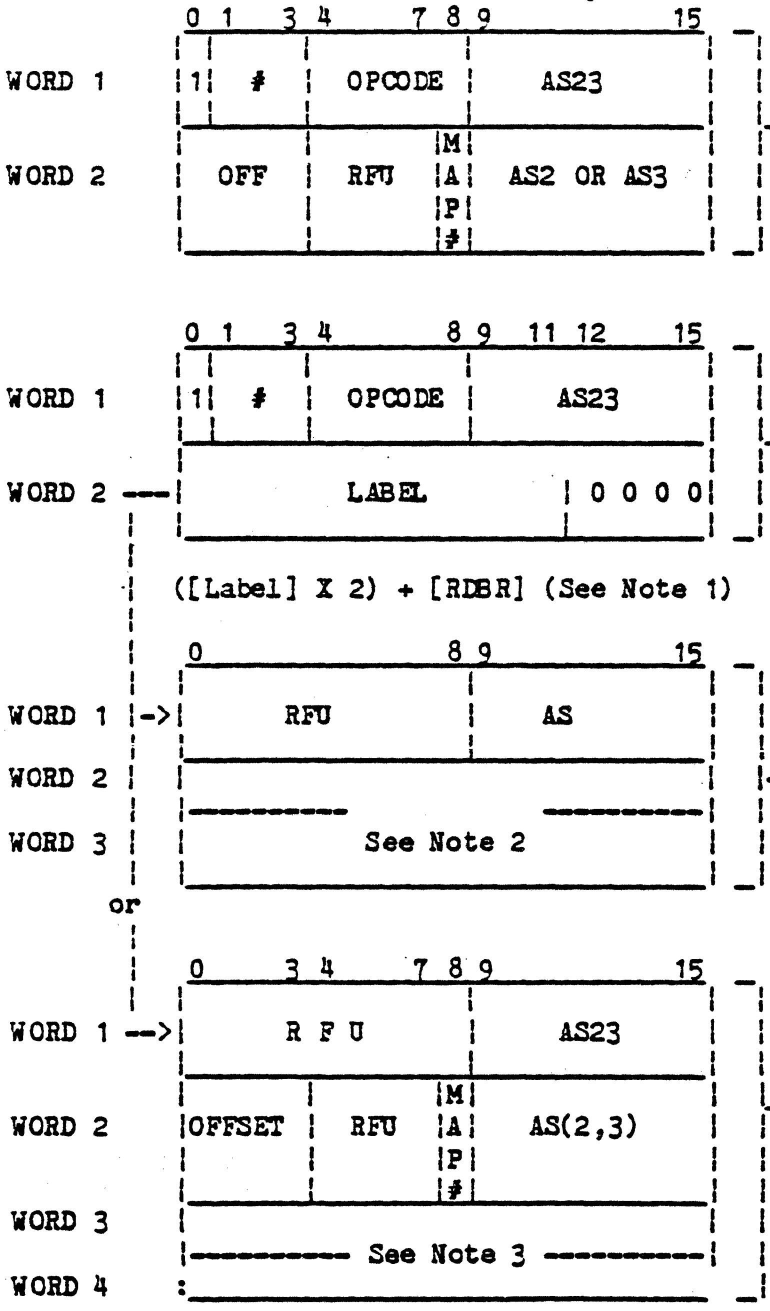
3.11.4 Remote Descriptor Specifier (M6X and M6XE only)

When the four low order bits of the word containing an AS from Map 2 or Map 3 equal Zero, the twelve most significant bits of the word containing the AS are interpreted as a label. The label is multiplied by two (since all remote descriptors are assumed to be two words long) and added to the base value in RDBR. This address defines the location of the remote descriptor which is to be used.

A remote descriptor may not point to another remote descriptor; if it does, a Trap TV12 is posted.

Figure 3-17 shows how the different instructions appears when using an escape AS and when using a remote descriptor.

AS INTERPRETATION FOR GENERAL INSTRUCITONS
(AS Not Equal to EII1 or EII23)



Does not specify a Remote Descriptor (RD).

Specifies a Remote Descriptor in a Remote Descriptor array.

Remote Descriptor using an AS from AS Map 1 (except AS23). Usage of entries EII1 or EII23 will cause a TV16.

Remote Descriptor using an AS from either AS Map 2 or 3.

Figure 3-17 Remote Descriptor Formats (Sheet 1 of 4)

AS INTERPRETATION FOR EXTENDED INTEGER INSTRUCTIONS

```
                0 1   3 4    7 8 9          15
                _____________________________  _
               | |   |          |            |  |
WORD 1         |1| K# |  OPCODE  |EII1 or EII23 |  |
               | |   |          |            |  |_ Does not specify a
               |_____________________________|  |  Remote Descriptor
               |               |M|           |  |  (RD).
WORD 2         |     DATA      |A| AS1 OR AS2 |  |
               |     TYPE      |P|  OR AS3    |  |
               |_______________|#|___________|  _|


                0 1   3 4      8 9  11 12   15
                _____________________________  _
               | |   |          |            |  |
WORD 1         |1| K# |  OPCODE  |   EII23    |  |  Specifies a Remote
               | |   |          |            |  |_ Descriptor in a
               |_____________________________|  |  Remote Descriptor
WORD 2 ---|          LABEL            | 0 0 0 0|  |  array.
          |    |___________________________|___|  _:
          |
          |   ([Label] X 2) + [RDBR] (See Note 1)
          |
          |     0            7 8 9          15
          |     _____________________________  _
          |    |               |M|           |  |
WORD 1    |->  |     DATA      |A|    AS     |  |
          |    |     TYPE      |P|           |  |  Remote Descriptor
          |    |_______________|#|___________|  |_ using an AS from
WORD 2    |    |                             |  |  AS Map 1 (except
          |    |----------         ----------|  |  AS23.
WORD 3    |    |         See Note 2          |  |
          |    |_____________________________|  _|
          |
         or
          |
          |     0     3 4    7 8 9          15
          |     _____________________________  _
WORD 1 -->     |     R F U      |    AS23     |  |
               |________________|____________|  |
               |               |M|           |  |  Remote Descriptor
WORD 2         |     DATA      |A|  AS(2,3)  |  |_ using an AS from
               |     TYPE      |P|           |  |  either AS Map 2
               |_______________|#|___________|  |  or 3.
WORD 3         |                             |  |
               |---------- See Note 3 ----------|  |
WORD 4         :_____________________________  _|
```

Figure 3-17 Remote Descriptor Formats (Sheet 2 of 4)

AS INTERPRETATION FOR COMMERCIAL INSTRUCTIONS

```
             0               10        15     _
            |__________________________________|  |
WORD 1      | 0 0 0 0 0 0 0 0 0 1  X X X X X   |  |
            |__________________________________|  |
            |                                  |  |
WORD 2      |         DATA DESCRIPTOR          |  |  Data Descriptors
WORD 3      |              DD1                 |  |- do not specify
            |__________________________________|  |  a Remote Descriptor.
            |                                  |  |
WORD 4      |         DATA DESCRIPTOR          |  |
WORD 5      |              DD2                 |  |
            |__________________________________| _|

             0               10        15 :   _
            |__________________________________|  |
WORD 1      | 0 0 0 0 0 0 0 0 0 1  X X X X X   |  |
            |__________________________________|  |  Data Descriptor for
            |                                  |  |  the second operand
WORD 2      |         DATA DESCRIPTOR          |  |_ specifies a Remote
WORD 3      |              DD1                 |  |  Descriptor in a
            |__________________________________|  |  Remote Descriptor
            |                       |          |  |  array.
WORD 4 ---- |         LABEL         | 0 0 0 0  |  |
         |  |_______________________|__________| _|
         |
         |  ([LABEL] x 2) + [RDBR] (See Note 1)
         |
         |   0    1    2    3    7 8 9     15     _
         |  |    |    |    |       | |         |  |  Remote Descriptor
WORD 1   |->| C1 | C2 | C3 |   L   |T|   AS    |  |_ using an AS from
         |  |____|____|____|_______|_|_________|  |  AS Map 2 (except
WORD 2   |  |                                  |  |  AS3).
         |  |----------- See Note 2 -----------|  |
WORD 3   |  |__________________________________| _|
         |
        or
         |
         |   0    1    2    3    7 8 9     15     _
         |  |                        |         |  |  Remote Descriptor
WORD 1   -->|         R F U          |   AS3   |  |  using an AS from
            |________________________|_________|  |  AS Map 3.  Usage
            |    |    |    |       | |         |  |- of entries FT + D
WORD 2      | C1 | C2 | C3 |   L   |T|   AS    |  |  + Rn + 0 or =Kn
            |____|____|____|_______|_|_________|  |  will result in a
WORD 3      |                                  |  |  TV16.
            |----------- See Note 3 -----------|  |
WORD 4      |__________________________________| _|
```

Figure 3-17 Remote Descriptor Formats (Sheet 3 of 4)

AS INTERPRETATION FOR NON-INTRINSIC SCIENTIFIC INSTRUCTIONS

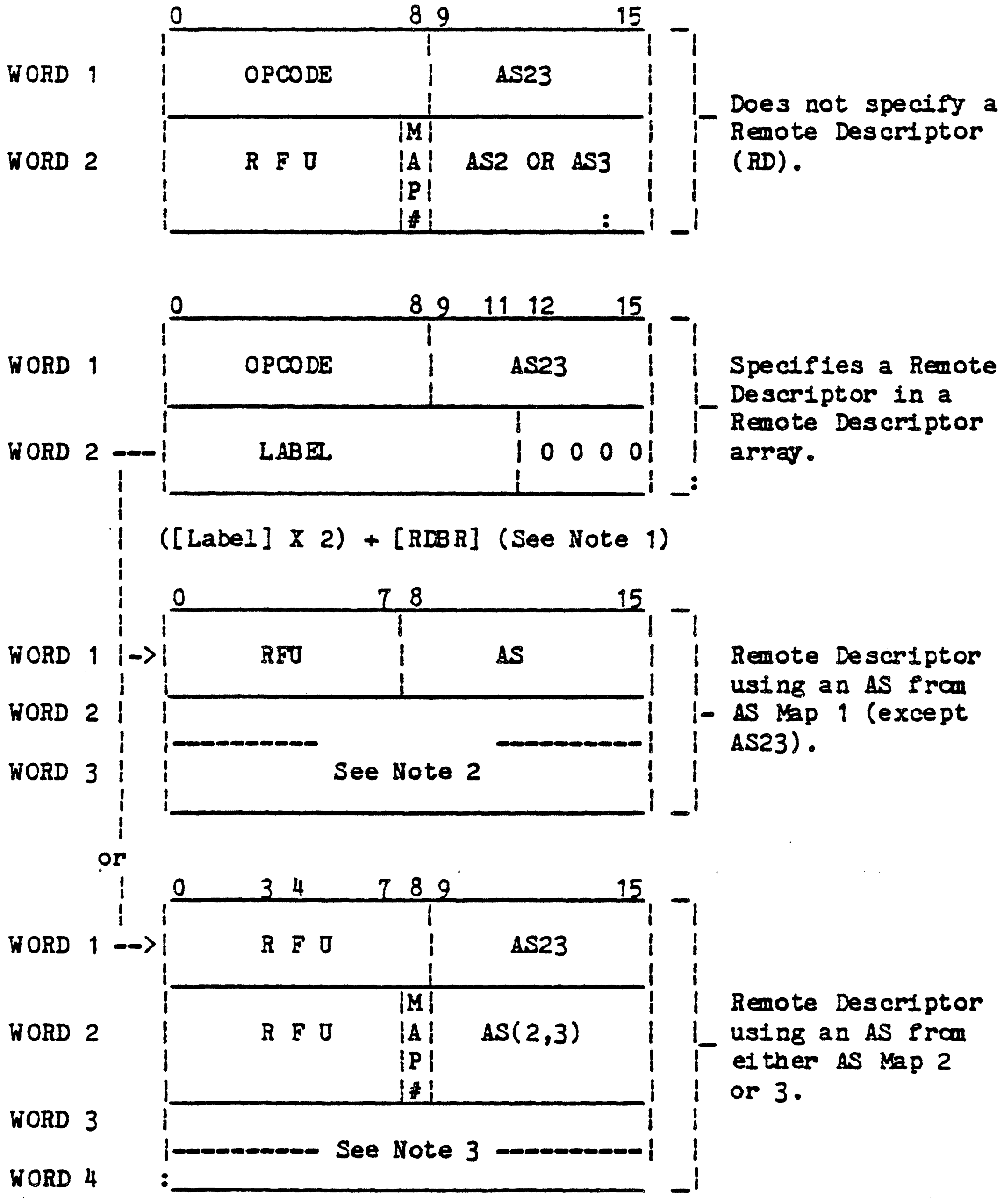

0 8 9 15
WORD 1
OPCODE
AS23
WORD 2
R F U
M A P #
AS2 OR AS3
Does not specify a Remote Descriptor (RD).
0 8 9 11 12 15
WORD 1
OPCODE
AS23
WORD 2
LABEL
0 0 0 0
Specifies a Remote Descriptor in a Remote Descriptor array.
([Label] X 2) + [RDBR] (See Note 1)
0 7 8 15
WORD 1
RFU
AS
WORD 2
WORD 3
See Note 2
Remote Descriptor using an AS from AS Map 1 (except AS23).
or
0 3 4 7 8 9 15
WORD 1
R F U
AS23
WORD 2
R F U
M A P #
AS(2,3)
WORD 3
See Note 3
WORD 4
Remote Descriptor using an AS from either AS Map 2 or 3.

Figure 3-17 Remote Descriptor Formats (Sheet 4 of 4)

Notes for Figure 3-17

1. An RD entry consists of two words, hence the formula ([Label] X 2) + [RDBR]. As a function of the AS used, the RD may use 1, 2, 3, or 4 words. Whenever more than 2 words are required, two entries in the RD array must be reserved.

Notes for Figure 3-17 (continued)

2. RDs using entries from AS Map 1 may consist of:
   - 1 Word (e.g., AS = Bn)
   - 2 Words (e.g., AS = Bn+D)
   - 3 Words (e.g., AS = IMA)

3. RDs using entries from AS Map 2 or 3 may consist of:
   - 2 Words (e.g., AS = @FT+O↑)
   - 3 Words (e.g., AS = Bn+D+O)
   - 4 Words (e.g., AS = P+ $\underline{\triangle}$+O)

   Note that the offset field O is recognized only when executing subword instructions. Refer to Table 3-7.

3.12 INDEXING

Many address syllable forms specify indexing. During EA generation, the value of the index register is algebraically added as the last step, after any indirection, to form an EA.

Two sets of index registers are available:

o Short index registers, R1 through R7. The index value is a signed integer data word with range r: $-2^{15} \leq r \leq +2^{15}-1$.

o Long index registers, K1 through K3 (M6X and M6XE only). The index value is a signed integer data double word with range r: $-2^{31} \leq r \leq 2^{31}-1$.

While indexing, the hardware automatically aligns the index value to correspond to the size of the item (atom) being referenced. The size of the item (i.e., bit, byte, decimal digit, word, double word or quad word) is determined by the opcode type, or by the data type field of the data descriptor (e.g., word, double word or address).

As an example, consider the address syllable B_1+R_2 which specifies indexing. If $[B_1]$ = 100 and $[R_2]$ = $\pm$ n, then as a function of the atom the following applies:

atom = bit

	0															15
99	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
B_1 = 100	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
101	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

atom = digit (4 bits)

```
              0                                       15
             ------------------------------------------
     99      |    -4     |    -3     |   -2    |   -1   |
             |-----------|-----------|---------|--------|
 B1 = 100    |     0     |     1     |    2    |    3   |
             |-----------|-----------|---------|--------|
    101      |     4     |     5     |    6    |    7   |
             ------------------------------------------
```

atom = byte

```
              0                                       15
             ------------------------------------------
     99      |         -2            |       -1         |
             |-----------------------|------------------|
 B1 = 100    |          0            |        1         |
             |-----------------------|------------------|
    101      |          2            |        3         |
             ------------------------------------------
```

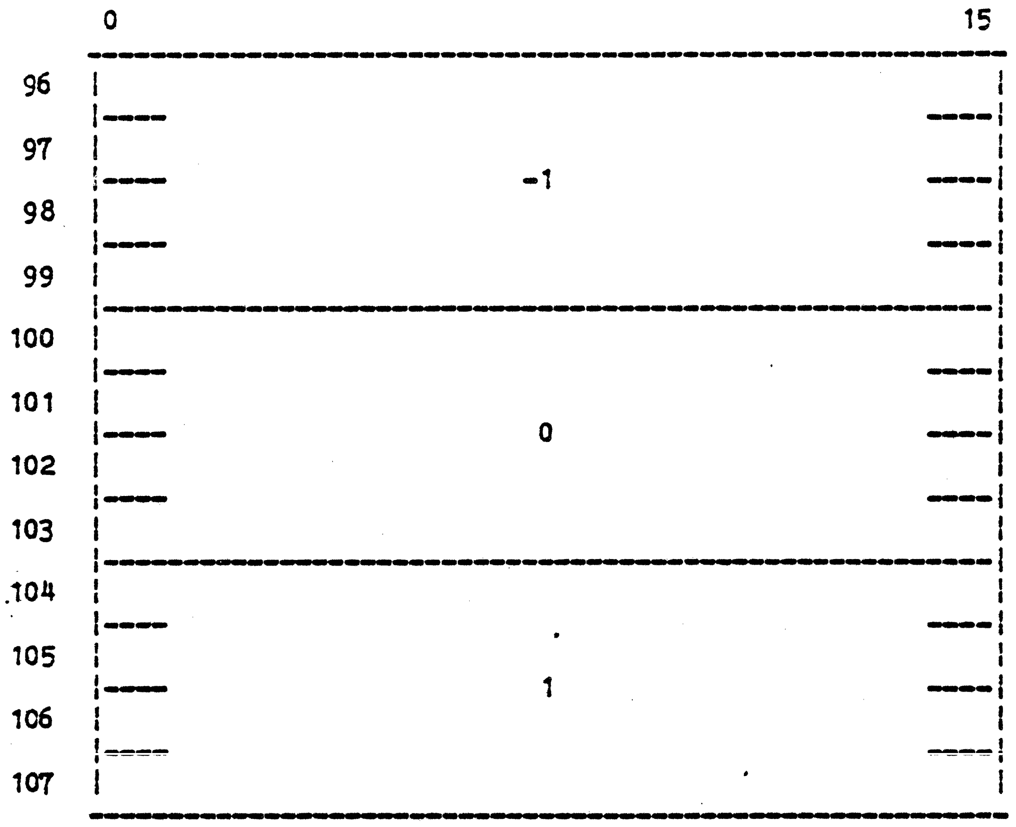
atom = word

```
              0                                       15
             ------------------------------------------
     99      |                   -1                     |
             |------------------------------------------|
 B1 = 100    |                    0                     |
             |------------------------------------------|
    101      |                    1                     |
             ------------------------------------------
```

atom = double word

```
              0                                       15
             ------------------------------------------
     98      |                                          |
             |----               -1                 ----|
     99      |                                          |
             |------------------------------------------|
 B1 = 100    |                                          |
             |----                0                 ----|
    101      |                                          |
             |------------------------------------------|
    101      |                                          |
             |----                1                 ----|
    102      |                                          |
             ------------------------------------------
```

atom = quad word



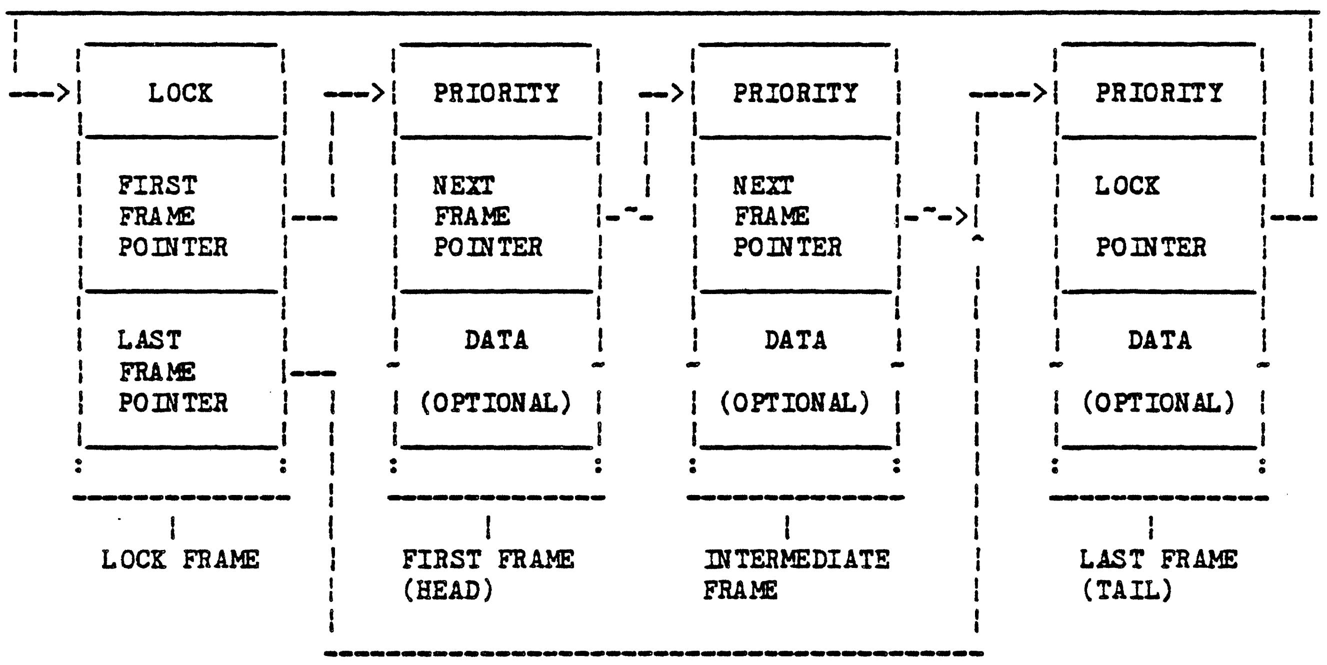
3.13 MEMORY ADDRESS BOUNDS

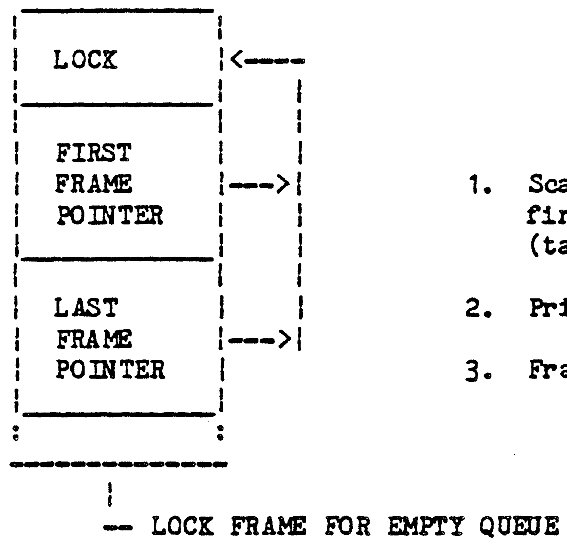
Following conversion of a VA to a PA, the PA is used to address the main memory. Any attempt to address an uninstalled memory address will result in a reference to unavailable resource trap (TV15).

Concerning memory sizing, since the Physical Address Space exceeds the logical addressing limits in all models but the M5XE and M6XE, memory sizing operations must use the segment base relocation capability of those systems, using memory management. Adjusting the value of the segment base field in conjunction with the CMZ with B + X address syllable will cause a TV#15 if $[B+X] \geq M$ or $[B+X] < 0$ (where M = configured memory size). Usage of any other instruction to size memory can result in undefined operation.

3.14 QUEUE MANAGEMENT

The processor provides a queue management capability that facilitates maintenance of ordered lists of "frames". A frame contains a frame priority number, a next frame pointer, and an associated data structure. Each list is identified by a LOCK frame which contains a LOCK word and list head and tail pointers. Reference Figure 3-18.





NOTES

1. Scanning (if any) is always performed from the first frame (head) toward the last frame (tail).
2. Priority is an unsigned 16-bit integer.
3. Frame pointers are two words.

Figure 3-18 Queue Management

The following generic instructions are provided to enqueue/dequeue/search frames in the list:

- o Queue on Head (QOH)
- o Queue on Tail (QOT)
- o Dequeue from Head (DQH)
- o Dequeue by Address (DQA)
- o Search Queue from Head (SQH) (M6X or M6XE only)
- o Search Queue by Address (SQA) (M6X or M6XE only)

For a description of the above instructions, refer to Section 5.

3.14.1 Lock Word

The LOCK word is used to insure that only one procedure is accessing a particular queue at a time. Each instruction causes a fetch of the LOCK word with a Read-Modify-Write (RMW) cycle. If the low order bit of the LOCK = 1 (i.e., list is locked), the RMW cycle is completed without changing LOCK, I(C) is cleared, and the next instruction is fetched. If the low order bit of LOCK = 0 (that is, list is unlocked), the CSS completes the RMW cycle, writing a one into the low order bit of the LOCK word and initiates execution of the queue management instruction.

For the QOH, QOT, DQH and DQA instructions, the LOCK word is cleared (that is, list is unlocked) upon completion of the instruction.

For the SQH and SQA instructions, the LOCK word is left set (that is, list is left locked) upon completion of the instruction. To unlock the list, a non-queue-management instruction must be used. Any instruction, using an RMW cycle, capable of clearing the low order bit of LOCK word to Zero can be used (e.g., LBF).

3.14.2 Scan

The execution of all the queue management instructions causes a scan of the frames in the list, from head toward the tail.

The scan continues until the conditions of the particular instruction are met (a hit), the last frame is reached without a hit, or an interrupt occurs.

When a hit occurs or if the last frame is reached without a hit, the frame is linked into or out of the list as appropriate. I(G) and I(L) indicate the results of the scan.

If an interrupt occurs, the CSS will stop the scan, initiate an RMW cycle, write Zeros in the LOCK word, clear I(C), leave I(G) and I(L) undefined and backup P to point to the queue management instruction, before servicing the interrupt.

3.14.3 Ring Movement

During the scan, the effective Ring number (REF) is moved outward whenever a frame is scanned in a segment having less privilege. If a frame is scanned in a segment requiring greater write privilege than REF then, a trap TV14, access violation, is posted. Following a successful scan (i.e., one where no access violations were encountered) the CSS performs any enqueuing or dequeuing operations without any further access checks.

3.14.4 Lock Frame

Software must build the lock frame of each list to be used. A list with no entries is a lock frame in which the first and last frame pointers point to the LOCK word, see Figure 3-18. The CSS will leave the lock frame in the same condition when a frame is unlinked from a single frame list.

3.14.5 Queue Security

For security reasons it is recommended that the Lock frame as well as all frames of a queue, reside in one or more segments having the same write access rights.

3.15 CSS HALT STATES

The processor supports the following types of HALT states:

- o Procedure Halt
- o Level 63 Halt
- o Operator Halt
- o Super Halt (M6X and M6XE only)
- o Halt (CR41E and M5XE only)

3.15.1 Procedure Halt

This halt state is entered by the CSS when it detects a Halt instruction during procedure execution. In this state, the CSS does not execute any instructions but simply waits for an interrupt. Depression of the Execute key, by the operator, will cause the CSS to skip over the HALT instruction and resume execution at the next instruction. Other operator commands are also honored when in this state.

3.15.2 Level 63 Halt

This halt state is entered by the CSS when it goes to Level 63 and finds the IV = NULL. In this state, the CSS does not execute any instructions but simply waits for an interrupt. Depression of the Execute key, by the operator, will have no effect in this state. Other operator commands however are honored when in this state.

3.15.3 Operator Halt

This halt state is entered by the CSS when so commanded by the operator via the SCF. In this state the CSS does not execute any instructions or honor interrupts. It exits this state only via operator command or initialize.

3.15.4 Super Halt (M6X and M6XE only)

This halt state is entered by the CSS upon detection of an error condition. Upon entering this state, the CSS sets the SH bit in the S register and waits for either operator intervention or an interrupt. Refer to subsection 3.3.4.1 and Table 3-9.

Table 3-9 Super Halt Conditions (M6X and M6XE only)

CONDITIONS CAUSING A SUPER HALT
o LEVEL CHANGE
- TV15 or TV17 is detected when accessing any of the constructs specified in subsection 3.5.2.
- During an INRUSH, the ASV points to an invalid segment.
- During an INRUSH, a TV15 or TV17 is detected when accessing any of the SMMU mode SDs.
o RTT
- [IV] = NULL
- TV15 or TV17 is detected when accessing any of the constructs specified in subsection 3.6.2.
o TRAP
- [NATSAP] = NULL
- [TV] = NULL
- [IV] = NULL
- TV15 or TV17 is detected when accessing any of the constructs specified in subsection 3.6.2.

3.15.5 Halt (CR41E and M5XE only)

This halt state is entered by the CSS upon detection of an error condition. Upon entering this state, the CSS waits for either operator intervention or an interrupt. Refer to Table 3-10.

Table 3-10 Halt Conditions (CR41E and M5XE only)

CONDITIONS CAUSING A HALT
o RTT
- [IV] = NULL
o TRAP
- [NATSAP] = NULL
- [TV] = NULL
- [IV] = NULL

This page is intentionally blank.

SECTION 4 MEMORY MANAGEMENT UNIT

4.1 OVERVIEW OF THE MEMORY MANAGEMENT UNIT

The Memory Management Unit (MMU) supports segmentation with memory relocation and Read-Write-Execute protection based on a ring protection scheme. Depending upon the memory management software used to manage the system, a simple base-and-bounds foreground/background system or a secure, segmented system is provided.

Two Memory Management modes are available to the DPS6 Stage 3 Central Subsystems (CSSs): Standard Memory Management (SMMU) mode and Extended Memory Management (EMMU) mode. The SMMU mode is available on all CSSs while the EMMU mode is offered only on the M5XE and M6XE CSSs as a software selectable option.

An address is treated by the MMU as a logical address consisting of three fields: a Segment Number field, a Block Number field, and an Offset field. The creation of a logical address (LA) takes no cognizance of this field division; hence it is possible, for example, to increment/decrement an LA across Block and Segment boundaries. The MMU enforces protection and relocation whenever the CSS performs memory references.

Two segment sizes are defined: Small Segments (512 bytes $\leq$ size $\leq$ 8192 bytes) and Large Segments (512 bytes $\leq$ size $\leq$ 131,072 bytes).

Note that Small Segments are not supported in EMMU mode.

4.2 MMU FUNCTIONALITY

Subsection 4.2.1 describes the unique functionality of the SMMU and subsection 4.2.2 describes that of the EMMU.

4.2.1 SMMU Functionality

In SMMU mode, a task is provided with a 2^{20} words (2 Megabytes) of Logical Address Space (LAS). 31 Segments (16 small segments and 15 large segments) are used to describe the LAS. The small segments are denoted SD0.0 through SD0.F (hexidecimal) and represent the least significant 128 Kbyte region of the Logical Address Space (LAS). The large segments are denoted SD1.0 through SDF.0 (hexidecimal) and represent the LAS regions above 128 Kbytes through the maximum of 2 Megabytes.

The SMMU contains a storage array consisting of 31 entries. Each entry holds a Segment Descriptor (SD) consisting of 32 bits, which are used to perform address relocation and access checking. The SMMU resides at the CSS's interface with the Level 6 bus, where it mediates all CSS accesses to memory. SDs are loaded into the SMMU array. Figure 4-1 is a block diagram of the SMMU; Figure 4-2 shows how a logical address is interpreted (parsed) and absolutized by the SMMU.

The locations in the SMMU storage array are assigned as follows:

o Locations 0 to 15 are used for segments 0.0 to 0.F
o Locations 16 to 30 are used for segments 1.0 to F.0.

The organization of the SMMU is shown in Figure 4-3.

4.2.2 Extended MMU Functionality

4.2.2.1 EMMU OVERVIEW

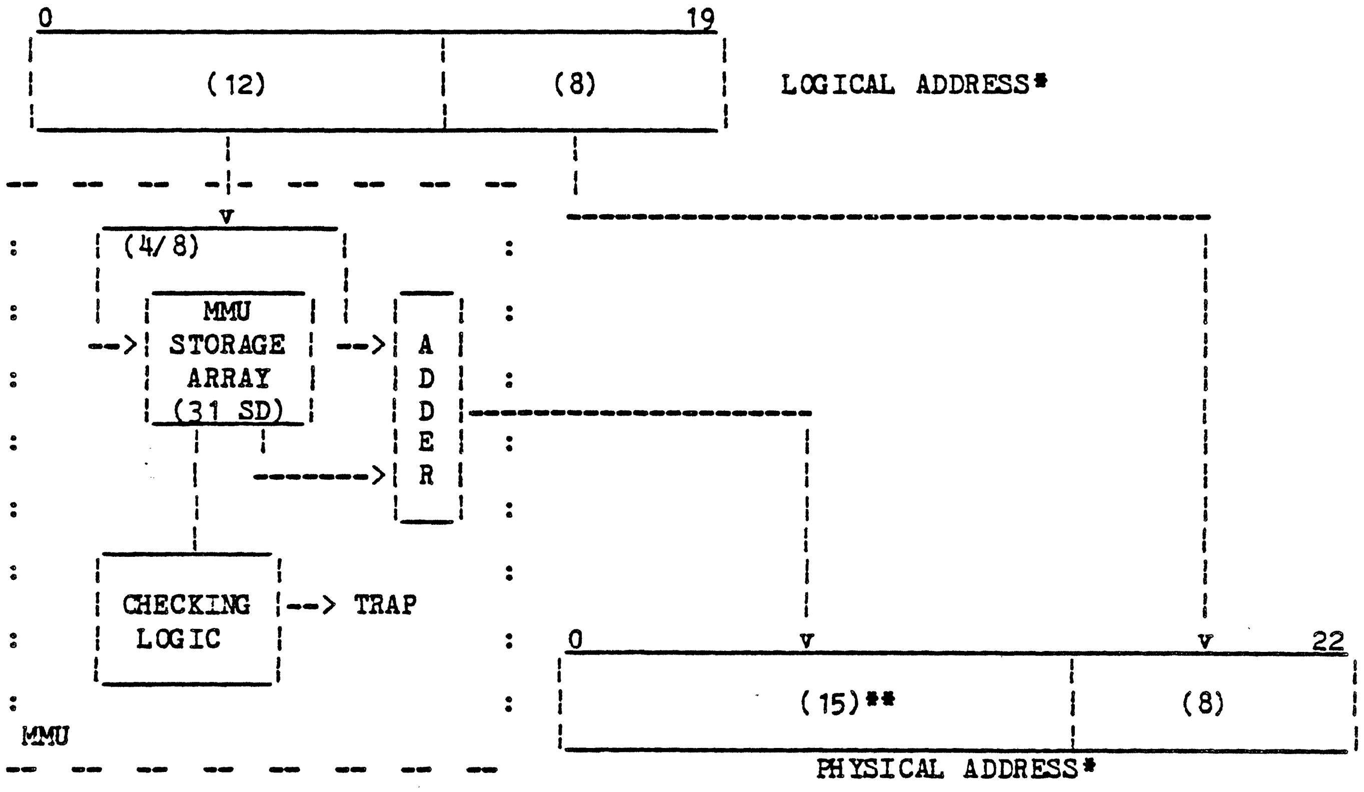
The EMMU features the following extensions:

o The Logical Address Space (LAS) is extended from 2^{20} words (2 Megabytes) to 2^{24} words (32 Megabytes).

o The maximum number of segments is extended to 256 (Large Segments only).

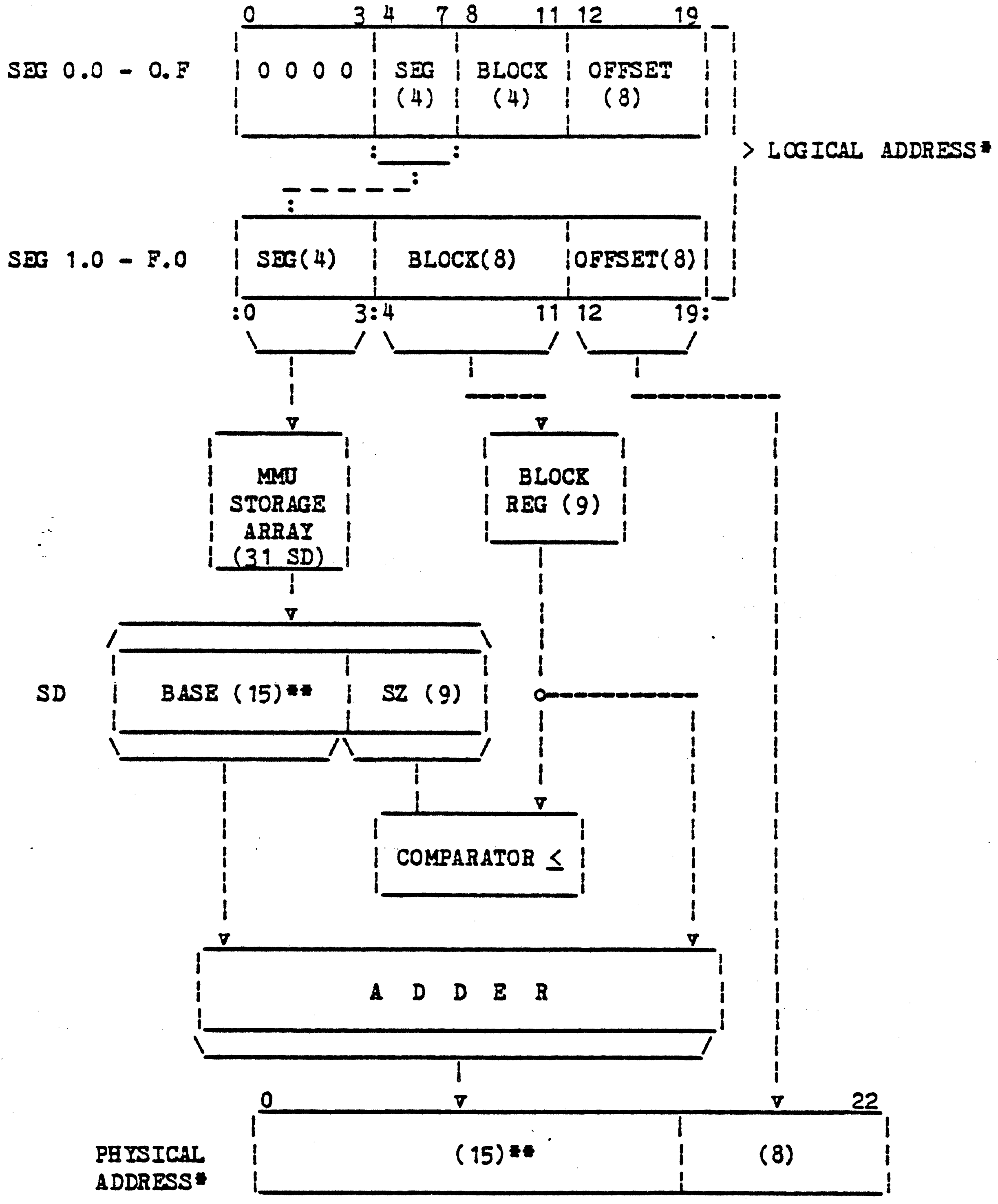
o The Logical Address Space of a Task is divided evenly between the system space and task space (128 segments assigned to each). A System Segment Table (SST) is used to describe the common (to all tasks) segments. A Task Segment table (TST) is used to describe the per-task segments.

o Multiple variable length Segment Tables (STs) located in main memory are supported. One ST for the System Segments (SST) and one ST for each task (TST). At any given time only the SST and one TST are active.



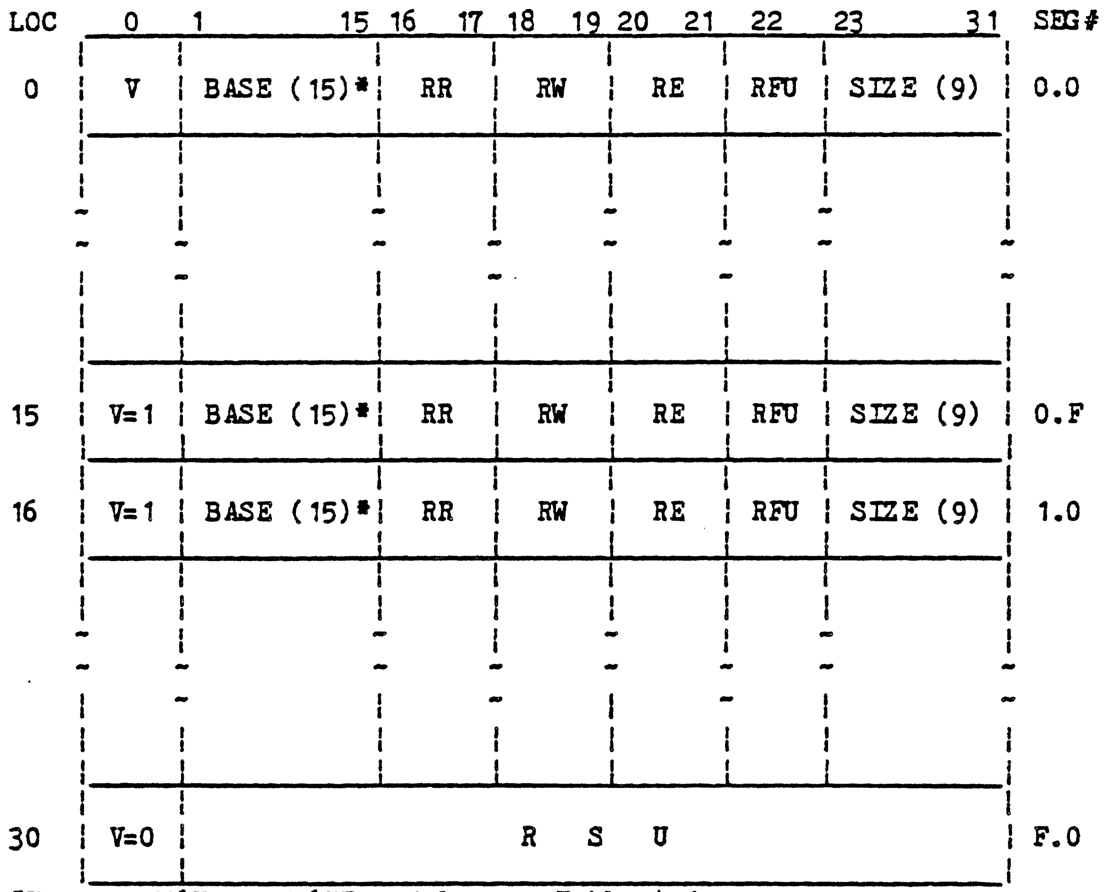
*All Addresses are Word Addresses.
**For non-M6X or -M6XE models see Table 4-1.

Figure 4-1 SMMU Block Diagram



*All Addresses are Word Addresses.
**For non-M6X or -M6XE models see Table 4-1.

Figure 4-2 SMMU Logical Segment Address Interpretation



*For non-M6X or -M6XE models see Table 4-1.

Figure 4-3 M5X MMU Storage Array Layout

Table 4-1 DPS6 Model Specific Variations in Logical/Physical Address Space

DPS6 BASE CSS MODEL	LOGICAL ADDRESS LIMIT (BYTES)	PHYSICAL ADDRESS LIMIT (BYTES)	SD BASE SIZE (BITS)
CR41	2 MB (2^{21})	2 MB (2^{21})	12
CR41E	2 MB (2^{21})	4 MB (2^{22})	13
M5X	2 MB (2^{21})	2 MB (2^{21})	12
M5XE	2 MB (2^{21})/32 MB (2^{25})*	8 MB (2^{23})	14
M6X	2 MB (2^{21})	16 MB (2^{24})	15
M6XE	2 MB (2^{21})/32 MB (2^{25})*	16 MB (2^{24})	15

*EMMU mode selected.

The key differences between the SMMU and EMMU modes are as follows:

o <u>Small Segment Descriptors</u>

In SMMU mode, the lower 128K-byte region of the Logical Address Space (00000 --> 0FFFF Hex) is mapped through the Small SDs (SD0.0 --> SD0.F). In EMMU mode, this region is described by one Large SD (Segment #00).

Small Segment Descriptors are not supported in EMMU mode.

o <u>Logical Address Space</u>

Unlike SMMU mode, which presents one contiguous Logical Address Space (LAS), EMMU mode partitions the LAS, into two separate and distinct regions: System Space and Task Space.

- System Space:

The System Space region is assigned to be the lower 16 MB portion of the 32 MB Logical Address Space. It is described by the System Segment Table (SST). System Space is accessed through SDs 00 through 7F. The SST may be activated (i.e., loaded in to the EMMU) only through execution of either the Activate System Segment Table (ASST) instruction or the Activate Segment Descriptor (ASD) instruction. The ASST instruction causes n system SDs to be copied, from the SST in memory, into the EMMU storage array and the remaining m system SDs (where m = SDs n+1 through 7F) to be loaded as invalid SDs. The ASD instruction allows any one system SD, to be updated in the EMMU storage array.

- Task Space:

The Task Space region is assigned to the remaining (upper) portion of the LAS and is described by the Task Segment Table (TST). Task Space is accessed through Segment Descriptors 80 through FF.

Task Space is assumed to specify 'Task Local' procedure/data. The TST may be activated either through execution of the Activate Task Segment Table (ATST) instruction or at INRUSH time (i.e., during interrupt processing). Consequently, a TST must be attached to each Interrupt Save Area (ISA) which requires a TST change. The Address Space Vector (ASV) in the ISA is used to point to the TST. A Task Segment Table Limit (TSTL) field in the ISA specifies the number of SDs in the TST. At activation time the TST SDs are not 'INRUSH'ed as in SMMU mode. Rather, the pointer to (e.g., ASV) and the size of (e.g., TSTL) the TST are stored in EMMU registers, the old EMMU Task Context is flushed from the EMMU task cache and the new Task SDs are demand loaded, individually, as needed. SDs in the TST must be double word aligned else post a Trap (TV18).

The execution of the ASD instruction is different when activating a task SD. The single task SD is not actually activated by the ASD instruction, but rather it is marked "not Present" in the EMMU task Cache. These task SDs are demand loaded into the EMMU task Cache upon subsequent memory reference.

A view of the EMMU Logical Address Space is given in Figure 4-4 and a description of the EMMU Logical Address interpretation is given in Figure 4-5.

Segment Descriptor formats supported by the EMMU are identical to those used in SMMU mode and are described in subsection 4.2.3.

4.2.3 Segment Descriptor Definition

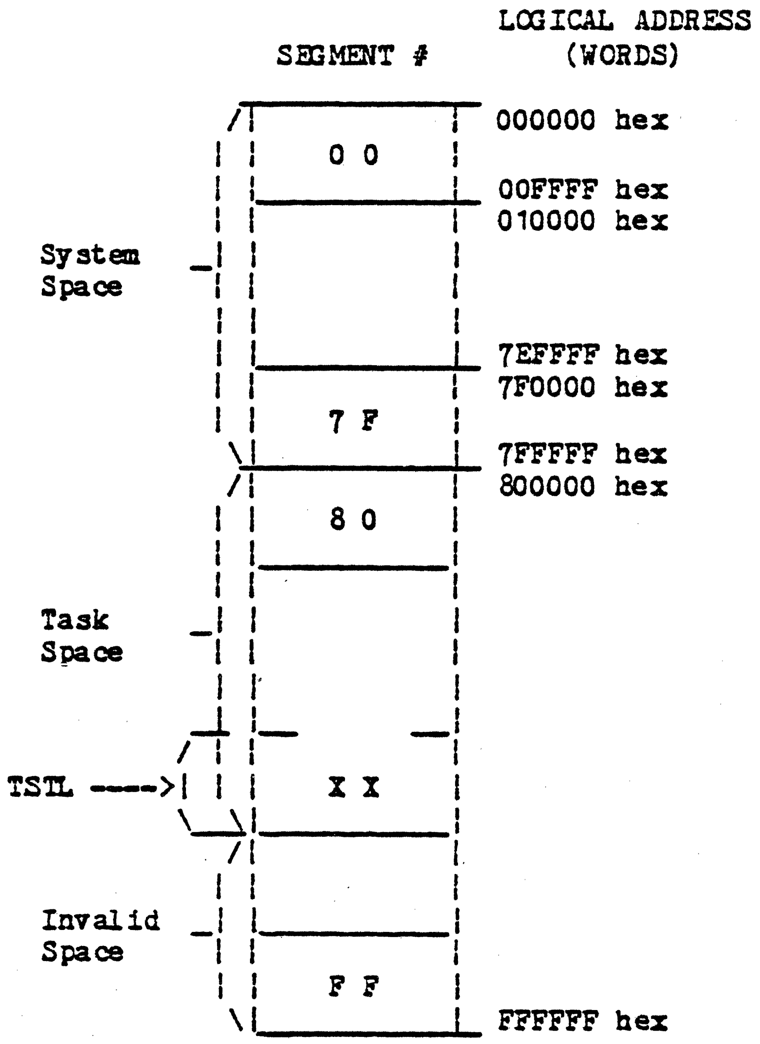
The Valid (V) bit of an SD defines how to interpret the other fields of the SD. If the Valid bit is a Zero, the remainder of the SD is reserved for software use (RSU) and any attempt to access through that SD causes a Trap (TV15). If the Valid bit is One, then the remainder of the SD is interpreted as follows:

The base field defines the modulo 256 word (512 Byte) physical address of the start of the segment.

The RR, RW and RE bits are used to enforce the protection of memory against illegal access. The access rules are described in subsection 4.3.5. If the rules are violated, the access is not performed and a Trap (TV14) is taken.

The size field defines the size of the segment in terms of 256 word (512 Byte) blocks. The size value defines the highest allocated block number. Being nine bits in length, it allows for specification of more blocks (up to 512) than can exist within a single segment (i.e., in SMMU mode, 16 for segments 0.0 - 0.F or 256 for segments 1 - F). This functionality allows any number of segments to be treated as concatenated segments by the MMU if Memory Management software wishes to define them in this way. Inherent in the creation of concatenated segments by the Memory Management software is the requirement that such segments must be logically and physically contiguous. The size field thus indicates a number of blocks that are physically contiguous with respect to this particular segment base. The ability to represent concatenated segments to the MMU is required in connection with checking of instructions that operate on arrays of contiguous data (e.g., commercial and scientific instructions and the VLD instruction). An address space consisting of a single set of concatenated segments is a simple memory partition conventionally described by a base and bounds (size) register pair.

If the size checks are not satisfied, the access or instruction is not performed and a Trap (TV15) is posted. If all checks are satisfied, the Block Register is added to the Base field in the SD and the offset is concatenated to form the physical address. If a carry occurs while performing this addition, a Trap (TV15) is posted. Note that although the Logical address in a M5XE and M6XE is 24 bits, whenever a memory reference is made in SMMU mode, only the low order 20 bits of the logical address are used. The high order four bits of a logical address MBZ in SMMU mode, else Trap (TV15).



where TSTL (Task Segment Table Limit) is defined as the highest numbered segment assigned (addressable) to the Task.

Figure 4-4 EMMU Logical Address Space

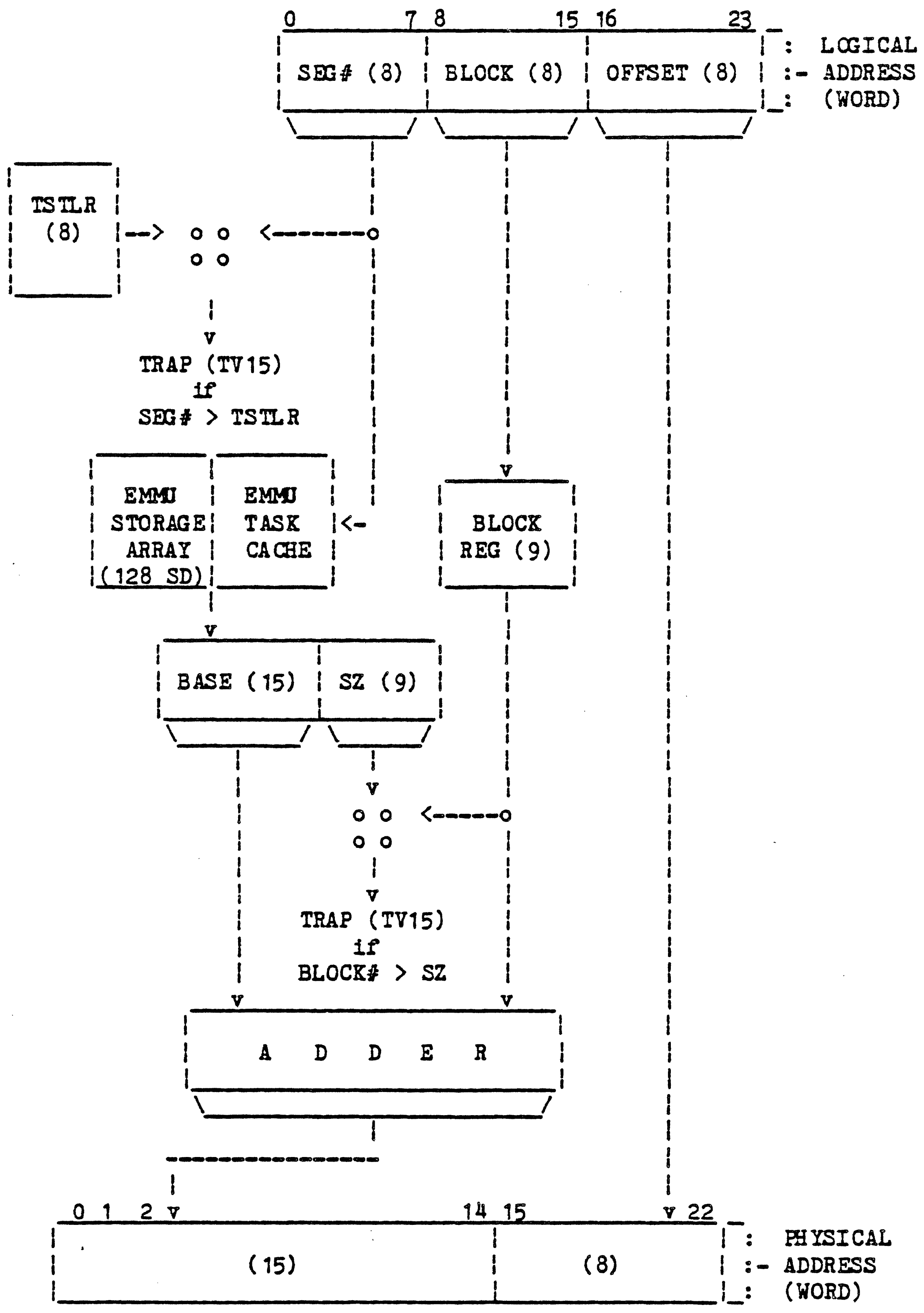


Figure 4-5 EMMU Segment Address Interpretation

4.2.4 MMU Initialization

At initialize time the MMU is always set to a default state. This state is applicable to all the CSS models. The mode selected is SMMU mode. The MMU is loaded with default SDs which define a 2MB Address Space. The default setting of the SDs follows:

- o The Valid bits are set to One,
- o The Size fields are set to all Ones,
- o The RR, RW and RE fields are set to all Zeros.
- o The Base fields are loaded with values that result in a one-to-one mapping between logical and physical memory.

If a Main Memory Prom Option (MMPO) is configured in an M5XE, M6X or M6XE and its base address is set greater than 2MB, then:

- for panel in the unlocked state set SD# F as specified above.
- for panel in the locked state set SD# F to map to the MMPO base address.

MMU functionality, during normal operation, has no overall enabling control; it is always enabled. It starts in the default state and may be subsequently altered by the software. On the M5XE, M6X and M6XE, an MMU diagnostic instruction is provided which allows for example, address relocation (mapping) and access rights checks to be disabled. Refer to subsection 5.9.28 for a description of the MMUD instruction.

The means by which the MMU storage array may be loaded or altered after Initialization, are described in subsection 4.3.4.

4.2.5 MMU Checking

The Block Registers shown in Figures 4-2 and 4-5 are fictitious registers shown here to facilitate the description of the checking performed by the MMU.

The MMU uses the Segment field of the logical address to select an entry in its storage array and uses its contents to perform validity, access right and size checks. The size check verifies that the nine-bit content of the Block Register is equal to or less than the Size field from the MMU entry. When performing this check the Block Register consists of either, the four Block field bits from the logical address with five high order Zeros (for Small Segments 0.0 to 0.F); or the eight Block field bits from the logical address with one high order Zero (for Large Segments).

An additional check is performed in EMMU mode to dynamically verify that the selected SD# is not greater than the Task Segment Table Limit (TSTL), else TRAP (TV15).

On certain commercial, certain scientific and Validate instructions, the CSS uses the MMU to perform a Limit check; the MMU storage array entry (i.e., the SD) used is determined by the segment field of the LA. Before performing the Limit check the CSS calculates a Limit address by taking the block and Offset of the LA, and adds to this a 16-bit Range value derived from the instruction. The high order eight bits of this result, together with the carry, if any, are sent to the Block Register as the nine-bit value to be used to perform the Limit check.

The Limit check is used to verify that the operand(s) of the instruction about to be executed, which could involve a field of n atoms, is within the segment or concatenated segments.

4.2.6 I/O Order Considerations

I/O orders are privileged and thus may be executed only in ring 0 or ring 1. The Validate order must be used to ensure that the I/O buffer is within a segment or two concatenated segments. When the IOLD instruction is executed, the address of the buffer, defined by the AAS, is absolutized by the MMU.

4.3 HARDWARE/SOFTWARE INTERFACES

4.3.1 General

The protection scheme of the MMU is based on a ring structure. Memory objects are visible to a process through SDs. Each SD specifies the access rights applicable to the segment (object). A process executing in ring R has access to any segment in ring R and in all rings of lesser privilege. A process may switch between rings only under tightly controlled conditions. Access rights are enforced through a comparison of an effective ring number and a pertinent access control field in the appropriate SD. This comparison is performed by the MMU hardware.

4.3.1.1 PRIVILEGED INSTRUCTIONS

In order to ensure system integrity the following instructions can be executed only when in:

- o Ring 0: MEMD, MMUD, DHR, CPID, ASST and ATST,
- o Ring 0 or 1: ASD*, LEV, HLT, IO, IOH, IOLD, RSC, RTCN, RTCF, WDTN and WDTF.

Violations of the above conditions result in Trap (TV13).

4.3.1.2 EXECUTION PRIVILEGE OF A PROCESS

The execution privilege R current (RCR) of a process that is running on a processor is defined by a two-bit ring number in the S register. At dispatch time the process execution privilege is saved in its ISA. If a trap is posted while the process is running, then its execution privilege is saved in the Z word (bits 8 and 9) of the TSA.

* ASD may be executed only in ring 0 in a CR41E, M6X or M6XE.

4.3.1.3 ADDRESS SPACE VECTOR

In order to accommodate hardware dispatching of processes with their respective address spaces, the ISA supports an Address Space Vector (ASV). The format of the ASV is shown in Figure 4-6.

The PTR field specifies the base of a segment table (ST) in memory. The ST consists of 62 locations in SMMU mode and from Zero up to 256 locations for the TST in EMMU mode, as determined by TSTL (see subsection 4.3.4). The structure of the ST is shown in Figure 4-7 and the TST in Figure 4-8.

Note that the TST must be memory resident during execution due to demand loading of SDs by the EMMU.

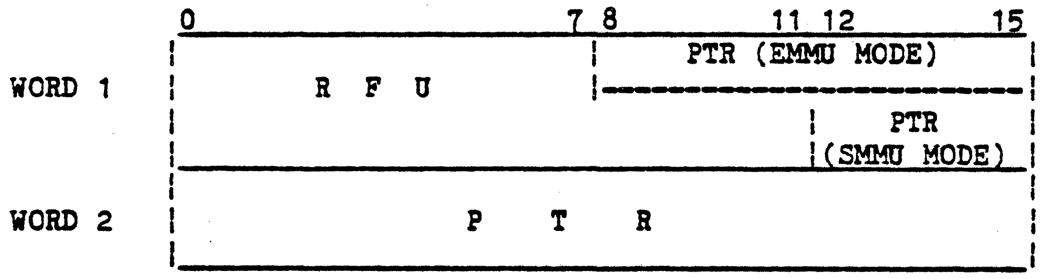
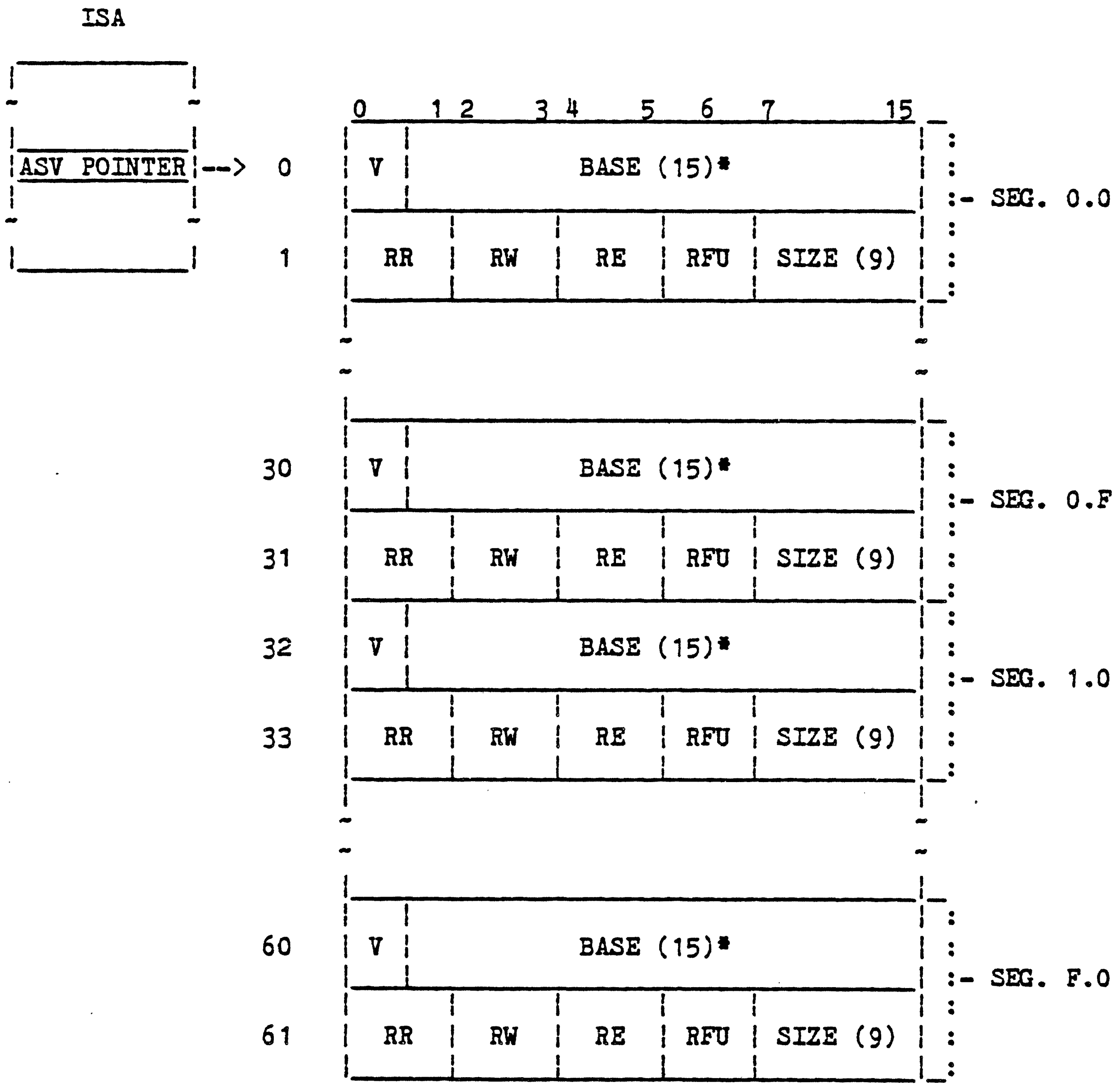


Figure 4-6 ASV Format

SEGMENT TABLE FORMAT FOR SMMU MODE



*For non, M6X or M6XE models (see Table 4-1)

Figure 4-7 Segment Table Format in Memory in SMMU Mode

```
      ISA                      TASK SEGMENT TABLE FORMAT FOR EMMU MODE

 |____________|
 -            -            0    1 2    3 4   5  6  7        15 _
 |            |           |    |                              | :
 |____________|           |    |                              | :
 |ASV POINTER| -->  0     | V  |          BASE (15)           | :
 |   TSTL    |            |____|______________________________| :- SEG. 80
 |___________|            |      |      |      |     |        | :
 |            |     1     |  RR  |  RW  |  RE  | RFU | SIZE (9)| :
 -            -           |______|______|______|_____|________|_:
 |____________|           |                                   |
                          -                                   -
                          -                                   -
                          |___________________________________|_
                          |    |                              | :
                  254*    | V  |          BASE (15)           | :
                          |____|______________________________| :- SEG. FF
                          |      |      |      |     |        | :
                  255*    |  RR  |  RW  |  RE  | RFU | SIZE (9)| :
                          |______|______|______|_____|________|_:
```

* Location of the last segment descriptor word pair is dependent upon the value of the TSTL. Shown here is the case where TSTL = FF (128 SDs assigned). Space requirements for TST are (TSTL - 80)* 2 words.

Figure 4-8 Task Segment Table Format in Memory in EMMU Mode

4.3.1.4 EMMU MODE TASK SEGMENT TABLE LIMIT (TSTL)

The TSTL is contained within a word adjacent to the ASV of the ISA and defines the maximum addressable segment available to the Task. Each memory reference dynamically compares the value of the TSTL with the referenced segment number (80 --> FF) and if SEG# > TSTL, then Trap (TV15).

For a description of the ASV and TSTL fields refer to the description of the ISA in subsection 3.5.2.3. The format of TSTL is shown in Figure 4-9.

```
 0                    7 8                  15
|                      |                    |
|                      |  Highest Segment   |
|         RFU          |  Number Allocated  |
|                      |    To The Task     |
|______________________|____________________|
```

TSTL = 00 --> FF, for values 00 --> 7F, no Task Space is allocated.

Figure 4-9 TSTL Format

4.3.2 Operating System

Operating System (OS) procedures typically are assigned to run in privileged rings. In order to reduce context switching, the OS is not centralized into a separate process, but rather it is distributed (i.e., it is a part of each process). Thus a switch from a user to an OS procedure occurs through a controlled switch in ring of execution.

4.3.3 Segment Descriptors

As shown in Figures 4-10 and 4-11, memory is viewed by a process through its segment descriptor (SD).

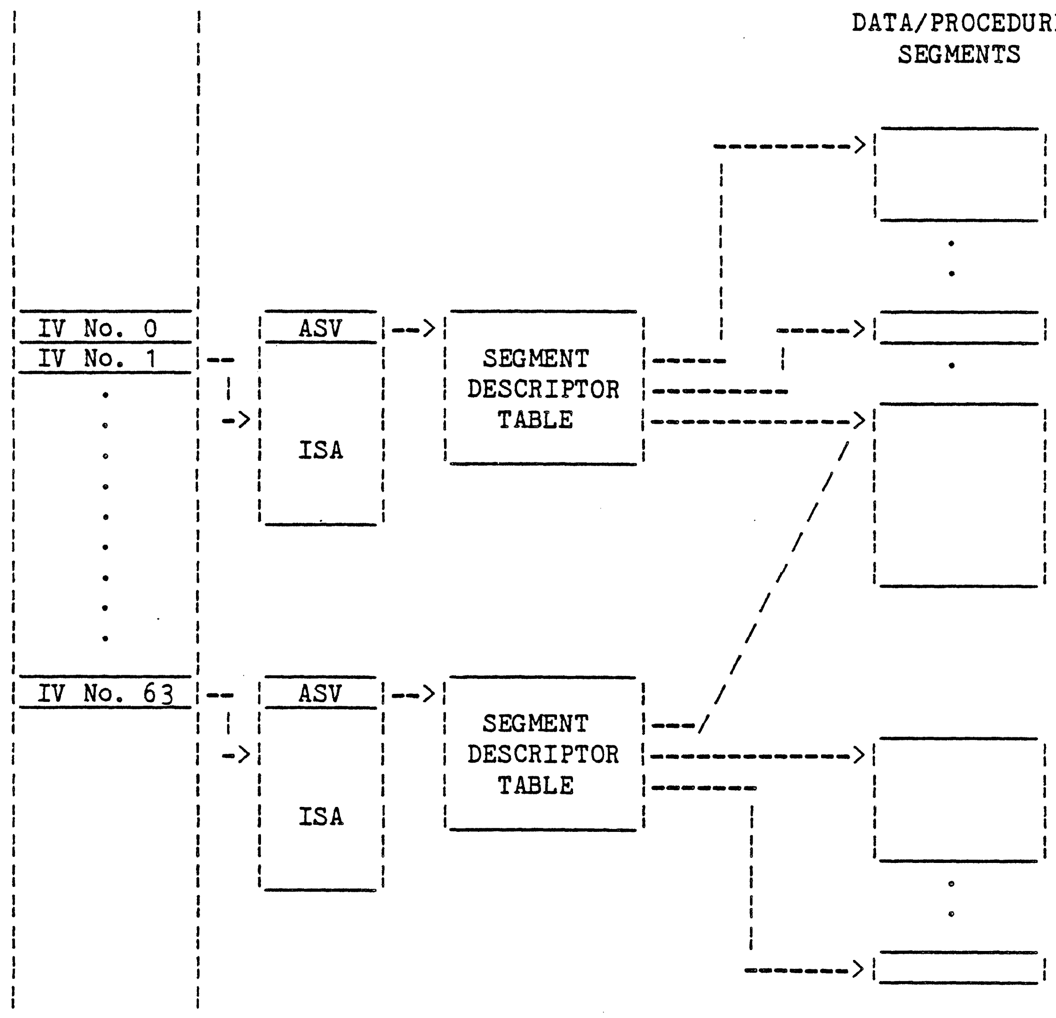
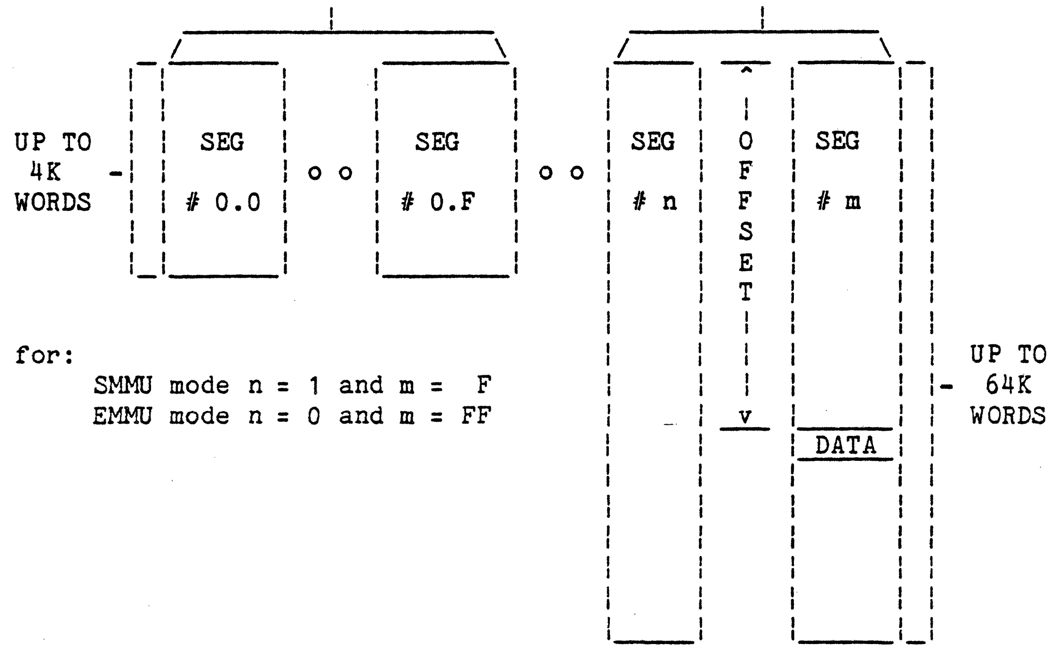


Figure 4-10 Process Context and Address Space Definition

SMMU MODE SUPPORTS 16 SMALL SEGMENTS AND 15 LARGE SEGMENTS
EMMU MODE SUPPORTS 0 SMALL SEGMENTS AND 256 LARGE SEGMENTS



Where for:

SMMU mode n = 1 and m = F
EMMU mode n = 0 and m = FF

Figure 4-11 Process View of Memory

Addresses generated by a process are logical addresses and are mapped by the MMU through the SDs stored in its storage array into physical addresses.

The SMMU is designed to accept a logical word address of 20 bits and maps this into a physical address. Refer to Table 4-1 for the appropriate physical address size as a function of model.

The EMMU is designed to accept a logical word address of 24 bits and maps this into a 23-bit physical address.

4.3.4 Address Space Switching

The OS assigns a logical space to each Task. The logical space of a Task is defined by its ST in SMMU mode or by the combination of the SST and its TST when in EMMU mode. The transition from one address space to another (which takes place at process dispatch time) can be performed in one of two ways:

o INRUSH - This method is applicable at context switch time and uses ISM2 bit 0 (MU bit) to cause an address space change.

  In SMMU mode, the SDs contained in the ST pointed to by the ASV are loaded into the MMU storage array at context load time.

  In EMMU mode, the MMU task cache used to store Task SDs is flushed and the [ASV] and [TSTL] are loaded into MMU registers. All subsequent references to the TST will be made dynamically through on-demand loading of the MMU task cache.

In an M5XE the following applies at inrush time:

The TST is checked to insure that it is in a valid segment. If not then the TSTL is set to 7F (i.e., no task space is allocated). Subsequent references to the TST will result in a Trap (TV15).

The TST is not checked to insure that it is double word aligned. Subsequent references of a segment descriptor from the TST will result in a Trap (TV18).

In an M6XE the following applies at inrush time:

The TST is checked to insure that it is in a valid segment. If not post superhalt for the level.

The SST is not affected by an INRUSH.

o ACTIVATE - This method uses the Activate Segment Descriptor (ASD), Activate System Segment Table (ASST) and Activate Task Segment Table (ATST) instructions to cause an address space change.

In SMMU mode the OS may use ASDs to cause an address space change,.by changing only the required number of SDs in the MMU storage array (e.g., it may change the task SDs but not the system SDs).

In EMMU mode, the OS may use an ASST and/or an ATST to cause an address space change. Typically it will only use an ATST since the system SDs are common to all tasks.

4.3.5 Access Checks

The access permission checks performed by the MMU are as follows:

o A segment descriptor contains three fields defining the access control attributes of the segment as follows:

- RW = Write permission
- RR = Read permission
- RE = Execute permission.

o The current privilege of a process is defined by an R current (RCR) and is contained in S(1:2).

o R effective (REF) is used to determine whether the process may make a memory reference as follows:

- REF = RCR if no indirection is involved, or
- REF = Least privileged of RCR or RW of the segment containing the indirect pointer.

o Whenever fetching an instruction, check that:

RCR $\leq$ RE   i.e., RCR is as privileged or more privileged than the RE of the segment containing the instruction.

- o Whenever reading an operand, check that:

  REF $\leq$ RR

- o Whenever writing an operand, check that:

  REF $\leq$ RW

In all statements of access checks in this specification the tests are stated in terms of ring values rather than in terms of the encoding of the ring values (as given in the next subsection). On this basis, for example, Ring 0 < Ring 1 < Ring 2 < Ring 3 although this is not true of the binary codes which represent these ring values.

Ring 0 always has access rights to any valid segment in the currently defined address space.

4.3.6 Ring Definition

In a system using four ring values the following assignments could be used:

- o Ring 0 - Kernel (Privileged)
- o Ring 1 - Supervisor (Privileged)
- o Ring 2 - Subsystem (Non-privileged)
- o Ring 3 - User (Non-privileged).

In a system using only two ring values, the following assignments could be used:

- o Ring 0 - Supervisor (Privileged)
- o Ring 3 - User (Non-privileged).

Privileges and access rights accorded the various rings are in inverse order to the ring's number (i.e., Ring 0 is most privileged).

There are two kinds of ring values of interest: Current Value (RCR) and Effective Value (REF). REF can be equal to or can differ from RCR (see subsection 4.3.5).

RCR is contained in S(1:2) and may be changed only under certain limited situations (see subsection 4.3.7).

REF is calculated whenever one of the following situations are encountered:

1. Indirect addressing or remote descriptor reference is performed. In this case REF = lesser privileged of REF or RW of segment containing the indirect pointer/remote descriptor.

2. When the firmware references system-base type areas such as interrupt vectors, trap vectors, saving areas, etc. In this case, since the firmware is considered to be part of OS, an Effective Ring Value of Zero (highest privilege) is used.

For compatibility with lower processors of the Level 6 family, the ring values are encoded as one's complements whenever they appear. Thus the ring values and their binary encodings are as shown below:

Ring Values	Binary Code	
0	11	↑
1	10	Increasing
2	01	Privileges
3	00	

4.3.7 Ring Transitions

Figure 4-12 shows the four rings in which a process can be and the transition paths between rings. Ring crossing occurs whenever the RCR of a process running on a processor is changed. The conditions under which this occurs are described below.

Following a system initialize, the CSS places the current process in ring 0. To exit ring 0, i.e., perform an outward ring crossing, the process can either execute an ENTER instruction and thus cause a transition to ring 3, or execute an RTT instruction that results in a transition (through a Trap or emulation of a Trap) to the ring (e.g., 1, 2 or 3) specified in the TSA.

Inward ring crossings are performed via a trap (e.g., Monitor Call instruction). As a function of the Trap vector (i.e., odd or even address), a ring crossing to ring 0 or no ring change occurs. Note that performing an inward ring crossing to a ring other than ring 0 requires that first an inward ring crossing to ring 0 be performed and then an outward ring crossing to the desired ring.

4.3.7.1 TRAP

A Ring crossing can occur as a result of a trap. The trap procedure can run in either the same or the most privileged ring (ring 0).

When a trap occurs, the CSS accesses the next Available TSA Pointer to obtain a TSA, links the TSA to the current ISA and stores appropriate information in the TSA. These operations are accomplished using an REF = Zero, since these are CSS firmware functions.

RCR is saved in bits 8 & 9 of the Z word in the TSA. Still using REF = 0, the CSS then accesses the appropriate Trap Vector. If the least significant bit of the Trap Vector is a Zero, the Current Ring Value in S1, S2 is retained unaltered (i.e., execute the Trap procedure at the current ring value); if it is a One, the Current Ring Value in S1, S2 is set to 11 (i.e., execute the Trap procedure in ring 0).

The address in the Trap Vector is delivered to the P counter and the Trap procedure is executed using the new current ring value.

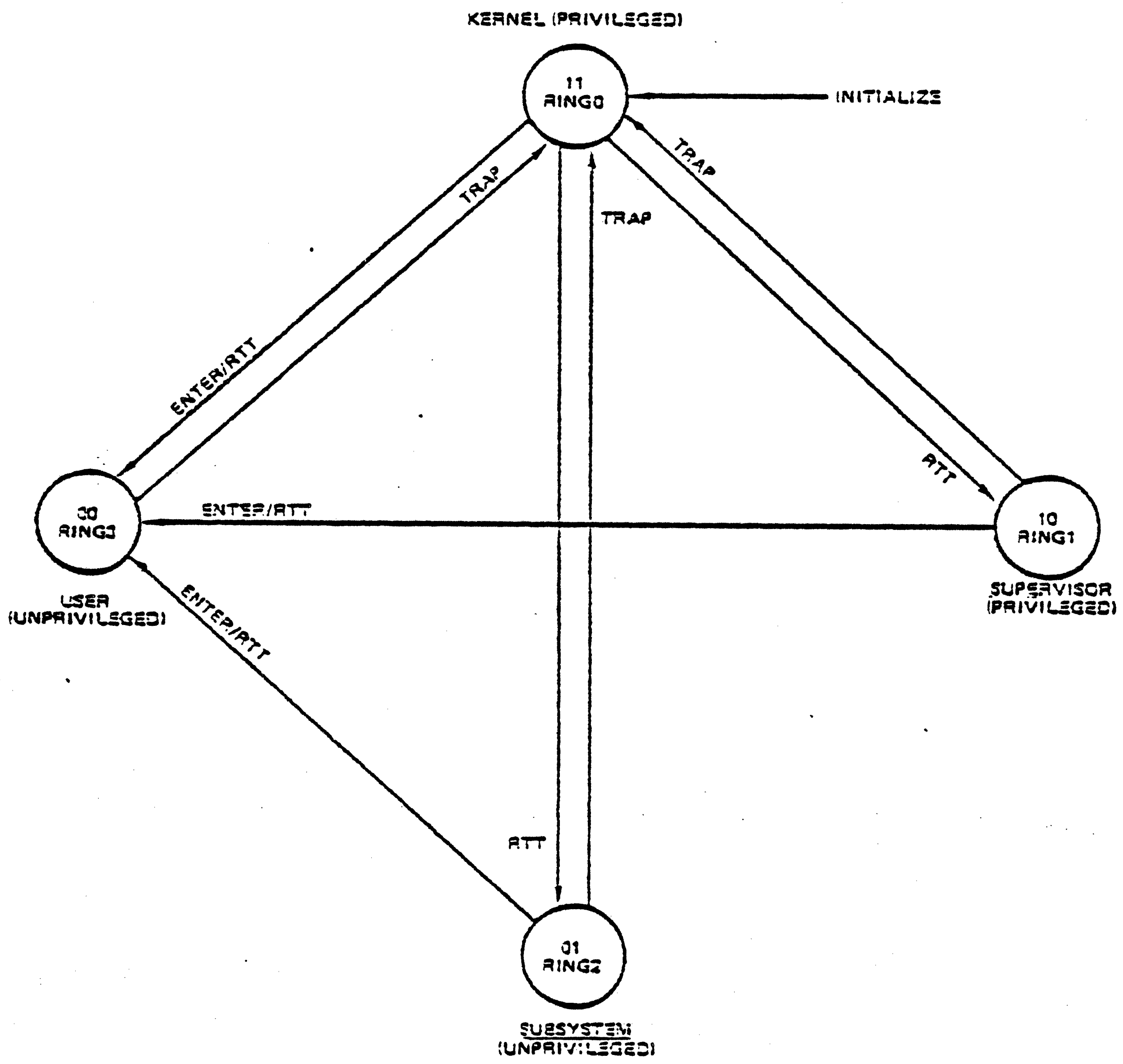


Figure 4-12 State Transitions - Secure System

Non-privileged procedures should not be allowed access to either interrupt or trap vectors or saving areas. Thus, in order for a non-privileged trap procedure to gain read access to the information regarding the trap, it is necessary for a privileged procedure to move this information to a non-privileged area and then pass control to the non- privileged procedure.

4.3.7.2 RTT INSTRUCTION

A Ring crossing may occur as a result of an RTT instruction. The RTT restores the context contained in the TSA, which includes the ring information in bits 8 and 9 of the TSA Z word. Refer to Section 5 for more information about the RTT instruction.

4.3.7.3 ENTER INSTRUCTION

A Ring crossing may occur as a result of an ENTER instruction. The ENTER instruction causes a ring change from any ring to ring 3. Refer to Section 5 for more information about the ENTER instruction.

4.3.8 Transfer of Control Instructions

During the execution of a transfer of control type instruction (e.g., branch, jump, etc.) no ring change takes place regardless of the outcome of the instruction. It is the software's responsibility to insure that privileged procedures not branch to less privileged procedures since this would allow a less privileged procedure to run in a more privileged ring thus jeopardizing system security.

4.3.9 Mode Switching (M5XE and M6XE Only)

Switching the CSS between SMMU and EMMU modes is accomplished with the Activate System Segment Table (ASST) instruction. The ASST instruction is used to activate a new system address space and to activate (or deactivate) EMMU mode as follows:

- o For SST pointer ([B5]) equal to NULL:

  Deactivate EMMU mode (switch to SMMU mode) and generate a trivial map in the MMU (31 SD) Storage Array. All subsequent references are made from the MMU Storage array and the processor is in SMMU mode.

- o For SST pointer ([B5]) not equal NULL:

  INRUSH the SDs contained in the system segment table pointed to by [B5] into the EMMU (128 SD) storage array up to and including the SD whose # = [R6] where 00 $\leq$ [R6] $\leq$ 7F. Note that not all SDs loaded are necessarily Valid/assigned. Mark in the EMMU storage array all SDs beyond the SD specified in R6 as invalid and set the TSTL Register to 7F (127). EMMU mode is activated and all subsequent references are made through the System Space descriptors until a level change specifying an INRUSH is made or an ATST instruction is executed. For a definition of the ASST and ATST instructions, refer to subsection 5.9.

4.3.10 MMU Support Instructions

The following briefly describes the instructions which are pertinent to the MMU. Refer to Section 5 for a detailed description of these instructions.

4.3.10.1 VALIDATE

The Validate instruction provides a means whereby a called procedure running in a more privileged ring can validate the access rights to data of a less privileged caller. This assures that any reference made by the more privileged procedure on behalf of the less privileged procedure is consistent with the less privileged procedure's access rights. The Validate instruction operates identically in both SMMU and EMMU modes.

4.3.10.2 CONVERT LOGICAL ADDRESS TO PHYSICAL ADDRESS (CVP)

The CVP instruction converts a logical address to a physical address and performs a Validate operation. The CVP instruction operates identically in both SMMU and EMMU modes.

4.3.10.3 ACTIVATE SEGMENT DESCRIPTOR (ASD)

The ASD instruction allows a SD to be updated in the MMU storage array. As a function of MMU mode the following applies:

- SMMU Mode - The specific SD (one of 31) is updated in the MMU storage array.
- EMMU Mode - For segment # equals 00 -7F, the specific system SD is updated in the EMMU storage array.
- EMMU Mode - For segment # equals 80 -FF, the specific task SD is flushed from the EMMU task cache.

4.3.10.4 ACTIVATE SYSTEM SEGMENT TABLE (ASST)

The ASST instruction allows entry into or exit from EMMU mode. It also defines the system address space by causing the system segments in the system segment table to be loaded into the EMMU storage array.

4.3.10.5 ACTIVATE TASK SEGMENT TABLE (ATST)

The ATST instruction defines the task address space by causing the EMMU task cache to be flushed and the pointer to the TST and its limit TSTL to be saved in the EMMU. All subsequent references to the task address space will be allowed and will take place via on demand loading of the EMMU storage array.

4.3.11 Segment Descriptor Access Checks (EMMU mode only)

During demand fetching, of a segment descriptor, from the EMMU storage array certain checks are performed by the CSS. Refer to the definition of TV18, in subsection 3.6.3, for a description of these checks.

SECTION 5 GENERAL INSTRUCTION SET DEFINITION

5.1 GENERAL INSTRUCTION SET

This section describes the General instruction set. Table 1-2 lists the various conventions and definitions used to describe these instructions.

The instructions are classified as follows:

- o Double Operand (subsection 5.2)
- o Single Operand (subsection 5.3)
- o Short Value Immediate (subsection 5.4)
- o Branch on Registers (subsection 5.5)
- o Branch on Indicators (subsection 5.6)
- o Shift Operation (subsection 5.7)
- o Input/Output (I/O) (subsection 5.8)
- o Generics (subsection 5.9)

The double and single operand and I/O instructions use an Address Syllable (AS) field. Refer to subsection 3.11 for a description of an AS.

For convenience the instructions are summarized in Appendix E. Appendix E also includes the summaries of those Commercial and Scientific instructions which use the format of general instructions.

5.1.1 Instruction Rules

While reading instruction definitions, the following special cases should be noted:

1. IMO usage: This AS should be used with caution. Improper usage can lead to a modification of the procedure (i.e., IMO AS should never be the destination of an operand).

2. Indexing: Indexing can be used to select bits, bytes, words or double words when using a word address form or bytes only when using a byte address form. Refer to subsection 3.13 for details.

Compare Instructions

Execution of a Compare instruction results in the setting of I(G) or "greater than," I(L) or "less than," and I(U) or "sign unlike" bits of the indicator register. The compare is performed between two operands, namely, OP1 and OP2, where OP1 is contained either in register R or B or [EA] as determined by the opcode. OP2 can reference a memory location or an R- or B-register, or have an implied value of zero, as determined by the opcode. For the purpose of the compare, both quantites (OP1 and OP2) are treated as unsigned binary integers.

- If OP1 > OP2 then I(G) <— 1 else I(G) <—0

- If OP1 < OP2 then I(L) <— 1 else I(L) <—0

- If OP1(0) $\neq$ OP2(0) then I(U) <—1 else I(U) <—0, with the exception of the CMB (Compare with B-register) and CMN (Compare with NULL) instructions during which the setting of I(U) is undefined.

Arithmetic Instructions

Most arithmetic operations (e.g., add, subtract, multiply, arithmetic shifts, etc.) affect the setting of I(OV) or Overflow and I(C) or Carry bit of the indicator register.

I(OV) will set if the Result that is being loaded into a register exceeds the capacity of the register.

This operation is shown for the ADD instructions in truth-table form in Table 5-1.

Table 5-1 Carry and Overflow Truth Table

(Assuming Add Operation)					
BIT 0 OF OPERAND 1	BIT 0 OF OPERAND 2	CARRY-IN FROM BIT 1	BIT 0 OF RESULT	STATE OF I(C)	STATE OF I(OV)
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	0	1	1
1	1	1	1	1	0

During a divide operation, I(OV) is also set to a One under the following conditions:

1. If the divisor = 0, or

2. If $Q > 2^N - 1$, or

3. If $Q < -2^N$ (where N = the size minus one of the register receiving Q), e.g.,

   If the dividend = -2^N and the divisor = -1 (the true value of the quotient would be 2^N, which is not representable and therefore constitutes an OV).

I(C) is set during some arithmetic and shift operations. If an add or subtract operation results in a carry from bit 0, then I(C) is set. This operation is shown for the ADD instructions in truth-table form in Table 5-1.

For shift operations that modify I(C), I(C) reflects the state of the last bit shifted out.

I(C) is set during divide if a non-zero remainder is discarded. If the above operation does not set I(C) and/or I(OV), these indicators are cleared.

5.2 DOUBLE OPERAND INSTRUCTIONS

Double operand instructions have the following format:

```
 0  1  3 4       8 9          15
---------------------------------
| 1 |  # |    OP    |    AS      |
|-------------------------------|
|   Additional words as needed   |
---------------------------------
```

or (M6X and M6XE only)

```
 0  1  3 4       8 9          15
---------------------------------
| 1 |  # |    OP    |    ASN     |
|-------------------------------|
| OFFSET | RFU |MAP#|   AS2,3    |
|-------------------------------|
|0      3 4  6 7  8 9          15|
|                                |
-   Additional words as needed   -
|                                |
---------------------------------
```

where # = Selects one of the general registers. The type of register selected (i.e., B, R, M) is a function of the opcode;

OP = Opcode field;

AS = Address Syllable field.

Within this group, the following types of instructions are available:

- o Address register (B) instructions
- o Word operand register (R) instructions
- o Halfword (byte) instructions
- o Mode register (M) instructions

Depending upon whether the AS specifies RAS, MAS, or IMO form, the double operand instructions are defined to have the following format respectively:

- o RR: register to register
- o RM: register to memory
- o RI: register immediate

These instructions are summarized in Table 5-2. Their numerical representation is given in Table 5-3.

Table 5-2 Double Operand Instructions (Sheet 1 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
WORD OPERAND REGISTER INSTRUCTIONS					
5.2.1	LDR	Load Register R	[R#] <-- [EA]		
5.2.2	STR	Store Register R	[EA] <-- [R#]		
5.2.3	SWR	Swap Register R	[R#] <---> [EA]		
5.2.4	CMR	Compare with Register R	[R#] :: [EA]	G, L, U	
5.2.5	ADD	Add to Register R	[R#] <-- [R#] + [EA]	C, OV	
5.2.6	SUB	Subtract from Register R	[R#] <-- [R#] - [EA]	C, OV	
5.2.7	MUL	Multiply Register R	[R#] <-- [R#] * [EA] except if # = 7, then [R6, R7] <-- [R7] * [EA]	If # < 7, OV	[R6, R7] is double integer format
5.2.8	DIV	Divide Register R	[R#] <-- [R#] / [EA] except if # = 7, then [R7] <-- [R6, R7] / [EA]; [R6] <-- remainder	If # < 7, C, OV If # = 7, OV	[R6, R7] is double integer format
5.2.9	OR	OR with Register R	[R#] <-- [R#] \/ [EA]		
5.2.10	XOR	Exclusive OR with Register R	[R#] <-- [R#] ⊕ [EA]		
5.2.11	SRM	Store R through Mask	For $0 \leq i \leq 15$, if [M(i)] = 1, [EA(i)] <- [R(i)] else [EA(i)] is unchanged		If [M] = 0, use [R1] as Mask. If [R1] = 0, SRM = NOP
5.2.12	AND	AND with Reg. R	[R#] <-- [R#] /\ [EA]		

Table 5-2 Double Operand Instructions (Sheet 2 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
		<u>BYTE INSTRUCTIONS</u> *			
5.2.13	LDH	Halfword (byte) Load Register R	[R#(8:15)] <-- [EA]; [R#(0:7)] <-- [R#(8)]		
5.2.14	STH	Halfword (byte) Store Register R	[EA] <-- [R#(8:15)]		
5.2.15	CMH	Halfword (byte) Compare Reg. R	[R#] :: [EA]	G, L, U	[EA] is sign extended
5.2.16	ORH	Halfword (byte) OR with Register R	[R#] <-- [R#] \/ [EA]		[EA] is sign extended
5.2.17	XOH	Halfword (byte) Exclusive OR with Register R	[R#] <-- [R#] ⊕ [EA]		[EA] is sign extended
5.2.18	ANH	Halfword (byte) AND with Register R	[R#]<-- [R#] /\ [EA]		[EA] is sign extended
5.2.19	LLH	Halfword (byte) Load Logical Register R	[R#(8:15)] <-- [EA]; [R#(0:7)] <-- 0		
		<u>MODE REGISTER INSTRUCTIONS</u>			
5.2.20	MTM	Modify and/or Test Register M	See text	B	
5.2.21	STM	Store Register M	[EA(0:7)] <-- FF; [EA(8:15)] <-- [M#]		

*In all byte instructions [EA] = addressed byte.

Table 5-2 Double Operand Instructions (Sheet 3 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
ADDRESS REGISTER INSTRUCTIONS					
5.2.22	LDB	Load Register B	[B#] <-- [EA]		
5.2.23	STB	Store Register B	[EA] <-- [B#]		
5.2.24	CMB	Compare with Register B	[B#] :: [EA]	G, L, U	
5.2.25	SWB	Swap Register B	[B#] <--> [EA]		
5.2.26	LAB	Load Effective Address into B	[B#] <-- EA		
5.2.27	LNJ	Link Jump	[B#] <-- [P]; [P] <-- EA		[B#] points to the first word after Link Jump. If M1(J) =1 then Trap TV02.

Table 5-3 Numerical Representation of Double Operand Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
5.2.20	8+r	0	0+m	n	MTM	Word
5.2.13	8+r	0	8+m	n	LDH	Byte
5.2.15	8+r	1	8+m	n	CMH	Byte
5.2.6	8+r	2	0+m	n	SUB	Word
5.2.19	8+r	2	8+m	n	LLH	Byte
5.2.8	8+r	3	0+m	n	DIV	Word
5.2.27	8+r	3	8+m	n	LNJ	Word
5.2.9	8+r	4	0+m	n	OR	Word
5.2.16	8+r	4	8+m	n	ORH	Byte
5.2.12	8+r	5	0+m	n	AND	Word
5.2.18	8+r	5	8+m	n	ANH	Byte
5.2.10	8+r	6	0+m	n	XOR	Word
5.2,17	8+r	6	8+m	n	XOH	Byte
5.2.21	8+r	7	0+m	n	STM	Word
5.2.14	8+r	7	8+m	n	STH	Byte
5.2.1	8+r	8	0+m	n	LDR	Word
5.2.4	8+r	9	0+m	n	CMR	Word
5.2.5	8+r	A	0+m	n	ADD	Word
5.2.11	8+r	A	8+m	n	SRM	Word
5.2.7	8+r	B	0+m	n	MUL	Word
5.2.26	8+r	B	8+m	n	LAB	Word
5.2.22	8+r	C	8+m	n	LDB	DWord
5.2.24	8+r	D	8+m	n	CMB	DWord
5.2.3	8+r	E	0+m	n	SWR	Word
5.2.25	8+r	E	8+m	n	SWB	DWord
5.2.2	8+r	F	0+m	n	STR	Word
5.2.23	8+r	F	8+m	n	STB	DWord

where r = Register number contained in bits 1 through 3
and m, n = Coordinates of AS Map (see subsection 3.11).

5.2.1 Load Register R, LDR

Format:

RR, RM, RI

Description:

The contents of the location specified by the AS are loaded into the designated R-register.

Operation:

[R#] <-- [EA]

Indicator Condition:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

5.2.2 Store Register R, STR

Format:

RR, RM, RI

Description:

The contents of the designated R-register are stored in the location specified by the AS.

Operation:

[EA] <-- [R#]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Use of IMO address form may cause alteration of procedure.

5.2.3 Swap Register R, SWR

Format:

RR, RM, RI

Description:

The contents of the designated R-register are swapped with the contents of the location specified by the AS.

Operation:

[R#] <--> [EA]

Indicator Conditions:

```
C  Unchanged
B  Unchanged
I  Unchanged
G  Unchanged
L  Unchanged
U  Unchanged
OV Unchanged
```

Special Conditions:

Use of IMO address form may cause alteration of procedure.

5.2.4 Compare With Register R, CMR

Format:

RR, RM, RI

Description:

The contents of the designated R-register and the contents of the location specified by the AS are compared as unsigned integers. G-, L-, and U-indicator bits are set to indicate the results of the compare.

Operation:

I(G), I(L), I(U) <-- [R#] :: [EA]

Indicator Conditions:

```
                               C  Unchanged
                               B  Unchanged
                               I  Unchanged
If [R#] > [EA] then            G  <-- 1 else G <-- 0
If [R#] < [EA] then            L  <-- 1 else L <-- 0
If [R#(0)] ≠ [EA(0)] then      U  <-- 1 else U <-- 0
                               OV Unchanged
```

Special Conditions:

None.

5.2.5 Add To Register R, ADD

Format:

RR, RM, RI

Description:

The sum of the contents of the designated R-register and the contents of the location specified by the AS is loaded into R.

Operation:

```
[R#] <-- [R#] + [EA]
```

Indicator Conditions:

```
If carry then    C  <-- 1 else C <-- 0
                 B  Unchanged
                 I  Unchanged
                 G  Unchanged
                 L  Unchanged
                 U  Unchanged
If overflow then OV <-- 1 else OV <-- 0
```

Special Conditions:

If an overflow occurs and M1(#) = 1, then a Trap TV06 occurs.

5.2.6 Subtract From Register R, SUB

Format:

RR, RM, RI

Description:

The difference of the contents of the designated R-register and the contents of the location specified by the AS is loaded into R.

Operation:

```
[R#] <-- [R#] - [EA]    (Using two's complement arithmetic)
```

Indicator Conditions:

```
If carry then    C  <-- 1 else C <-- 0
                 B  Unchanged
                 I  Unchanged
                 G  Unchanged
                 L  Unchanged
                 U  Unchanged
If overflow then OV <-- 1 else OV <-- 0
```

Special Conditions:

If an overflow occurs and M1(#) = 1, then a Trap TV06 occurs.

5.2.7 Multiply Register R, MUL

Format:

RR, RM, RI

Description:

The product of the contents of the designated R-register and the contents of the location specified by the AS is loaded into R. If the designated R-register is R7, then the product (double precision format) is loaded into R6 and R7 with R7 being loaded with the least significant portion of the product.

Operation:

```
[R#] <-- [R#] * [EA]
except if # = 7
then [R6, R7] <-- [R7] * [EA]
```

Indicator Conditions:

```
                C  Unchanged
                B  Unchanged
                I  Unchanged
                G  Unchanged
                L  Unchanged
                U  Unchanged
If overflow then OV <-- 1 else OV <-- 0 except for # = 7,
                 then OV is cleared.  All operands remain
                 unchanged if OV <-- 1.
```

Special Conditions:

If an overflow occurs and M1(#) = 1, then a Trap TV06 occurs.

5.2.8 Divide Register R, DIV

Format:

RR, RM, RI

Description:

The designated R-register contents are divided by the contents of the location specifed by the AS, and the resulting quotient is loaded into R. If the designated R-register is R7, then the operation is performed on the double integer operand contained in R6 and R7, and the remainder in single integer format is loaded into R6.

If the contents of the location specified by the AS (divisor) are zero, or if the resulting quotient (Q) is $Q < -2^{15}$ or $Q > 2^{15} -1$, then the overflow condition is set, the contents of the selected registers are unchanged, and the C-indicator bit is in an undefined state.

Operation:

[R#] <-- [R#] / [EA]

except if # = 7, then

[R7] <-- [R6, R7] / [EA];
[R6] <-- Remainder (supplied with the same sign bit as the dividend)

Indicator Conditions:

```
For # ≠ 7, if the remainder ≠ 0, then C  <--1, else C <-- 0
For # = 7                             C  Unchanged
                                      B  Unchanged
                                      I  Unchanged
                                      G  Unchanged
                                      L  Unchanged
                                      U  Unchanged
```

If the divisor is zero, or if the quotient causes overflow, then the operands remain unchanged and OV <--1.

Special Conditions:

If an overflow occurs and M1(#) = 1, then a Trap TV06 occurs.

5.2.9 OR With Register R, OR

Format:

RR, RM, RI

Description:

The inclusive OR of the contents of the designated R-register and the contents of the location specified by the AS is loaded into R.

Operation:

[R#] <-- [R#] \/ [EA]

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged

L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

None.

5.2.10 Exclusive OR With Register R, XOR

Format:

RR, RM, RI

Description:

The exclusive OR of the contents of the designated R-register and the contents of the location specified by the AS is loaded into R.

Operation:

[R#] <-- [R#] $\oplus$ [EA]

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

None.

5.2.11 Store R Under Control Of Mask, SRM

Format:

RR, RM, RI

Description:

Transfer bits of R# as specified by a Mask word to the corresponding bit positions of [EA]. The Mask word is the last word of the instruction.

Operation:

For $0 \leq i \leq 15$, if [M(i)] = 1 then [EA (i)] <-- [R#(i)]; otherwise, [EA(i)] is unchanged. (M = Mask word.)

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

1. If [M] = 0, use [R1] as the Mask word, and if [R1] = 0 treat the instruction as a NOP.

2. The use of IMO address form may cause alteration of procedure.

3. If AS requires additional words (i.e., IMA, displacement, descriptor), then these precede the Mask word.

5.2.12 AND With Register R, AND

Format:

RR, RM, RI

Description:

The logical AND of the contents of the designated R-register and the contents of the location specified by the AS is loaded into R.

Operation:

[R#] <-- [R#] /\ [EA]

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

None.

5.2.13 Halfword (Byte) Load Register R, LDH

Format:

RR, RM, RI

Description:

The byte specified by the AS is loaded (sign extended) into the designated R-register.

Operation:

[R#(8:15)] <-- byte addressed by EA; [R# (0:7)] <-- [R#(8)]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

The byte specified by the AS is a function of the AS format:

- o When a MAS with indexing is used, the byte used is as specified in subsection 3.12
- o When an IMO or MAS without indexing is used, the leftmost byte of the addressed location is selected.
- o When R-register AS is used, the right byte, i.e., [R#(8:15)], is taken.

5.2.14 Halfword (Byte) Store Register R, STH

Format:

RR, RM, RI

Description:

The least significant eight bits of the contents of the designated R-register are stored into the byte location specified by the AS.

Operation:

[Byte addressed by EA] <-- [R#(8:15)]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged

U Unchanged
OV Unchanged

Special Conditions:

1. The byte specified by the AS is as described in the LDH instruction under Special Conditions.

2. Use of IMO address form may cause alteration of procedure.

5.2.15 Halfword (Byte) Compare Register R, CMH

Format:

RR, RM, RI

Description:

The unsigned comparison of the contents of R# with the byte (sign extended) specified by the AS is used to set the G-, L-, and U-indicators.

Operation:

```
[TEMP (8:15)] <-- [Byte addressed by EA]
[TEMP (0:7)] <-- [TEMP(8)] - Sign Extend Operation
I(G), I(L), I(U) <-- [R#] :: [TEMP]
```

Indicator Conditions:

```
                                 C  Unchanged
                                 B  Unchanged
                                 I  Unchanged
If (R#) > [TEMP] then            G  <-- 1 else G <-- 0
If (R#) < [TEMP] then            L  <-- 1 else L <-- 0
If [R# (0)] ≠ [TEMP(0)] then     U  <-- 1 else U <-- 0
                                 OV Unchanged
```

Special Conditions:

The byte specified by the AS is as described in the LDH instruction under Special Conditions.

5.2.16 Halfword (Byte) OR With Register R, ORH

Format:

RR, RM, RI

Description:

The inclusive OR of the contents of the designated R-register and the byte (sign extended) specified by the AS is loaded into R.

Operation:

```
[TEMP (8:15)] <-- [Byte addressed by EA]
[TEMP (0:7)] <-- [TEMP (8)] Sign Extend Operation
[R#] <-- [R#] \/ [TEMP]
```

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

The byte specified by the AS is as described in the LDH instruction under Special Conditions.

5.2.17 Halfword (Byte) Exclusive OR With Register R, XOH

Format:

RR, RM, RI

Description:

The exclusive OR of the contents of the designated R-register and the byte (sign extended) specified by the AS is loaded into R.

Operation:

```
[TEMP (8:15)] <-- [Byte addressed by EA]
[TEMP (0:7)] <-- [TEMP (8)] Sign Extend Operation
[R#] <-- [R#] ⊕ [TEMP]
```

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

The byte specified by the AS is as described in the LDH instruction under Special Conditions.

5.2.18 Halfword (Byte) AND With Register R, ANH

Format:

RR, RM, RI

Description:

The logical AND of the contents of the designated R-register and the byte (sign extended) specified by the AS is loaded into R.

Operation:

```
[TEMP 8:15)] <-- [Byte addressed by EA]
[TEMP 0:7)] <-- [TEMP (8)] Sign Extend Operation
[R#) <-- [R#] /\ [TEMP]
```

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
H Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

The byte specified by AS is as described in the LDH instruction under Special Conditions.

5.2.19 Halfword (Byte) Load Logical Register R, LLH

Format:

RR, RM, RI

Description:

The byte specified by the address syllable is loaded into the right half of R#; i.e., R# (8:15). The left half of R#, i.e., R# (0:7) is cleared to Zero.

Operation:

```
[R# 8:15)] <-- [Byte addressed by EA]; [R# (0:7)] <-- 00
```

Indicator Condition:

C Unchanged
B Unchanged
I Unchanged
G Unchanged

L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

The byte specified by the AS is as described in the LDH instruction under Special Conditions.

5.2.20 Modify Or Test Register M, MTM

Format:

RR, RM, RI

Description:

The designated eight-bit register M is altered and/or tested as specified by the contents of the location specified by the AS.

Operation:

```
[I(B)] <— 0, then
for 0 ≤ i ≤ 7
If [EA (i)] = 1 then [M#(i)] <— [EA (i+8)] else
If [EA (i)] = 0, [EA (i+8)] = 1, and [M#(i)] = 1,
then [I(B)] <— 1 else
If [EA(i)] = 0, and [EA(i+8)] = 0, then [M#(i)] unchanged.
```

Indicator Conditions:

	C Unchanged
If for i = 0,1,....,6,7 [M#(i)] = 1 and [EA (i)] = 0 and [EA (i+8)] = 1 then	B <— 1 else B<— 0
	I Unchanged
	G Unchanged
	L Unchanged
	U Unchanged
	OV Unchanged

Special Conditions:

If M6 or M7 is specified in a CR41E then Trap TV05.

5.2.21 Store Register M, STM

Format:

RR, RM, RI

Description:

The designated eight-bit register M is loaded into the right half of the location specified by the AS. The left half of the location specified by the AS is set to all Ones.

Operation:

```
[EA (0:7)] <-- FF;
[EA (8:15)] <-- [M#]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. If M6 or M7 is specified in a CR41E then Trap TV05.

2. Use of IMO address form may cause alteration of procedure.

5.2.22 Load Register B, LDB

Format:

RR, RM, RI

Description:

The contents of the location specified by the AS are loaded into the designated B-register.

Operation:

```
[B#] <-- [EA]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. The two words beginning at EA are loaded into the designated B register. Bits 12 (or bits 8, if EMMU) through 15 of the EA are loaded as the high-order bits of B. [EA + 1] is loaded as the low-order 16 bits of B.

2. If high order bits of [EA] are non zero then Trap TV15. See subsection 3.2.1.2.

5.2.23 Store Register B, STB

Format:

RR, RM, RI

Description:

The contents of the designated B-register are stored in the location specified by the AS.

Operation:

[EA] <-- [B#]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. The contents of the designated B register is stored in the two words beginning at EA. Bits 12 (Bit 8 if EMMU) through 15 of EA contain the high-order bits of B. [EA + 1] contains the 16 low-order bits of B.

2. Use of IMO address form may cause alteration of procedure.

5.2.24 Compare With Register B, CMB

Format:

RR, RM, RI

Description:

The comparison of the contents of the designated B-register and the contents of the location specified by the AS is used to set the G- and L-indicators.

Operation:

I(G), I(L) <-- [B#] :: [EA]

Indicator Conditions:

```
                         C  Unchanged
                         B  Unchanged
                         I  Unchanged
  If [B#] > [EA] then G  <-- 1 else G <-- 0
  If [B#] < [EA[ then L  <-- 1 else L <-- 0
                         U  <-- X i.e., setting of U is undefined
                         OV Unchanged
```

Special Conditions:

1. [B#] and [EA] are treated as addresses, i.e., unsigned 20 or 24-bit integers.

2. If the high-order 12 bits (high order 8bits if EMMU) of [EA] are not Zero, then G <-- 0, L <--1.

5.2.25 Swap Register B, SWB

Format:

RR, RM, RI

Description:

The contents of the designated B-register are swapped with the contents of the location specified by the AS.

Operation:

[B#] <--> [EA]

Indicator Conditions:

```
C  Unchanged
B  Unchanged
I  Unchanged
G  Unchanged
L  Unchanged
U  Unchanged
OV Unchanged
```

Special Conditions:

1. The two words beginning at EA are swapped with B. The four high-order bits of B (8 high-order bits if EMMU) are swapped with [EA (12:15)] (or [EA (8:15)] if EMMU. The 16 low-order bits of B are swapped with [EA + 1].

2. Use of IMO address form may cause alteration of procedure.

3. If high order bits of [EA] are non zero then Trap TV15. See subsection 3.2.1.2.

5.2.26 Load Effective Address Into B, LAB

Format:

RM, RI

Description:

The EA specified by the AS is loaded into the designated B-register.

Operation:

[B#] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. For IMO form of AS, B# is loaded to point to the second word of the instruction.

2. Indexing and IMO AS act as if LAB had a one-word operand and B# is loaded to point to IMO.

3. If high order bits of EA are non zero then Trap TV15. See subsection 3.2.1.2.

5.2.27 Link Jump, LNJ

Format:

RM, RI

Description:

Store the program counter in the designated B-register, then jump to the location specified by the AS. On completion of the instruction, the contents of that B-register points to the instruction following LNJ. Thus, LNJ can be used for subroutine linkage, with B containing the return address.

Operation:

[B#] <-- [P]; [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If M1(J) = 1 then trap to TV02.

5.3 SINGLE OPERAND

Single operand instructions have the following format:

```
 0  1   3 4        8 9           15
---------------------------------
| 1 | X00 |    OP    |     AS      |
|---------------------------------|
|    Additional words as needed   |
---------------------------------
      or (M6X and M6XE only)
 0  1   3 4        8 9           15
---------------------------------
| 1 | X00 |    OP    |     ASN     |
|---------------------------------|
| OFFSET  | RFU |MAP#|    AS2,3    |
|---------------------------------|
|0       3 4   6 7  8 9          15|
-    Additional words as needed   -
|                                 |
---------------------------------
```

where X = 0 for all single operand instructions except
= 1 for scientific single operand instructions (reference Section 8);

OP = Opcode field;

AS = Address Syllable field.

Within this group the following types of instructions exist:

- o Modify operands
- o Bit instructions
- o Control instructions.

Depending upon whether the AS specifies a RAS, MAS or IMO form, these instructions are defined to have the following format:

R = Register only
M = Memory only
I = Immediate only.

These instructions are summarized in Table 5-4. The numerical representation is given in Table 5-5.

Table 5-4 Single Operand Instructions (Sheet 1 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
		MODIFY OPERANDS			
5.3.1	INC	Increment	[I(B)] <-- [EA(0)]; [EA] <-- [EA] + 0001	OV, C, B	Read Modify Write
5.3.2	DEC	Decrement	[EA] <-- [EA] + FFFF; [I(B)] <-- [EA](0)	OV, C, B	Read Modify Write
5.3.3	NEG	Negate	[EA] <-- 0000 - [EA]	OV, C	
5 3.4	CPL	Complement	[EA] <-- [EA] ⊕ FFFF		
5.3.5	CL	Clear	[EA] <-- 0000		
5.3.6	CLH	Clear Halfword	[EA] <-- 00		
5.3.7	CMZ	Compare with Zero	[EA] :: 0000	G, L, U	
5.3.8	CMN	Compare Address to Null	[EA(2:31)] :: 00000000	G, L	
5.3.9	CAD	Add Carry	[EA] <-- [EA] + I(C)	OV, C	
		CONTROL INSTRUCTIONS			
5.3.10	STS	Store S Register	[EA] <-- [S]		
5.3.11	JMP	Jump	[P] <-- EA		Trap TV02 if M1(J) = 1
5.3.12	ENT	Enter	[P] <-- EA [S.RN] <-- 00		Trap TV02 if M1(J) = 1

Table 5-4 Single Operand Instructions (Sheet 2 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
		<u>CONTROL INSTRUCTIONS</u> (Continued)			
5.3.13	LEV	Level	See text		Privi-leged In-struction
5.3.14	SAVE	Save Context	[EA] + 0, 1,...] <-- [B7-B1, I, R7-R1, M1]		Save and Restore are done under control of the Mask that follows the in-struction See rules in sub-section 3.3.2.2
5.3.15	RSTR	Restore Context	[B7-B1, I, R7-R1, M1] <-- [EA + 0, 1,...]	See text	
		<u>BIT INSTRUCTIONS</u>*			
5.3.16	LB	Load Bit	[I(B)] <-- [EA]	B	
5.3 17	LBF	Load Bit and set False	[I(B)] <-- [EA]; [EA] <-- 0	B	
5.3 18	LBT	Load Bit and set True	[I(B) <-- [EA]; [EA] <-- 1	B	
5.3.19	LBC	Load Bit and Complement	[I(B)] <-- [EA]; [EA] <-- $\overline{[EA]}$		
5.3.20	LBS	Load Bit and Swap	[I(B)] <---> [EA]	B	

*All instructions excluding LB execute in Read-Modify-Write mode.

If the AS involves indexing, the index value is aligned to count bits and [EA] is the bit thus addressed.

If the AS is not indexed, [EA] is the logical product of the addressed word and the 16-bit Mask following the instruction.

Table 5-4 Single Operand Instructions (Sheet 3 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
		DOUBLE WORD			
5.3.21	AID	Add Integer Double	[R6], [R7] <-- [R6], [R7] + [EA]	OV,C	
5.3.22	LDI	Load Double Word Integer	[R6], [R7] <-- [EA]		
5.3.23	SDI	Store Double Word Integer	[EA] <-- [R6, [R7]		
5.3.24	SID	Subtract Integer Double	[R6], [R7] <-- [R6],]R7] - [EA]	OV,C	

Table 5-5 Numerical Representation of Single Operand Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
5.3.3	8	2	0+m	n	NEG	Word
5.3.16	8	2	8+m	n	LB	Bit
5.3.11	8	3	8+m	n	JMP	Word
5.3.21	8	4	0+m	n	AID	DWord
5.3.24	8	4	8+m	n	SID	DWord
5.3.4	8	6	0+m	n	CPL	Word
5.3.5	8	7	0+m	n	CL	Word
5.3.6	8	7	8+m	n	CLH	Byte
5.3.17	8	8	0+m	n	LBF	Bit
5.3.2	8	8	8+m	n	DEC	Word
5.3.18	8	9	0+m	n	LBT	Bit
5.3.7	8	9	8+m	n	CMZ	Word
5.3.20	8	A	0+m	n	LBS	Bit
5.3.1	8	A	8+m	n	INC	Word
5.3.19	8	B	0+m	n	LBC	Bit
5.3.12	8	B	8+m	n	ENT	Word
5.3.10	8	C	0+m	n	STS	Word
5.3.22	8	C	8+m	n	LDI	DWord
5.3.23	8	D	0+m	n	SDI	DWord
5.3.8	8	D	8+m	n	CMN	DWord
5.3.13	8	E	0+m	n	LEV	Word
5.3.9	8	E	8+m	n	CAD	Word
5.3.14	8	F	0+m	n	SAVE	Word
5.3.15	8	F	8+m	n	RSTR	Word

Where m,n = coordinates of AS Map (see subsection 3.11)

5.3.1 Increment, INC

Format:

R, M, I

Description:

Increment by one the contents of the location specified by the AS.

Operation:

[I(B)] <-- [EA(0)]; [EA] <-- [EA] + 0001

Indicator Conditions:

```
If carry then         C  <-- 1 else C <-- 0
Before incrementing,
if [EA (0)] = 1 then  B  <-- 1 else B <-- 0
                      I  Unchanged
                      G  Unchanged
                      L  Unchanged
                      U  Unchanged
If overflow then      OV <-- 1 else OV <-- 0
```

The setting of OV for this instruction will not cause a trap.

Special Conditions:

1. Use of IMO address form may cause alteration of procedure.

2. Operand is addressed with a read-modify-write cycle.

5.3.2 Decrement, DEC

Format:

R, M, I

Description:

Decrement by one the contents of the location specified by the AS.

Operation:

[EA] <-- [EA] + FFFF; [I(B)] <-- [EA(0)]

Indicator Conditions:

```
If carry then                           C  <-- 1 else C <-- 0
After decrement if [EA(0)] = 1 then B   <-- 1 else B <-- 0
                                        I  Unchanged
                                        G  Unchanged
                                        L  Unchanged
                                        U  Unchanged
    If overflow then                    OV <-- 1 else OV <-- 0
```

The setting of OV for this instruction will not cause a trap.

Special conditions:

1. Use of IMO address form may cause alteration of procedure.
2. Operand is addressed with a read-modify-write cycle.

5.3.3 Negate, NEG

Format:

R, M, I

Description:

Two's complement on the contents of the location specified by the AS.

Operation:

[EA] <-- 0000 - [EA]

Indicator Conditions:

```
    If [EA] = 0 then         C  <-- 1 else C <-- 0
                             B  Unchanged
                             I  Unchanged
                             G  Unchanged
                             L  Unchanged
                             U  Unchanged
    If [EA] = 8000, then OV  <-- 1 else OV <-- 0
```

The setting of OV for this instruction will not cause a trap.

Special conditions:

Use of IMO address form may cause alteration of procedure.

5.3.4 Complement, CPL

Format:

R, M, I

Description:

One's complement on the contents of the location specified by the AS.

Operation:

```
[EA] <-- [EA] ⊕ FFFF
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Use of IMO address form may cause alteration of procedure.

5.3.5 Clear, CL

Format:

R, M, I

Description:

Zero is stored in the location specifed by the AS.

Operation:

```
[EA] <-- 0000
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Use of IMO address form may cause alteration of procedure.

5.3.6 Clear Halfword (Byte), CLH

Format:

R, M, I

Description:

Zero is loaded into the byte location specified by the AS.

Operation:

```
[Byte Addressed by EA] <-- 00
```

Indicator Conditions:

```
C  Unchanged
B  Unchanged
I  Unchanged
G  Unchanged
L  Unchanged
U  Unchanged
OV Unchanged
```

Special Conditions:

1. The byte specified by the AS is as described in the LDH instruction under Special Conditions. Refer to subsection 5.2.13.

2. The use of IMO address form may cause alteration of procedure.

5.3.7 Compare With Zero, CMZ

Format:

R, M, I

Description:

The unsigned comparison of Zero with the content of the location specified by the AS is used to set the G-, L-, and U-indicators.

Operation:

```
I(G), I(L), I(U), <-- [EA] :: 0000
```

Indicator Conditions:

```
                      C  Unchanged
                      B  Unchanged
                      I  Unchanged
If [EA] ≠ 0000 then G  <-- 1 else G <-- 0
                      L  <-- 0
If [EA](0) = 1 then U  <-- 1 else U <-- 0
                      OV Unchanged
```

Special Conditions:

None.

5.3.8 Compare Address To Null, CMN

Format:

R, M, I

Description:

Compare the contents of EA to a null address.

Operation:

```
I(G) <-- [EA] :: 00000000
I(L) <-- 0
I(U) <-- Undefined
```

Indicator Conditions:

```
                   C  Unchanged
                   B  Unchanged
                   I  Unchanged
If [EA] ≠ 0 then G  <-- 1 else G <-- 0
                   L  <-- 0
                   U  <-- X (i.e., setting of U is undefined)
                   OV Unchanged
```

Special Conditions:

The EA is defined to contain an address.

5.3.9 Add Carry, CAD

Format:

R, M, I

Description:

Add the carry bit I(C) to the contents of the location specified by the AS.

Operation:

```
[EA] <-- [EA] + [I(C)]
```

Indicator Conditions:

```
If carry then    C  <-- 1 else C <-- 0
                 B  Unchanged
                 I  Unchanged
```

G Unchanged
L Unchanged
U Unchanged

If overflow then OV <-- 1 else OV <-- 0

The setting of OV for this instruction does not cause a Trap TV06.

Special Conditions:

Use of IMO address form may cause alteration of procedure.

5.3.10 Store S-Register, STS

Format:

R, M, I

Description:

Store [S] in the word specified by the AS.

Operation:

[EA] <-- [S]

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

Use of IMO address form may cause alteration of procedure.

5.3.11 Jump, JMP

Format:

M

Description:

Jump to the memory location specified by the AS.

Operation:

[P] <-- EA

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the trace trap bit (i.e., M1(J)) is on, a Trap TV02 is performed after the JMP instruction has been executed.

5.3.12 Enter, ENT

Format:

M

Description:

Jump to memory location specified by the AS and change the process privilege (RCR) to ring 3.

Operation:

```
[P]    <-- EA; and
[S.RN] <-- 00.
```

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the trace trap bit (i.e., M1(J)) is on, a Trap TV02 is performed after the ENT instruction has been executed.

5.3.13 Level, LEV

Format:

R, M, I

Description:

Set/clear level "activity" flag bit, inhibit interrupts, or enable interrupts under control of the operand.

Operation:

Reference subsection 3.5.4.

Indicator Conditions:

C	See subsection 3.5.4
B	See subsection 3.5.4
I	See subsection 3.5.4
G	See subsection 3.5.4
L	See subsection 3.5.4
U	See subsection 3.5.4
OV	See subsection 3.5.4

Special Conditions:

The LEV instruction can only be executed if in ring 0 or 1, else Trap TV13.

5.3.14 Save Context, SAVE

Format:

M

Description:

Save registers under control of a 16-bit Mask word. The Mask word is the last word of the instruction.

Operation:

[EA + 0,1...] <— [B7 - B1, I, R7-R1, M1], under control of Mask word.

Indicator Condition:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. Refer to Restore instruction, special conditions.

2. If AS requires additional words (that is, IMA, displacement, descriptor), then these precede the Mask word.

5.3.15 Restore Context, RSTR

Format:

M

Description:

Restore registers under control of a 16-bit Mask word. The Mask word is the last word of the instruction.

Operation:

[B7 - B1, I, R7 - R1, M1] <-- [EA + 0, 1...]. under control of Mask word.

Indicator Conditions:

For Mask.I = 0

I is Unchanged

For Mask.I = 1

I <-- [EA+N]

Special Conditions:

Mask Format for SAVE and RSTR

```
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
----------------------------------------
|M R R R R R R R   B  B  B  B  B  B  B |
|                I                     | [[P]+(n-1)]
|1 1 2 3 4 5 6 7   1  2  3  4  5  6  7 |
----------------------------------------
```

If bit = 1 save/restore corresponding register

- o If Mask is all Zeros use R1 as mask; if R1 = 0 then treat the instruction as NOP.
- o For address syllable forms that require auto increment/decrement, the address is incremented or decremented by a unit of one word regardless of the number of mask bits that are on.
- o For address syllable form ↓FT↑, FT is incremented or decremented by 23 words regardless of the number of mask bits that are on.

For SAVE:

The processor registers selected by the mask bits are stored in memory in consecutive address locations starting at the specified location. The mask bits are scanned from right (bit 15) to left (bit 0). If the mask bit is On, the corresponding register is stored in memory and the address register is incremented. Only as many memory words as needed are used.

For RSTR:

The contents of memory in consecutive address locations starting at the specified location are placed in the registers selected by the mask word bits. The mask bits are scanned from right (bit 15) to left (bit 0). The contents of the starting address is placed in the register corresponding to the first mask bit that is on. The contents of the consecutive memory locations are placed in the appropriate registers as each mask bit that is on is encountered.

The high order bits of the address values being loaded in to the base registers must be zero else Trap TV15. Refer to subsection 3.2.1.2.

5.3.16 Load Bit, LB

Format:

R, M, I

Description:

The bit specified by the AS is copied into I(B).

Operation:

[I(B)] <-- [Bit addressed by EA]

Indicator Conditions:

```
                              C  Unchanged
If addressed bit = 1 then B  <-- 1 else B <-- 0
                              I  Unchanged
                              G  Unchanged
                              L  Unchanged
                              U  Unchanged
                              OV Unchanged
```

Special Conditions:

If AS involves indexing, the index value is aligned to count bits and [EA] is the bit thus addressed. If AS is not indexed, [EA] is the logical product of the addressed word and the 16-bit Mask in the last word of the instruction, and I(B) receives the logical OR of the selected bits. If the Mask is Zero, [R1] is used as the mask; if [R1] is Zero, I(B) is cleared.

5.3.17 Load Bit And Set False, LBF

Format:

R, M, I

Description:

The bits specified by the AS are loaded into I(B). The addressed bits are then cleared to Zero.

Operation:

[I(B)] <-- [EA]; [EA] <-- 0 (Bit position selected)

Indicator Conditions:

```
                               C  Unchanged
If addressed bit = 1 then B  <-- 1 else B <-- 0
                               I  Unchanged
                               G  Unchanged
                               L  Unchanged
                               U  Unchanged
                               OV Unchanged
```

Special Conditions:

1. See LB instruction (subsection 5.3.16).
2. Operand is addressed with a read-modify-write cycle.

5.3.18 Load Bit And Set True, LBT

Format:

R, M, I

Description:

The bits specified by the AS are loaded into I(B). The addressd bits are then set to One.

Operation:

[I(B)] <-- [EA]; [EA] <-- 1 (Bit position selected)

Indicator Conditions:

```
                               C  Unchanged
If addressed bit = 1 then B  <-- 1 else B <-- 0
                               I  Unchanged
                               G  Unchanged
                               L  Unchanged
                               U  Unchanged
                               OV Unchanged
```

Special Conditions:

1. See LB instruction (subsection 5.3.16).
2. Operand is addressed with a read-modify-write cycle.

5.3.19 Load Bit And Complement, LBC

Format:

R, M, I

Description:

The bits specified by the AS are loaded into I(B). The addressed bits are then complemented.

Operation:

[I(B)] <-- [EA]; [EA] <-- $[\overline{EA}]$ (Bit position selected)

Indicator Conditions:

```
                             C  Unchanged
If addressed bit = 1 then B  <-- 1 else B <-- 0
                             I  Unchanged
                             G  Unchanged
                             L  Unchanged
                             U  Unchanged
                             OV Unchanged
```

Special Conditions:

1. See LB instruction (subsection 5.3.16).
2. Operand is addressed with a read-modify-write cycle.

5.3.20 Load Bit And Swap, LBS

Format:

R, M, I

Description:

The bits specified by the AS are swapped with I(B).

Operation:

[I(B)] <--> [EA] (Bit position selected)

Indicator Conditions:

```
                                C  Unchanged
If addressed bit = 1 then B  <-- 1 else B <-- 0
                                I  Unchanged
                                G  Unchanged
                                L  Unchanged
                                U  Unchanged
                                OV Unchanged
```

Special Conditions:

1. See LB instruction (subsection 5.3.16).
2. Operand is addressed with a read-modify-write cycle.

5.3.21 Add Integer Double, AID

Format:

R, M, I

Description:

The sum of the double word integer contained in R6 and R7 and the contents of the double word location specified by the AS is loaded into R6, R7. R6(0) is considered the high-order bit; R7(15) is considered the low-order bit.

Operation:

[R6], [R7] <-- [R6], [R7] + [EA]

Indicator Conditions:

```
If carry then        C  <-- 1 else C <-- 0
                     B  Unchanged
                     I  Unchanged
                     G  Unchanged
                     L  Unchanged
                     U  Unchanged
   If overflow then OV <-- 1 else OV <-- 0
```

Special Conditions:

1. See LDI instruction (subsection 5.3.22 below).

2. If an overflow occurs and M1(6) = 1, then a Trap TV06 occurs.

5.3.22 Load Double Word Integer, LDI

Format:

R, M, I

Description:

Place contents of the EA into R6 and R7.

Operation:

[R6], [R7] <— [EA]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. RAS form of addressing is defined as follows:

#IN RAS	REGISTER PAIR SELECTED	COMMENT
3	R2, R3	
5	R4, R5	
7	R6, R7	
1,2,4,6	X	Operation Unspecified

2. IMO and MAS forms of AS select a two-word operand.

5.3.23 Store Double Word Integer, SDI

Format:

R, M, I

Description:

Place the contents of R6 and R7 into the location specified by EA.

Operation:

[EA] <— [R6], [R7]

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

1. See LDI instruction (subsection 5.3.22).

2. The use of IMO form may cause alteration of procedure.

5.3.24 Subtract Integer Double, SID

Format:

R, M, I

Description:

The difference of the double word integer contained in R6, R7 and the contents of the double word location specified by the AS is loaded into R6, R7. R6(0) is considered the high-order bit; R7(15) is considered the low-order bit.

Operation:

[R6], [R7] <-- [R6], [R7] - [EA] (using two's complement arithmetic)

Indicator Conditions:

```
If carry then    C  <-- 1 else C <-- 0
                 B  Unchanged
                 I  Unchanged
                 G  Unchanged
                 L  Unchanged
                 U  Unchanged
If overflow then OV <-- 1 else OV <-- 1
```

Special Conditions:

1. See LDI instruction (subsection 5.3.22).

2. If an overflow occurs and M1(6) = 1, then a Trap TV06 occurs.

5.4 SHORT VALUE IMMEDIATE

Short value immediate (SI) instructions have the following format:

```
 0   1    3 4      7 8           15
|-----------------------------------|
| 0 |  #  |   OP   |       V        |
|-----------------------------------|
```

where # = Selects one of the seven word operand registers (R);

OP = Opcode field;

V = Immediate Operand Value, sign extended range is: $-128 \leq V \leq +127$

These instructions operate on the R-registers and perform load, compare, add, and multiply operations.

These instructions are summarized in Table 5-6. Their numerical representation is given in Table 5-7.

Table 5-6 Short Value Immediate Instructions

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
5.4.1	LDV	Load Value	[R#] <-- V		In all cases V is sign extended
5.4.2	CMV	Compare with Value	[R#] :: V	G,L,U	
5.4.3	ADV	Add Value	[R#] <-- [R#] + V	C, OV	
5.4.4	MLV	Multiply by Value	[R#] <-- [R#] * V except if R# = 7, then (R6, R7) <-- (R7) * V	OV, if #≠7	

Table 5-7 Numerical Representation of Short Value Immediate Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
5.4.1	0+r	C	V		LDV	NOT
5.4.2	0+r	D	V		CMV	APPLI-
5.4.3	0+r	E	V		ADV	CABLE
5.4.4	0+r	F	V		MLV	

where r = Register number contained in bits 1 through 3 of the instruction

V = Immediate Operand Value.

5.4.1 Load Value, LDV

Format:

SI

Description:

Load the 8-bit V-field (sign extended) into the designated R-register.

Operation:

```
[R#(8:15)] <-- [V(8:15)] ; [R#(0:7)] <-- [V(8)]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

5.4.2 Compare With Value, CMV

Format:

SI

Description:

The comparison of the contents of the designated R-register and the 8-bit V-field (sign extended) is used to set G-, L-, and U-indicators.

Operation:

```
[TEMP(8:15)] <-- [V(8:15)] ;
[TEMP(0:7)] <-- [V(8)] ;
I(G), I(L), I(U) <-- [R#] :: [TEMP]
```

Indicator Conditions:

```
                          C  Unchanged
                          B  Unchanged
                          I  Unchanged
If [R#] > [TEMP] then G  <-- 1 else G <-- 0
If [R#] < [TEMP] then L  <-- 1 else L <-- 0
If R#(0) ≠ V(8) then  U  <-- 1 else U <-- 0
                          OV Unchanged
```

Special Conditions:

None.

5.4.3 Add Value, ADV

Format:

SI

Description:

The sum of the contents of the designated R-register and the V-field (sign extended) is loaded into R.

Operation:

```
[TEMP(8:15)] <-- [V(8:15)];
[TEMP(0:7)] <-- [V(8)];
[R#] <-- [R#] + [TEMP]
```

Indicator Conditions:

```
If carry then    C  <-- 1 else C <-- 0
                 B  Unchanged
                 I  Unchanged
                 G  Unchanged
                 L  Unchanged
                 U  Unchanged
If overflow then OV <-- 1 else OV <-- 0
```

Special Conditions:

If an overflow occurs and M1(#) = 1, then a Trap TV06 occurs.

5.4.4 Multiply By Value, MLV

Format:

SI

Description:

The product of the designated R-register contents and the eight-bit V-field (sign extended) is loaded into R. If # is 7, then the signed double precision product is loaded into R6 and R7 with R7 being loaded with the least significant portion of the product.

Operation:

```
[TEMP(8:15)] <-- [V(8:15)];
[TEMP(0:7)] <-- [V(8)];
[R#] <-- [R#] * [TEMP]
except if # = 7 then [R6, R7] <-- [R7]*[TEMP]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged

If Overflow then OV <-- 1 else OV <-- 0 except if # = 7,
then OV is cleared.

Special Conditions:

1. If OV <-- 1, [R#] is unchanged.

2. If an overflow occurs and M1(#) = 1, then a Trap TV06 occurs.

5.5 BRANCH ON REGISTER

General Branch on Register (BR) instructions have the format shown in Figure 5-1.

These instructions enable branching on selected R-registers; e.g., equal to zero, less than zero, increment and test, etc. These instructions are defined in summary form in Table 5-8. Their numerical representation is given in Table 5-9.

5.5.1 Branch If [R] Less Than Zero, BLZ

Format:

BR

Description:

Branch to EA if the contents of the R-register are negative.

Operation:

```
If [R# (0)] = 1 then [P] <-- EA
```

Indicator Condition:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BLZ instruction is executed.

```
             0  1   3 4       8 9          15
             -------------------------------
    WORD 1   | 0 | R# |    OP    | d ≠ 0 or 1 |  -64 ≤ d ≤ 63
             -------------------------------


             -------------------------------
    WORD 1   | 0 | R# |    OP    |   d = 1    |  -2^15 ≤ D ≤ 2^15-1
             -------------------------------
    WORD 2   |               D              |
             -------------------------------


             -------------------------------
    WORD 1   | 0 | R# |    OP    |   d = 0    |
             -------------------------------
    WORD 2   |                              |
             |--             IMA          --|
    WORD 3   |                              |
             -------------------------------
```

where R# = One of seven operand registers (R)

OP = Opcode - determines the branch condition

d = Displacement - defines how to compute the EA:

- o If d ≠ 0 or 1, EA = Pd + d, where Pd is the address of the word containing d (or D); $-64 \le d \le +63$; and d is a word displacement.
- o If d = 1, EA = Pd + D.
- o If d = 0, then EA = IMA;

Figure 5-1 Branch on Register Instruction Formats

Table 5-8 Branch On Register Instructions

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
5.5.1	BLZ	Branch if [R] less than 0	If [R#(0)] = 1, then [P] <-- EA		In all branch operations, if the branch condition is true (i.e., branch is executed) and if M1(J) = 1 then trap to TV02
5.5.2	BGEZ	Branch if [R] Greater than or Equal to 0	If [R#(0)] = 0, then [P] <-- EA		
5.5.3	BEZ	Branch if [R] Equal to 0	If [R#] = 0, then [P] <-- EA		
5.5.4	BNEZ	Branch if [R] Not Equal to 0	If [R#] ≠ 0, then [P] <-- EA		
5.5.5	BGZ	Branch if [R] Greater than 0	If [R#(1:15)] ≠ 0 and if [R#(0)] = 0, then [P] <-- EA		
5.5.6	BLEZ	Branch if [R] Less than or Equal to 0	If [R#(0)] = 1 or if [R#] = 0, then [P] <-- EA		
5.5.7	BODD	Branch if [R] is Odd	If [R#(15)] = 1, then [P] <-- EA		
5.5.8	BEVN	Branch if [R] is Even	If [R#(15)] = 0, then [P] <-- EA		
5.5.9	BINC	Branch and Increment	[R#] <-- [R#] + 0001; if [R#] ≠ 0 then [P] <-- EA		
5.5.10	BDEC	Branch and Decrement	[R#] <-- [R#] + FFFF; if [R#] ≠ FFFF, then [P] <-- EA		

Table 5-9 Numerical Representation of Branch On Register Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
5.5.10	0+r	7	0+d		BDEC	
5.5.9	0+r	7	8+d		BINC	
5.5.1	0+r	8	0+d		BLZ	NOT
5.5.2	0+r	8	8+d		BGEZ	
5.5.3	0+r	9	0+d		BEZ	APPLI-
5.5.4	0+r	9	8+d		BNEZ	
5.5.5	0+r	A	0+d		BGZ	CABLE
5.5.6	0+r	A	8+d		BLEZ	
5.5.8	0+r	B	0+d		BEVN	
5.5.7	0+r	B	8+d		BODD	

where r = register number contained in bits 1 through 3 of the instruction;

d = Seven-bit displacement.

5.5.2 Branch If [R] Greater Than Or Equal To Zero, BGEZ

Format:

BR

Description:

Branch to EA if the contents of the R-register are positive or Zero.

Operation:

If [R#(0)] = 0 then [P] <-- EA

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BGEZ instruction is executed.

5.5.3 Branch If [R] Equal To Zero, BEZ

Format:

BR

Description:

Branch to EA if the contents of the R-register are Zero.

Operation:

If [R#] = 0 then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BEZ instruction is executed.

5.5.4 Branch If [R] Not Equal To Zero, BNEZ

Format:

BR

Description:

Branch to EA if the contents of the R-register are not Zero.

Operation:

If [R#] ≠ 0 then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch will be executed) and M1(J) = 1, then a Trap TV02 occurs after the BNEZ instruction is executed.

5.5.5 Branch If [R] Greater Than Zero, BGZ

Format:

BR

Description:

Branch to EA if the contents of the R-register are greater than Zero.

Operation:

```
If ([R#] ≠ 0) /\ ([R#(0)] = 0) then [P] <-- EA
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BGZ instruction is executed.

5.5.6 Branch If [R] Less Than Or Equal To Zero, BLEZ

Format:

BR

Description:

Branch to EA if the contents of the R-register are less than or equal to Zero.

Operation:

```
If ([R#(0)] = 1) \/ ([R#] = 0) then [P] <-- EA
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged

G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BLEZ instruction is executed.

5.5.7 Branch If [R] Odd, BODD

Format:

BR

Description:

Branch to EA if the R-register contains an odd value.

Operation:

If [R#(15)] = 1 then [P] <-- EA

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BODD instruction is executed.

5.5.8 Branch If [R] Even, BEVN

Format:

BR

Description:

Branch to EA if the R-register contains an even value.

Operation:

If [R#(15)] = 0 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BEVN instruction is executed.

5.5.9 Branch And Increment, BINC

Format:

BR

Description:

Add one to the contents of the R-register. If the result is not Zero, then branch to the EA.

Operation:

[R#] <-- [R#] + 0001 ; if [R#] $\neq$ 0 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Condition:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BINC instruction is executed.

5.5.10 Branch And Decrement, BDEC

Format:

BR

Description:

Subtract one from the contents of the R-register. If the result is not equal to -1 (FFFF), then branch to the EA.

Operation:

[R#] <-- [R#] + FFFF ; if [R#] $\neq$ FFFF then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BDEC instruction is executed.

5.6 BRANCH ON INDICATOR

Branch On Indicator (BI) instructions have the format shown in Figure 5-2.

These instructions enable branching on various indicator conditions, e.g., carry, equal, less than, greater than, I/O bit, etc. These instructions are summarized in Table 5-10. Their numerical representation is given in Table 5-11.

5.6.1 Branch, B

Format:

BI

Description:

Branch to EA.

Operation:

[P] <-- EA

Indicator conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If M1(J) = 1, then a Trap TV02 occurs after the B instruction is executed.

5.6.2 No Operation, NOP

Format:

BI

```
          0  1              8 9          15
          -----------------------------------
WORD 1   | 0 |       OP       | d ≠ 0 or 1 |   -64 ≤ d ≤ 63
          -----------------------------------


          -----------------------------------
WORD 1   | 0 |       OP       |    d = 1   |   -2^15 ≤ D ≤ 2^15-1
          -----------------------------------
WORD 2   |                 D               |
          -----------------------------------


          -----------------------------------
WORD 1   | 0 |       OP       |    d = 0   |
          -----------------------------------
WORD 2   |                                 |
         |--               IMA           --|
WORD 3   |                                 |
          -----------------------------------
```

where OP = Opcode - determines the branch condition

d = Displacement - defines how to compute the EA:

- o If $d \neq 0$ or 1, EA = Pd + d, where Pd is the address of the word containing d (or D); $-64 \leq d \leq +63$; and d is a word displacement.
- o If d = 1, EA = Pd + D.
- o If d = 0, then EA = IMA.

Figure 5-2 Branch on Indicator Instruction Formats

Table 5-10 Branch On Indicator Instructions (Sheet 1 of 2)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
5.6.1	B	Branch	[P] <-- EA		In all branch operations if the branch condition is true (i.e., branch is executed) and if M1(J) = 1 then trap to TV02.
5.6.2	NOP	No Operation			
5.6.3	BE	Branch on Equal	If [I(L)]V[I(G)] = 0, then [P] <-- EA		
5.6 4	BNE	Branch on Not Equal	If [I(L)]V[I(G)] = 1, then [P] <-- EA		
5.6.5	BAL	Branch on Algebraic Less than	If [I(L)] ⊕ [I(U)] = 1, then [P] <-- EA		
5.6.6	BAGE	Branch on Algebraic Greater or Equal	If [I(L)] ⊕ [I(U)] = 0, then [P] <-- EA		
5.6.7	BAG	Branch on Algebraic Greater	If [I(G)] ⊕ [I(U)] = 1, then [P] <-- EA		
5.5.8	BALE	Branch on Algebraic Less than or Equal	If [I(G)] ⊕ [I(U)] = 0, then [P] <-- EA		
5.6.9	BL	Branch on Less than	If [I(L)] = 1, then [P] <-- EA		
5.6.10	BGE	Branch on Greater than or Equal	If [I(L)] = 0, then [P] <-- EA		
5.6.11	BG	Branch on Greater than	If [I(G)] = 1, then [P] <-- EA		
5.6.12	BLE	Branch on Less than or Equal	If [I(G)] = 0, then [P] <-- EA		

Table 5-10 Branch On Indicator Instructions (Sheet 2 of 2)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
5.6.13	BSU	Branch on Signs Unlike	If [I(U)] = 1, then [P] <— EA		In all branch operations if the branch condition is true (i.e., branch is executed) and if M1(J) = 1 then trap to TV02.
5.6.14	BSE	Branch on Signs Equal	If [I(U)] = 0, then [P] <— EA		
5.6.15	BCT	Branch on Carry True	If [I(C)] = 1, then [P] <— EA		
5.6.16	BCF	Branch on Carry False	If [I(C)] = 0, then [P] <— EA		
5.6.17	BBT	Branch on Bit test indicator True	If [I(B)] = 1, then [P] <— EA		
5.6.18	BBF	Branch on Bit test indicator False	If [I(B)] = 0, then [P] <— EA		
5.6.19	BIOT	Branch on I/O indicator True	If [I(I)] = 1, then [P] <— EA		
5.6.20	BIOF	Branch on I/O indicator False	If [I(I)] = 0, then [P] <— EA		
5.6.21	BOV	Branch on Overflow	If [I(OV)] = 1, then [P] <— EA		
5.6.22	BNOV	Branch on No Overflow	If [I(OV)] = 0, then [P] <— EA		

Table 5-11 Numerical Representation of Branch On Indicator Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
5.6.9	0	2	0+d		BL	
5.6.10	0	2	8+d		BGE	
5.6.11	0	3	0+d		BG	
5.6.12	0	3	8+d		BLE	
5.6.21	0	4	0+d		BOV	
5.6.22	0	4	8+d		BNOV	
5.6.17	0	5	0+d		BBT	
5.6.18	0	5	8+d		BBF	NOT
5.6.15	0	6	0+d		BCT	
5.6.16	0	6	8+d		BCF	APPLI-
5.6.19	0	7	0+d		BIOT	
5.6.20	0	7	8+d		BIOF	CABLE
5.6.5	0	8	0+d		BAL	
5.6.8	0	8	8+d		BAGE	
5.6.3	0	9	0+d		BE	
5.6.4	0	9	8+d		BNE	
5.6.7	0	A	0+d		BAG	
5.6.8	0	A	8+d		BALE	
5.6.13	0	B	0+d		BSU	
5.6.14	0	B	8+d		BSE	
5.6.2	0	F	0+d		NOP	
5.6.1	0	F	8+d		B	

where d = 7-bit displacement.

Description:

No operation is performed.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

5.6.3 Branch On Equal, BE

Format:

BI

Description:

Branch to EA if I indicates equality.

Operation:

If [I(L)] \/ [I(G)] = 0 then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BE instruction is executed.

5.6.4 Branch On Not Equal, BNE

Format:

BI

Description:

Branch to EA if I indicates inequality.

Operation:

If [I(L)] \/ [I(G)] = 1 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BNE instruction is executed.

5.6.5 Branch On Algebraic Less Than, BAL

Format:

BI

Description:

Branch to EA if I indicates an algebraic less than.

Operation:

If [I(L)] @ [I(U)] = 1 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then Trap TV02 occurs after the BAL instruction is executed.

2. Should not be used following CMB or CMN instructions, as [I(U)] is undefined.

5.6.6 Branch On Algebraic Greater Than Or Equal, BAGE

Format:

BI

Description:

Branch to EA if I indicates an algebraic greater than or equal.

Operation:

If [I(L)] $\oplus$ [I(U)] = 0 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BAGE instruction is executed.

2. Should not be used following CMN or CMB instructions, as [I(U)] is undefined.

5.6.7 Branch On Algebraic Greater Than, BAG

Format:

BI

Description:

Branch to EA if I indicates an algebraic greater than.

Operation:

If [I(G)] $\oplus$ [I(U)] = 1 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BAG instruction is executed.

2. Should not be used following CMN or CMB instructions, as [I(U)] as undefined.

5.6.8 Branch On Algebraic Less Than Or Equal, BALE

Format:

BI

Description:

Branch to EA if I indicates an algebraic less than or equal to.

Operation:

```
If [I(G)] ⊕ [I(U)] = 0 then [P] <-- EA
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. If the branch conditions is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BALE instruction is executed.

2. Should not be used following CMN or CMB instructions, as [I(U)] is undefined.

5.6.9 Branch On Less Than, BL

Format:

BI

Description:

Branch to EA if I indicates less than.

Operation:

If [I(L)] = 1 then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BL instruction is executed.

5.6.10 Branch On Greater Than Or Equal, BGE

Format:

BI

Description:

Branch to EA if I indicates an algebraic greater than or equal to.

Operation:

If [I(L)] = 0 then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BGE instruction is executed.

5.6.11 Branch On Greater Than, BG

Format:

BI

Description:

Branch to EA if I indicates greater than.

Operation:

If [I(G)] = 1 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BG instruction is executed.

5.6.12 Branch On Less Than Or Equal, BLE

Format:

BI

Description:

Branch to EA if I indicates an algebraic less than or equal to.

Operation:

If [I(G)] = 0 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged

G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BLE instruction is executed.

5.6.13 Branch On Signs Unlike, BSU

Format:

BI

Description:

Branch to EA if I indicates that signs are unlike.

Operation:

If [I(U)] = 1 then [P] <-- EA

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

1. If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BSU instruction is executed.

2. Should not be used following CMB or CMN instruction, as [I(U)] is undefined.

5.6.14 Branch On Signs Equal, BSE

Format:

BI

Description:

Branch to EA if I indicates that signs are equal.

Operation:

If [I(U)] = 0 then [P] <-- EA

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

1. If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BSE instruction is executed.

2. Should not be used following CMB or CMN instructions, as [I(U)] is undefined.

5.6.15 Branch On Carry True, BCT

Format:

BI

Description:

Branch to EA if the carry indicator is true.

Operation:

```
If [I(C)] = 1 then [P] <-- EA
```

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BCT instruction is executed.

5.6.16 Branch On Carry False, BCF

Format:

BI

Description:

Branch to EA if the carry indicator is false.

Operation:

If [I(C)] = 0 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BCF instruction is executed.

5.6.17 Branch On Bit Test Indicator True, BBT

Format:

BI

Description:

Branch to EA if the bit test indicator is true.

Operation:

If [I(B)] = 1 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BBT instruction is executed.

5.6.18 Branch On Bit Test Indicator False, BBF

Format:

BI

Description:

Branch to the EA if bit test indicator is false.

Operation:

If [I(B)] = 0 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BBF instruction is executed.

5.6.19 Branch On I/O Indicator True, BIOT

Format:

BI

Description:

Branch to EA if I/O test indicator is true.

Operation:

If [I(I)] =1 then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BIOT instruction is executed.

5.6.20 Branch On I/O Indicator False, BIOF

Format:

BI

Description:

Branch to EA if IO test indicator is false.

Operation:

If [I(I)] = 0 then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, the a Trap TV02 occurs after the BIOF instruction is executed.

5.6.21 Branch On Overflow, BOV

Format:

BI

Description:

Branch to EA if the overflow indicator (OV) is true.

Operation:

If [I(OV)] = 1 then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged

G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BOV instruction is executed.

5.6.22 Branch On No Overflow, BNOV

Format:

BI

Description:

Branch to EA if the overflow indicator (OV) is false.

Operation:

If [I(OV)] = 0 then [P] <— EA

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a Trap TV02 occurs after the BNOV instruction is executed.

5.7 SHIFT OPERATIONS

The shift instruction has two formats: shift short (SHS) and shift long (SHL):

1. Shift short (SHS) format:

0	1 3	4 7	8	9 11	12 15
0	R#	0000	0	T	d

where R# = Selects one of seven operand registers (R)

T = Identifies type and direction of the shift

d = Distance ($1 \leq d \leq 15$); if d = 0, substitute content of R1 (12:15) for distance. If [R1(12:15)] = 0, the register's contents are unchanged and the indicators relevant to the instruction are cleared.

2. Shift long (SHL) format:

0	1 3	4 7	8	9 10	11 15
0	R#	0000	1	T	D

where R# = Selects one of the three word operand register pairs (R) where # must be 3, 5, or 7 or the operation is unspecified

T = Identifies type and direction of the shift

D = Distance ($1 \leq D \leq 31$); if D = 0, substitute contents of R1(11:15) for the distance. If [R1(11:15)] = 0, then the register's contents are unchanged and the indicators relevant to the instruction are cleared.

Various types of shifts on single or double registers (two registers linked together) are possible, e.g., closed, open, arithmetic, left, right, etc. For double shifts # must equal 3, 5, or 7, otherwise the operation is unspecified. Double shift pairing of registers is as follows, where #' is the implied register linked together with #.

#-1	#
2	3
4	5
6	7

Some shift operations modify I(C) or I(OV). I(C) will reflect the state of the last bit shifted out . When I(C) or I(OV) is to be modified and the actual shift distance is Zero (i.e., content of R1 (11/12:15) is Zero), then I(C) or I(OV) is cleared to Zero. The various types of shift instructions available are summarized in Table 5-12. The numerical representation is given in Table 5-13.

Table 5-12 Shift Operations (Sheet 1 of 2)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED
SHIFT SHORT (SHS) - SHIFT DISTANCE (N) IS 1 < N < 15				

5.7.1	SOL	Single Shift Open Left	`0 15`
			`I(C) <--| <-- |<-- 0`
			`^`
			`|`
			`-- Saves the last bit shifted out of R#(0)`
				C

5.7.2	SCL	Single Shift Closed Left	`0 15`
			`-<-| <-- |<---`
			`| |`
			`---------->---------`	

5.7.3	SAL	Single Shift Arithmetic Left	`0 1 15`
			`I(OV) <--|S| <-- |<-- 0`
			`^`
			`|`
			`-- Set to 1 if R#(0) changes during shift`
				OV

5.7.4	DCL	Double Shift Closed Left	`0 15 0 15`
			`-<-| R#-1 |<--| R# |<---`
			`| |`
			`------------>------------`	

5.7.5	SOR	Single Shift Open Right	`0 15`
			`0 -->| --> |--> I(C)`
			`^`
			`|`
			`Saves last bit shifted -- out of R#(15)`
				C

5.7.6	SCR	Single Shift Closed Right	`0 15`
			`--->| --> |->-`
			`| |`
			`---------<----------`	

5.7.7	SAR	Single Shift Arithmetic Right	`0 1 15`
			`-->|S| ---> |--> I(C)`
			`| | ^`
			`--<-- |`
			`Saves last bit --- shifted out of R#(15)`
				C

Table 5-12 Shift Operations (Sheet 2 of 2)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED
SHIFT SHORT (SHS) - SHIFT DISTANCE (N) IS 1 < N < 15 (Cont.)				
5.7.8	DCR	Double Shift Closed Right	0 15 0 15 —>¦ R#-1 ¦—>¦ R# ¦->- ¦ ¦ ————————<————————	
SHIFT LONG (SHL) - SHIFT DISTANCE (N) IS 1 < N < 31				
5.7.9	DOL	Double Shift Open Left	0 15 0 15 I(C)<-¦ R#-1 ¦<-¦ R# ¦<-0 ^ ¦ — Saves last bit shifted out of R#-1(0)	C
5.7.10	DAL	Double Shift Arithmatic Left	0 1 15 0 15 I(OV)<-¦S¦R#-1¦<-¦ R# ¦<-0 ^ ¦ — Set to 1 if R#-1(0) changes during shift	OV
5.7.11	DOR	Double Shift Open Right	0 15 0 15 0->¦R#-1 ¦->¦ R# ¦->I(C) ^ ¦ Saves last bit — shifted out of R#(15)	C
5.7.12	DAR	Double Shift Arithmetic Right	0 1 15 0 15 —>¦S¦R#-1¦->¦ R# ¦->I(C) ¦ ^ ^ ¦ ¦ ¦ ——— ¦ Saves last bit — shifted out of R#(15)	C

Table 5-13 Numerical Representation of Shift Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
5.7.1	r	0	0	d	SOL	
5.7.2	r	0	1	d	SCL	
5.7.3	r	0	2	d	SAL	
5.7.4	r	0	3	d	DCL	NOT
5.7.5	r	0	4	d	SOR	
5.7.6	r	0	5	d	SCR	APPLI-
5.7.7	r	0	6	d	SAR	
5.7.8	r	0	7	d	DCR	CABLE
5.7.9	r	0	8	D	DOL*	
5.7.9	r	0	9	D-16	DOL**	
5.7.10	r	0	A	D	DAL*	
5.7.10	r	0	B	D-16	DAL**	
5.7.11	r	0	C	D	DOR*	
5.7.11	r	0	D	D-16	DOR**	
5.7.12	r	0	E	D	DAR*	
5.7.12	r	0	F	D-16	DAR**	

where r selects one of the seven operand registers in bits 1 through 3 of the instruction;

d = Distance in bits, $1 \leq d \leq 15$.

D = Distance in bits, $1 \leq D \leq 31$;

* = For $D \leq 15$; and

** = For $D > 15$.

5.7.1 Single Shift Open Left, SOL

Format:

SHS

Description:

The contents of R# are shifted d bit positions left. Zeros fill the d least significant bit positions of R#. The last bit shifted out of R# (0) is saved in the C-indicator.

Operation:

```
                     0                 15
                    ----------------------
      I(C) <----  |         <--          | <---- 0
        ^          ----------------------
        |                   R#
        -- Saves the last bit shifted out of R#(0)
```

Indicator Conditions:

If d ≠ 0 and if the last bit shifted out of R# (0) = 1 then	C	<—1 else C <— 0
	B	Unchanged
	I	Unchanged
	G	Unchanged
	L	Unchanged
	U	Unchanged
	OV	Unchanged

Special Conditions:

If d = 0 then the R-register's contents are unchanged and I(C) is cleared.

5.7.2 Single Shift Closed Left, SCL

Format:

SHS

Description:

The contents of R# are shifted d bit positions left. Bits shifted out of [R#(0)] replace bits vacating [R# (15)].

Operation:

```
            0                   15
            ---------------------
   --<--  |        <--         | <----
   |        ---------------------     |
   |                 R#               |
   ---------------->------------------
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If d = 0 then the R-register's contents are unchanged.

5.7.3 Single Shift Arithmetic Left, SAL

Format:

SHS

Description:

The contents of R# are shifted d bit positions left. If the sign bit (i.e., R#(0)) changes at any time during the operation, the OV indicator is set. Zeros fill the d least significant bit positions of R#.

Operation:

```
              0  1             15
             ----------------------
   I(OV)<--| S |                  | <---- 0
     |       ----------------------
     |              R#
    --Set to 1 if R#(0) changes during shift.
```

Indicator Conditions:

```
                         C  Unchanged
                         B  Unchanged
                         I  Unchanged
                         G  Unchanged
                         L  Unchanged
                         U  Unchanged
   If [R#(0)] changes then OV <-- 1 else OV <-- 0
```

Special Conditions:

1. If d = 0 then the R registers' contents are unchanged and I(OV) is cleared.

2. If an overflow occurs and M1(#) = 1, then a Trap TV06 occurs.

5.7.4 Double Shift Closed Left, DCL

Format:

SHS

Description:

The contents of the even-odd register pair are shifted d bit positions left. Bits shifted out of [R#'(0)] of the even register replace bits vacating [R#(15)] of the odd register.

Operation:

```
        0              15    0              15
       ----------------    ----------------
   -<-|     R# - 1     |<--|      R#       |<--
   |   ----------------    ----------------    |
   ---------------------->----------------------
```

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged

G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

1. # in instructions must specify the odd registers 3, 5, or 7, else the operation is undefined.

2. If d = 0, then the R-register's contents are unchanged.

5.7.5 Single Shift Open Right, SOR

Format:

SHS

Description:

The contents of R# are shifted d bit positions right. Zeros fill the d most significant bit positions of R#. The last bit shifted out of [R#(15)] is saved in the C-indicator.

Operation:

```
           0                    15
           ----------------------
0  ---->  |     ---------->      |----> I(C)
           ----------------------          ^
                    R#                     |
Saves the last bit shifted out of R#(15)--
```

Indicator Conditions:

```
If d ≠ 0, and the last bit
shifted out of [R#(15)] = 1 then C  <-- 1 else 0
                                 B  Unchanged
                                 I  Unchanged
                                 G  Unchanged
                                 L  Unchanged
                                 U  Unchanged
                                 OV Unchanged
```

Special Conditions:

If d = 0 then the R-register's contents are unchanged and I(C) is cleared.

5.7.6 Single Shift Closed Right, SCR

Format:

SHS

Description:

The contents of [R#] are shifted d bit positions right. Bits shifted out of [R#(15)] replace bits vacating [R#(0)].

Operation:

```
          0                   15
          ----------------------
 ---->  |    ------------>     |--->--
 |        ----------------------     |
 |                 R#                |
 ------------------<------------------
```

Indicator Conditions:

```
C  Unchanged
B  Unchanged
I  Unchanged
G  Unchanged
L  Unchanged
U  Unchanged
OV Unchanged
```

Special Conditions:

If d = 0 then the R-register's contents are unchanged.

5.7.7 Single Shift Arithmetic Right, SAR

Format:

SHS

Description:

The contents of R# are shifted d positions right. Bits equal to the sign bit of the R-register fill the d most significant bits of the R-register. The last bit shifted out of [R#(15)] is saved in the C-indicator.

Operation:

```
       0  1                    15
      ----------------------------
  -->| S |         -->           | --> I(C)
  |   ----------------------------      ^
  |    ^            R#                  |
  |    |                                |
  ------                                |
Saves the last bit shifted out of R#(15)--
```

Indicator Conditions:

```
If d ≠ 0 and the last bit
shifted out of [R#(15)] = 1 then C  <-- 1 else C <-- 0
                                 B  Unchanged
                                 I  Unchanged
                                 G  Unchanged
                                 L  Unchanged
                                 U  Unchanged
                                 OV Unchanged
```

Special Conditions:

If d = 0 then the R registers' contents are unchanged and I(C) is cleared.

5.7.8 Double Shift Closed Right, DCR

Format:

SHS

Description:

The contents of the even-odd register pair are shifted d bit positions right. Bits shifted out of [R# (15)] (the odd register) replace bits vacating [R#'(0)] (the even register).

Operation:

```
        0              15   0              15
        ----------------    ----------------
    -->|     R# - 1     |-->|      R#      |->-
    |   ----------------    ----------------  |
    --------------------<---------------------
```

Indicator Conditions:

```
C  Unchanged
B  Unchanged
I  Unchanged
G  Unchanged
L  Unchanged
U  Unchanged
OV Unchanged
```

Special Conditions:

1. # in instruction must specify registers 3, 5, or 7, else operation is undefined.

2. If d = 0 then the R-register's contents are unchanged.

5.7.9 Double Shift Open Left, DOL

Format:

SHL

Description:

The contents of the even-odd register pair are shifted d bit positions left. Zeros fill the d least significant bit positions of the register pair. The last bit shifted out of [R#'(0)] (i.e., bit 0 of the even register) is saved in the C-indicator.

Operation:

```
             0              15    0              15
             ----------------     ----------------
I(C)<--|       R# - 1      |<--|       R#       |<-- 0
   ^         -----------------   ----------------
   |
   -- Saves the last bit shifted out of R#-1(0)
```

Indicator Conditions:

```
If d ≠ 0 and the last bit
shifted out of R#' = 1 then C  <-- 1 else C <-- 0
                            B  Unchanged
                            I  Unchanged
                            G  Unchanged
                            L  Unchanged
                            U  Unchanged
                            OV Unchanged
```

Special Conditions:

1. # in instruction must specify registers 3, 5,or 7 else the operation is undefined.

2. If d = 0 then the R-registers' contents are unchanged and I(C) is cleared.

5.7.10 Double Shift Arithmetic Left, DAL

Format:

SHL

Description:

The contents of the even-odd register pair are shifted d positions left. If the sign bit (i.e., [R#'(0)]) changes at any time during operation, then the overflow indication is set. Zeros fill the d least significant bit positions of the register pair.

Operation:

```
                0 1            15    0              15
                ---------------      ---------------
   I(OV)<--|S|     R# - 1     |<--|       R#        |<-- 0
     ^          ---------------      ---------------
     |
     -- Set to 1 if R#-1(0) changes suring shift
```

Indicator Conditions:

```
                          C  Unchanged
                          B  Unchanged
                          I  Unchanged
                          G  Unchanged
                          L  Unchanged
                          U  Unchanged
If [R#'(0)] changes then OV <-- 1 else OV <-- 0
```

Special Conditions:

1. # in the instruction must specify registers 3, 5, or 7, else operation is undefined.
2. If d = 0 then the R-registers' contents are unchanged and I(OV) is cleared.
3. If an overflow occurs and M1(#) = 1, then a Trap TV06 occurs.

5.7.11 Double Shift Open Right, DOR

Format:

SHL

Description:

The contents of the even-odd register pair are shifted d positions right. Zeros fill the d most significant bit positions of the register pair. The last bit shifted out of [R#(15)] (i.e., bit 15 of the odd register) is saved in the C-indicator.

Operation:

```
          0             15    0              15
          --------------      ---------------
   0 -->|    R# - 1    |-->|       R#       |--> I(C)
          --------------      ---------------       ^
                                                    |
        Saves the last bit shifted out of R#(15) ---
```

Indicator Conditions:

```
   If d ≠ 0 and the last bit
   shifted out of R#(15) = 1 then C  <-- 1 else C <-- 0
                                  B  Unchanged
                                  I  Unchanged
                                  G  Unchanged
```

```
                                          L  Unchanged
                                          U  Unchanged
                                          OV Unchanged
```

Special Conditions:

1. # in instruction must specify registers 3, 5, or 7, else operation is undefined.

2. If d = 0 then the R-registers' contents are unchanged and I(C) is cleared.

5.7.12 Double Shift Arithmetic Right, DAR

Format:

SHL

Description:

The contents of the even-odd register pair are shifted d bit positions right. Bits equal to [R#'(0)] (sign bit of the even register) fill the d most significant bits of the register pair. The last bit shifted out of [R#(15)] (i.e., bit 15 of the odd register) is saved in the C-indicator.

Operation:

```
      0  1            15     0             15
      --------------------   ----------------
---->| S |     R# - 1    |-->|      R#      |--> I(C)
|     --------------------   ----------------     ^
|     |                                           |
---<---   Saves the last bit shifted out of R#(15) --
```

Indicator Conditions:

```
If d ≠ 0 and the last bit
shifted out of R# (15) = 1 then C  <-- 1 else C <-- 0
                                B  Unchanged
                                I  Unchanged
                                G  Unchanged
                                L  Unchanged
                                U  Unchanged
                                OV Unchanged
```

Special Conditions:

1. # in the instruction must specify register 3, 5, or 7, else operation is undefined.

2. If d = 0 then the R-registers' contents are unchanged and I(C) is cleared.

5.8 INPUT/OUTPUT

I/O instructions have two formats:

- o Data and Command I/O (IO and IOH)
- o Address and Range Output (IOLD).

The governing document for I/O standards is the Extended Megabus EPS-1 (60126298). This subsection describes the format of the I/O instructions. For the purpose of programming I/O, see the bus and appropriate controller specifications. Also refer to section 10 of this EPS-1.

Data and Command I/O (IO and IOH) Instructions

These instructions specify two quantities, (1) a Data word or byte indentified by an AS called Data AS (DAS) analogous to the single operand instruction formats, and (2) a Control word identifying the external channel (or device) and the function it is to perform. The control word may be embedded in the procedure as follows:

```
                 0  1    3 4       8 9        15
                ------------------------------------
For             | 1 | 0 0 0 |    OP    |    DAS    | :
control         |----------------------------------| :  Points to data
word            |      Additional word(s) if       | :- word or byte
embedded        |         needed by DAS            | :
in the          |----------------------------------|-: Is the control
procedure       |         CH              |   F    | :- word (CW) when
                ------------------------------------    CW(0:8) ≠ 0
                0                        9 10     15
```

or the control word can be pointed to by the Channel Address Syllable (CAS) as follows:

```
                 0  1    3 4       8 9        15
                ------------------------------------
                | 1 | 0 0 0 |    OP    |    DAS    |
                |----------------------------------|
For             |      Additional word(s) if       |
control         |         needed by DAS            |
word            |----------------------------------|-
pointed         |  0 0 0 0 0 0 0 0 0  |    CAS     | :
to by CAS       |----------------------------------| :_ Points to the
                |      Additional word(s) if       | :  control word
                |         needed by CAS            | :  when CW(0:8) = 0
                ------------------------------------
```

If either DAS or CAS use an AS from an AS Map other than AS Map 1 (M6X and M6XE only), the following applies:

```
                 0  1     3 4    7   8 9         15
                ------------------------------------
                | 1 | 0 0 0 |    OP    | DAS = ASN  | :
                |-----------------------------------| :  For DAS using
   When         | OFFSET   | RFU |MAP #|   AS2,3    | :_ an AS from a
   either       |-----------------------------------| :  Map other than
   DAS or       |      Additional word(s) if        | :  Map 1
   CAS use      |         needed by DAS             | :
   an AS        |-----------------------------------|--
   from a       | 0 0 0 0 0 0 0 0 0 0  | CAS = ASN  | :
   Map other    |-----------------------------------| :  For CAS using
   than AS      | OFFSET   | RFU |MAP #|   AS2,3    | :_ an AS from a
   Map 1        |-----------------------------------| :  Map other than
                |      Additional word(s) if        | :  Map 1
                |         needed by CAS             | :
                ------------------------------------
```

where OP = Opcode field

DAS = Data Address Syllable - Specifies a location from/to which a data word or byte is transferred to/from the I/O bus

CH = Channel number (or device address), where:
- o CH is odd for output DMA transfer channels
- o CH is even for input DMA transfer channels.

F = Function Code, which is controller specific under the following constraints:

- o If F is even, data will be transferred from the controller to the CPU
- o If F is odd, data or commands will be transferred from the CPU to the controller

CAS = Control Address Syllable - Points to control word containing CH and F.

OFFSET = Specifies the atom offset when using a word address form or is not used if using a byte address form.

Address and Range Output Instruction

This instruction specifies three quantities:

1. Buffer virtual address identified by an AS analogous to that in the single operand instruction format.

2. Control word identifying the external channel (or device) and the function it is to perform.

3. Range identified by an AS analogous to that in the single operand instruction format.

The control word may be embedded in the procedure as follows:

```
                 0 1   3 4       8 9 10      15
                 ----------------------------------
For              | 1 | 0 0 0 |   OP   |    AAS    | :  Points
control          |--------------------------------| :- to the
word             | Additional word(s) if needed by AAS | :  Buffer
embedded         |--------------------------------|--
in the           |          CH          |    F    | :- Is the control word
procedure        |--------------------------------|--  (CW) when CW(0:8) ≠ 0
                 | 0 0 0 0 0 0 0 0 0 |    RAS    | :
                 |--------------------------------| :- Points to the
                 | Additional word(s) if needed by RAS | :  word that
                 ----------------------------------     specifies the
                                                        range of the buffer
```

or the control word can be pointed to by the CAS as follows:

```
                 0 1   3 4       8 9         15
                 --------------------------------
                 | 1 | 0 0 0 |   OP   |   AAS   |
                 |------------------------------|
                 | Additional word(s) if needed by AAS |
For              |------------------------------|--
control          | 0 0 0 0 0 0 0 0 0 |   CAS   | :  Points to the
word             |------------------------------| :- control word
pointed to       | Additional word(s) if needed by CAS | :  when CW(0:8) = 0
by CAS           |------------------------------|--
                 | 0 0 0 0 0 0 0 0 0 |   RAS   |
                 |------------------------------|
                 | Additional word(s) if needed by RAS |
                 --------------------------------
```

If any of the ASs use an AS from an AS Map other than AS Map 1 (M6X and M6XE only), then the following applies:

```
                 0  1     3 4    7   8 9           15
                 --------------------------------------
                 | 1 | 0 0 0 |    OP    |  AAS = ASN   | :   For AAS using
                 |-------------------------------------| :   an AS from a
                 |  OFFSET   | RFU|MAP #|    AS2,3     | :-  Map other than
                 |-------------------------------------| :   Map 1
When             | Additional word(s) if needed by AAS | :
either a         |-------------------------------------|--
DAS or CAS       | 0 0 0 0 0 0 0 0 0    |  CAS ≠ ASN   | :   From CAS using
uses an AS       |-------------------------------------| :-  an AS from
from a Map       | Additional word(s) if needed by CAS | :   Map 1
other than       |-------------------------------------|--
Map 1            |  0 0 0 0 0 0 0 0 0   |  RAS = ASN   | :   For RAS using
                 |-------------------------------------| :   an AS from a
                 |  OFFSET   | RFU|MAP #|    AS2,3     | :-  Map other than
                 |-------------------------------------| :   Map 1
                 | Additional word(s) if needed by RAS | :
                 |--------------------------------------
```

where AAS = Address Address Syllable, pointing to an address

OFFSET = Specifies the atom offset when using a word address form or is not used if using a byte address form

RAS = Range Address Syllable, pointing to a range word.

The F-field in the Address and Range output instruction must specify the function code needed to load the controller address register. If F is other than above, the operation is unspecified.

The type of I/O instructions available are summarized in Table 5-14. The numerical representation is given in Table 5-15.

Table 5-14 I/O Instructions (Sheet 1 of 2)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
5.8.2.1	IO	Word Input/Output	For F = odd, output word to channel For F = even, receive word from channel	I set if accepted	Privileged instruction
5.8.2.2	IOH	Halfword Input/Output	For F = odd, output byte to channel For F = even, receive byte from channel	I set if accepted	Privileged instruction

Table 5-14 I/O Instructions (Sheet 2 of 2)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
5.8.2.3	IOLD	Output Address and Range	Output address and range to channel	I set if accepted	IMO and REG illegal for AAS* (TV16), Privileged instruction

* for CR41E exception see IOLD instruction description.

Table 5-15 Numerical Representation of Word 1 of I/O Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE			
						AAS	DAS	CAS	RAS
5.8.3.1	8	0	0+m	n	IO	—	Word	Word	—
5.8.3.2	8	1	0+m	n	IOH	—	Byte	Word	—
5.8.3.3	8	1	8+m	n	IOLD	Byte	—	Word	Word

where m, n = map coordinates.

5.8.1 I/O Instruction Execution

Execution of I/O instructions requires ring 0 or 1 privilege.

During I/O instruction execution, the following criteria determines which functions are to be performed by the software and which by the hardware:

- o It is the software's responsibility to insure that the buffer used during the execution of an IOLD instruction be in memory.
- o It is the hardware's responsibility to perform the following access right checks:
  - For IO or IOH, check that:
    1. The control word can be read by the process.
    2. Data being sent to the device controller can be read by the process or the data being received from the device controller can be written in to the specified location by the process.

- For IOLD , check that:

  1. The control word can be read by the process.

  2. The range word can be read by the process.

  3. The first word of the data buffer can be read or written (as applicable) by the process.

5.8.2 I/O Instruction Description

5.8.2.1 WORD INPUT/OUTPUT, IO

Format:

IO

Description:

1. If F is Odd (indicating Output): Send the control word (CH, F) and the data word specified by the DAS to the addressed I/O channel.

2. If F is Even (indicating Input): Send the control word (CH, F) to the addressed I/O channel. If the channel accepts the command, receive a word response from the channel and store it in the word location specified by the DAS. If the channel does not accept the command, do not change the contents of that location.

In both cases above, if the I/O channel accepted the command, set the I-indicator bit, else clear it.

Operation:

1. F is Odd: L6 Bus <— CH, F; L6 Bus <— [DEA], where DEA is the Data Effective Address and is obtained from the DAS.

2. F Even: L6 Bus <— CH, F; if ACK, then [DEA] <— Response from I/O channel.

Indicator Conditions:

	C	Unchanged
	B	Unchanged
If addressed channel accepted the command then	I	<—1 else I <—0
	G	Unchanged
	L	Unchanged
	U	Unchanged
	OV	Unchanged

Special Conditions:

1. In the DAS, IMO usage is valid if F is odd (indicating output). If F is even, IMO usage can cause alteration of procedure.

2. In the CAS, IMO usage is valid.

3. Privileged instruction. If not in ring 0 or 1, then Trap TV13 in lieu of execution.

4. If the addressed controller does not answer, then Trap TV15.

5.8.2.2 HALFWORD (BYTE) INPUT/OUTPUT, IOH

Format:

IO

Description:

1. If F is Odd (indicating Output): Send the control word (CH, F) and the byte specified by the DAS to the addressed I/O channel.

2. If F is Even (indicating Input): Send the control word (CH, F) to the addressed I/O channel. If the channel accepts the command, receive a byte response from the channel and store it in the byte location specified by the DAS. If the channel does not accept the command, do not change the contents of that location.

In both cases above, if the I/O channel accepted the command, set the I-indicator bit, else clear it.

Operation:

1. F is Odd: L6 Bus <— CH, F; L6 Bus <— [DEA] where DEA is the Data Effective Address and is obtained from the DAS.

2. F is Even: L6 Bus <— CH, F; If ACK, then [DEA] <— Response from I/O channel.

Indicator Conditions:

	C	Unchanged
	B	Unchanged
If addressed channel accepted the command then	I	<—1 else I <—0
	G	Unchanged
	L	Unchanged
	U	Unchanged
	OV	Unchanged

Special conditions:

1. The byte specified by the DAS is as described in the LDH instruction sheet.

2. IMO usage as specified in I/O instruction.

3. Privileged instruction. If not in ring 0 or 1, then Trap TV13 in lieu of execution.

4. If the addressed controller does not answer, then Trap TV15.

5.8.3.3 OUTPUT ADDRESS AND RANGE, IOLD

Format:

IO

Description:

1. Send the Control word (CH, F) and the physical address derived from AAS to the addressed I/O channel.

2. If the channel accepts the above command, send the same function code incremented by 4 and the data item specified by RAS to the same I/O channel.

If the I/O channel accepted both of the above, set the I-bit of the indicator register, else clear it.

Operation:

```
1. First Bus Cycle
   L6 Bus <-- CH, F
   L6 Bus <-- AEA, where AEA is the effective address derived from AAS.

2. Second Bus Cycle
   L6 Bus <-- CH, F+4
   L6 Bus <-- [REA], where REA is the effective address derived from RAS.
```

Indicator Conditions:

```
                                  C  Unchanged
                                  B  Unchanged
If addressed channel
accepted both commands then       I  <--1 else I <--0
                                  G  Unchanged
                                  L  Unchanged
                                  U  Unchanged
                                  OV Unchanged
```

Special Conditions:

1. AS:

   a. IMO usage for AAS causes a Trap TV16 except in a CR41E where results are unspecified.
   b. REG usage for AAS causes a Trap TV16.
   c. Push and pop type ASs (e.g., ↓Bn, Bn↑) will cause unspecified results in a CR41E.

2. All addresses on the I/O bus are physical addresses.

3. Privileged instruction. If not in ring 0 or 1, then Trap TV13 in lieu of execution.

4. If the addressed controller does not answer, then Trap TV15.

5.9 GENERICS

A generic instruction (GE) has the following format:

```
0               7 8             15
----------------------------------
| 0 0 0 0 0 0 0 0 |       OP      |
|--------------------------------|
|   Additional words as needed   |
----------------------------------
```

where OP = Opcode:

$0 \leq OP \leq 127$ or $00 \leq OP \leq 7F$ - Standard or reserved for future HIS use;

$128 \leq OP \leq 256$ or $80 \leq OP \leq FF$ - RFU (user defined).

Within the first group, the types of generic instructions available are summarized in Table 5-16. Their numerical representation is given in Table 5-17.

Table 5-16 Generic Instructions (Sheet 1 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS CHANGED	COMMENTS
5.9.1	HLT	Halt Program Execution	See text		Privileged instruction
5.9.2	MCL	Call Monitor via Trap	TV01		
5.9.3	BRK	Breakpoint	TV02		
5.9.4	RTT	Return from Trap	[I], [R3], [B3], [P] <-- [TSA]; S.RN <-- [Z(8,9)]	Indicator register is restored from [TSA.I]	If [TSAP] = NULL then Trap TV16.
5.9.5	RTCN	Real-Time Clock On	See text		Privileged instruction
5.9.6	RTCF	Real-Time Clock Off	See text		Privileged instruction
5.9.7	WDTN	Watchdog Timer On	See text		Privileged instruction

Table 5-16 Generic Instructions (Sheet 2 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS CHANGED	COMMENTS
5.9.8	WDTF	Watchdog Timer Off	See text		Privileged instruction
5.9.9	MMM	Memory-to-Memory Move	See text		Interruptable
5.9.10	ASD	Activate Segment Descriptor	See text		Privileged instruction
5.9.11	VLD	Validate Access Rights	See text		
5.9.12	CVP	Convert Vir-tual to Phys-ical Address	See text		Not supported by CR41E, M5X
5.9.13	QOH	Queue on Head	See text	C	Interruptable
5.9.14	QOT	Queue on Tail	See text	C	Interruptable
5.9.15	DQH	Dequeue from Head	See text		Interruptable
5.9.16	DQA	Dequeue by Address	See text	C,G,L	Interruptable
5.9.17	SQH	Search Queue from Head	See text	C,G,L	Interruptable Not supported by CR41E, M5X, M5XE
5.9.18	SQA	Search Queue by Address	See text	C,G,L	Interruptable Not supported by CR41E, M5X, M5XE
5.9.19	SRDB	Store Remote Descriptor Base	[B3] <-- [RDBR]		
5.9.20	LRDB	Load Remote Descriptor Base	[RDBR] <-- [B3]		
5.9.21	LDT	Load Stack Address Register	[T] <-- [Bn]		
5.9.22	STT	Store Stack Register	[B7] <-- [T]		

Table 5-16 Generic Instructions (Sheet 3 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS CHANGED	COMMENTS
5.9.23	ACQ	Acquire Stack Space	See text		
5.9.24	RLQ	Relinquish Stack Space	See text		
5.9.25	MFL	Modify Frame Length	See text		Not supported by CR41E, M5X, M5XE
5.9.26	BSRCH	Bit String Search	See text	B	Not supported by CR41E, M5X, M5XE
5.9.27	MEMD	Main Memory Diagnostic	See text		Privileged instruction
5.9.28	MMUD	Memory Management Unit Diagnostic	See text		Privileged instruction Not supported by CR41E, M5X
5.9.29	ASST	Activate System Segment Table	See text		Privileged instruction Not supported by CR41E, M5X, M6X
5.9.30	ATST	Activate Task Segment Table	See text		Privileged instruction Not supported by CR41E, M5X, M6X
5.9.31	WGT	Who Goes There	See text		Privileged instruction
5.9.32	DHR	Dump H/W Registers	See text		Privileged instruction Not supported by CR41E, M5X, M5XE
5.9.33	RSC	Rescan Configuration	See text		Privileged instruction
5.9.34	CPID	CPU Self Identification	See text		Privileged instruction Not supported by M5X, M6X, M6XE

Table 5-17 Numerical Representation of Generic Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
5.9.1	0	0	0	0	HLT	
5.9.2	0	0	0	1	MCL	
5.9.3	0	0	0	2	BRK	
5.9.4	0	0	0	3	RTT	NOT
5.9.5	0	0	0	4	RTCN	
5.9.6	0	0	0	5	RTCF	
5.9.7	0	0	0	6	WDTN	
5.9.8	0	0	0	7	WDTF	
5.9.9	0	0	0	8	MMM	APPLI-
5.9.31	0	0	0	9	WGT	
5.9.10	0	0	0	A	ASD	
5.9.11	0	0	0	B	VLD	
5.9.20	0	0	0	C	LRDB	
5.9.19	0	0	0	D	SRDB	CABLE
5.9.27	0	0	0	E	MEMD	
5.9.32	0	0	0	F	DHR 1, 3	
	0	0	1	0	Stack	
5.9.22	0	0	0	0	STT 6,	
5.9.24	0	0	0	n	RLQ 6,	
5.9.21	0	0	m	0	LDT 6,	
5.9.23	0	0	m	n	ACQ 6,	
5.9.25	0	0	8+m	0	MFL 6, 1, 3	
5.9.33	0	0	1	1	RSC	
5.9.26	0	0	1	2	BSRCH 1, 3	
5.9.34	0	0	1	7	CPID 2, 4	
5.9.29	0	0	1	9	ASST 1, 5	
5.9.30	0	0	1	A	ATST 1, 5	
5.9.12	0	0	1	B	CVP 1	
5.9.28	0	0	1	E	MMUD 1	
5.9.16	0	0	6	0	DQA	
5.9.14	0	0	6	1	QOT	
5.9.15	0	0	6	2	DQH	
5.9.13	0	0	6	3	QOH	
5.9.18	0	0	6	4	SQA 1, 3	
5.9.17	0	0	6	6	SQH 1, 3	

Notes

1. Not supported by CR41E, M5X. If used, then post a Trap TV05.
2. Not supported by M5X. If used, then post a Trap TV05.
3. Not supported by M5XE. If used, then post a Trap TV05.
4. Not supported by M6X, M6XE. If used, then post a Trap TV05.
5. Not supported by M6X. If used, then post a Trap TV05.
6. The numerical representation given is for the second word of the instructio which determines the instruction type.

5.9.1 Stop Program Execution, HLT

Format:

GE

Description:

Stop Program Execution. The CSS enters the Procedure Halt state. This state is indicated through the SCF. In this state no instructions are executed but all interrupts are honored. If the operator depresses the Execute key on the SCF while the CSS is in this state then the CSS skips the HLT, goes to the next instruction and resumes instruction execution.

Operation:

As above

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Privileged instruction. If not in ring 0 or 1, then Trap TV13 in lieu of execution.

5.9.2 Call Monitor, MCL

Format:

GE

Description:

Initiate a Trap TV01.

Operation:

A Trap TV01 is generated.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged

L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

See subsection 3.6.

5.9.3 Breakpoint, BRK

Format:

GE

Description:

Initiate a Trap TV02.

Operation:

A Trap TV02 is generated.

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

Used for debugging. See subsection 3.6.

5.9.4 Return From Trap, RTT

Format:

GE

Description:

The current level ISA is used to access the Trap Save Area (TSA) from which all saved visible registers are restored. The privilege state is restored from the Saved Privilege field (Z.RN) in the TSA. The TSA block is returned to the TSA memory pool. Refer to Figure 5-3.

Operation:

```
[I], [R3], [B3], [P], [S.RN] <-- [TSA]
```

Indicator Conditions:

C <-- [TSA]
B <-- [TSA]
I <-- [TSA]
G <-- [TSA]
L <-- [TSA]
U <-- [TSA]
OV<-- [TSA]

Special Conditions:

If the Trap Save Area Pointer (TSAP) is NULL, then Trap TV16 in lieu of execution. Refer to subsection 3.6.

5.9.5 Real-Time Clock On, RTCN

Format:

GE

Description:

Enable Real-Time Clock (RTC). Refer to subsection 3.7.

Operation:

As above

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Privileged instruction. If not in ring 0 or 1 , then Trap TV13 in lieu of execution.

5.9.6 Real-Time Clock Off, RTCF

Format:

GE

```
                   *******
                  * START *
                   *******
                      |
                      v
        ___________________________________
       (                                   )
       (   FETCH THE  RTT INSTRUCTION      )
       (___________________________________)
                      |
                      v
        __________________________________
        |   USING THE CPU # AND THE       |
        |   CURRENT LEVEL #, COMPUTE      |
        |_THE VA OF THE IV AND FETCH IT_  |
                       |
                       v
               ___________________
              /      WAS A        \  YES          ************
             <       TRAP          >-------->*  HALT  OR  *
              \__ENCOUNTERED ?____/          * SUPER HALT *
                       | NO                   ************
                       v                           ^
               ___________________                 |
              /                   \  YES           |
             <  IS [IV] = NULL  ?  >-------------->|
              \___________________/                |
                       | NO                        |
                       v                           |
             ______________________                |
             | USING IV, COMPUTE THE |             |
             |  VA OF TSAP (IV-2)    |             |
             |___AND FETCH IT________|             |
                       |                           |
                       v                           |
               ___________________                 |
              /      WAS A        \  YES           |
             <       TRAP          >---------------
              \___ENCOUNTERED ?___/
                       | NO
                       v                        **********
               ___________________             * TRAP #16 *
              /                   \  YES       *  PROGRAM   *
             < IS [TSAP] = NULL  ? >------->*   ERROR    *
              \___________________/             **********
                       | NO
                       v
                      o o
                     ( A )
                      o o
```

Figure 5-3 RTT Instruction Flow Chart (Sheet 1 of 3)

```
                         o o
                        ( A )
                         o o
                          |
                          v
               _________________________
              |                         |
              | DETERMINE THE NATSAP #  |
              |_________________________|
                          |
                          v
             ______________________________
            |  USING THE CPU # AND THE     |
            |  NATSAP #, COMPUTE THE VA    |
            |_OF THE NATSAP AND FETCH IT___|
                          |
                          v
                 __________________
               /       WAS A        \  YES      ************
              <        TRAP          >------->*  HALT  OR  *
               \__ENCOUNTERED ?_____/         * SUPER HALT *
                          | NO                 ************
                          v                         ^
         _________________________________          |
        | UNLINK THE TSA FROM THE ISA AND |         |
        |  RETURN IT TO THE APPROPRIATE   |         |
        |      TSA POOL AS FOLLOWS:       |         |
        |                                 |         |
        |       [NATSAP] --> TEMP         |         |
        |       [TSAP]   --> NATSAP       |         |
        |       [TSAL]   --> TSAP         |         |
        |_______[TEMP]___--> TSAL_________|         |
                          |                         |
                          v                         |
                 __________________                 |
               /       WAS A        \  YES          |
              <        TRAP          >--------------
               \__ENCOUNTERED ?_____/
                          | NO
                          v
                         o o
                        ( B )
                         o o
```

Figure 5-3 RTT Instruction Flow Chart (Sheet 2 of 3)

```
                       o o
                      ( B )
                       o o
                        |
                        v
          ______________________________
         (                              )
         ( RESTORE RTT INSTRUCTION CONTEXT )
         (______________________________)
                        |
                        v
          ______________________________
         |         FETCH FROM TSA:        |
         | o  COPY OF I    o  RETURN ADD. |
         | o  B3           o  Z WORD      |
         |_o__R3__________________________|
                        |
                        v                       ************
                 _____________                  *  HALT  OR  *
                /    WAS A     \   YES          * SUPER HALT *
               < TRAP DETECTED >--------->*     ************
                \______?_______/
                        | NO
                        v
            ___________________________
           |                           |
           | RETURN ADDRESS    --> EA  |
           |_RING # IN Z WORD --> REF__|
                        |
                        v
             ________________________
            |   RESTORE SAVED RTT    |
            | INSTRUCTION CONTEXT IN |
            |   o  I         o  B3   |
            |___o__R3________________|
                        |
                        v
                 ______________
                |              |
                |_REF --> S.RN_|
                        |
                        v
               __________________
              | EA.SN  --> P.SN  |
              |_EA.DSP --> P.DSP_|
                        |
                        v
             ________________________
            | START PROCEDURE BEING  |
            | RETURNED TO, WITH PRI- |
            |_VILEGE DEFINED BY S.RN_|
                        |
                        v
                      *****
                      * END *
                      *****
```

Figure 5-3 RTT Instruction Flow Chart (Sheet 3 of 3)

Description:

Disable the Real-Time clock. Refer to subsection 3.7.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Privileged Instruction. If not in ring 0 or 1, then Trap TV13 in lieu of execution.

5.9.7 Watchdog Timer On, WDTN

Format:

GE

Description:

Enabled.the Watchdog Timer. Refer to subsection 3.7.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Privileged Instruction. If not in ring 0 or 1, then Trap TV13 in lieu of execution.

5.9.8 Watchdog Timer Off, WDTF

Format:

GE

Description:

Disable Watchdog Timer (WDT). Refer to subsection 3.7.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Privileged Instruction. If not in ring 0 or 1, then Trap TV13 in lieu of execution.

5.9.9 Memory To Memory Move, MMM

Format:

GE

Description:

n bytes of memory are moved from a sending field to a receiving field. The address of the first byte of the sending field is identified by [B2] + [R2], where R2 contains a signed byte displacement from [B2]. The first byte location of the receiving field is identified by [B3] + [R3], where R3 contains a signed byte displacement from [B3]. The number of bytes to be moved is contained in R6, where $1 \leq [R6] \leq 64K$.

Operation:

If [R6] = 0 then treat instruction as a NOP

Post a Trap TV16 if:
- [R6(0)] = 1 or
- [R2] + [R6] or [R3] + [R6] > 2^{15} -1 (overflow in R2 and/or R3)

Else repeat [R6] <-- [R6] - 1; [[B3] + [R3↑]] <-- [[B2] + [R2↑]] until [R6] = 0. Move is from left to right. Note also that R2, R3 and R6 are modified during instruction execution.

While performing the move post a Trap TV15 if [B2] + [R2] or [B3] + [R3] exceed segment size.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. MMM is interruptable.

2. Partially overlapped fields will cause unspecified results.

5.9.10 Activate Segment Descriptor, ASD

Format:

GE

Description:

Register B5 contains an EA which, in its segment field, specifies a segment descriptor entry in the MMU. Registers R6 and R7 contain the descriptor first and second words, respectively.

The results of ASD execution are dependent upon the MMU mode of operation (SMMU or EMMU) and the target address space (System or Task) as follows:

o SMMU mode ([B5] = 00XXXX through 0FXXXX) or EMMU mode, system address space ([B5] = 00XXXX through 7FXXXX), where X = dont care;

  The contents of R6 and R7 are copied into the specified MMU entry.

MMU RAM_{SD} <--- [R6], [R7]

o EMMU mode, Task address space ([B5] = 80XXXX through FFXXXX);

The SD specified by [B5] is marked "not present" in the EMMU Storage Array. Subsequent references to this descriptor causes it to be On-Demand loaded from the Task Segment Table pointed to by the ASV.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. Privileged Instruction. If not in ring 0 or 1, then Trap TV13 in lieu of execution. In a CR41E, M6X and M6XE, ASD may be executed only in ring 0.

2. In SMMU Mode (applicable to M5XE or M6XE only), if Segment number is greater than 0F ([B5] > 0FXXXX) then Trap (TV16).

3. Segment 0 cannot be invalidated in an M5XE, M6X or M6XE in any Mode, else Trap TV16.

4. The valid bit and base field of SD# 0.0 (in SMMU Mode) cannot be modified in a CR41E or M5X. No trap is posted.

5.9.11 Validate, VLD

Format:

GE

Description:

The VLD instruction allows a called procedure to determine if the operand being passed to it by a calling procedure would have been accessible to the calling procedure, and

o for the case where the operand is within one segment,

- the operand is within a valid segment;

o For the case where the operand spans two segments

- the operand is within two consecutive valid segments,
- the access right checks apply to both segments.

The VLD instruction's parameters are in dedicated registers.

1. The operand pointer is specified in B5.

2. The range (r) of the operand is specified in R3 and consists of a positive value in bytes where $0 \leq r \leq 2^{16}-1$.

   If the range equals zero, then the resultant value returned in R3 is model dependent.

3. The ring value given for validation (reflecting the calling procedure's execute priviledge) is specified in bits 1 and 2 of R5.

4. R3 is returned with an indication of the results:

R3 Contents	Meaning
-1	The segment number, in a M5XE or M6XE, is out of bounds (i.e., the segment number in B5 is greater than OF in SMMU mode or the segment number in B5 is greater than TSTL in EMMU mode), the segment descriptor is invalid or the operand is out of segment bounds
-2	No read access is permitted.
0	Read access is permitted, Write access is not permitted
+2	Read/Write access is permitted

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

5.9.12 Convert Virtual To Physical Address, CVP

Format:

GE

Description:

The CVP instruction converts a Logical Address (LA) into a Physical Address (PA) and performs a validate function.

Its parameters are in dedicated registers.

- o B5 supplies the LA to be converted.
- o K5 receives the PA in the M6XE; B6 receives the PA in the M5XE.
- o R3 supplies the range (a positive nonzero value in bytes) for use by the validate operation.
- o R5 bits 1 and 2 supply the effective ring value for use by the validate operation.

R3 is returned with indication of the results as follows:

Invalid LA

PA is not returned in this case, K5/B6 is undefined.

```
 0  1                                  15
 ----------------------------------------
 |  |                                   |
 |1 | X - - - - - - - - - - - - - - - X |  R3
 |  |                                   |
 ----------------------------------------
```

Invalid LA is posted if:

- o The Segment Descriptor is invalid
- o The offset specified by the LA is greater than the segment size

Valid LA

PA is returned in this case.

```
   0    1                               11  12   13   14   15
 -------------------------------------------------------------
 | 0 | 0- -  -  -  -  -  -  -  -  -  - 0| X  | R  | W  | E  | R3
 -------------------------------------------------------------
```

where for:

Bit 12 = 0:	Range is valid
Bit 12 = 1:	Range is invalid (too large or less than or equal to Zero)
Bit 13:	R = 1, Read access not permitted
Bit 14:	W = 1, Write access not permitted
Bit 15:	E = 1, Execute not permitted.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. If only a conversion from LA to PA is desired, then R3 and R5 need not be preloaded. The return code in R3 should be checked only for negative (invalid LA) and positive (valid LA) since the Validate operation takes place using undefined information. The return code in R3 bits 12 through 15 is thus unspecified.

2. In SMMU Mode (applicable to M5XE or M6XE only), if Segment number is greater than OF ([B5] > OFXXXX) then Trap (TV15).

3. The CVP instruction is supported only in the M5XE, M6X and M6XE models.

5.9.13 Queue On Head, QOH

Format:

GE

Description:

The QOH instruction links a new frame into the list before the first frame that has the same or numerically higher priority number, or as the last frame if no frame of equal or greater priority is found. B2 is used to point to the lock word of the list, and R5 contains the priority to be assigned to the new frame. The priority word will be loaded with the contents of R5. B1 points to the priority word of the frame to be added. The pointer following the priority word is loaded with the correct frame pointer. Reference subsection 3.14.

Operation:

As above.

Indicator Conditions:

```
Operation completed then C  <-- 1 else C <-- 0
                            (queue was locked)
                         B  Unchanged
                         I  Unchanged
                         G  Unchanged
                         L  Unchanged
                         U  Unchanged
                         OV Unchanged
```

Special Conditions:

1. QOH is interruptable.

2. The Lock word is addressed with a read-modify-write cycle.

5.9.14 Queue On Tail, QOT

Format:

GE

Description:

The QOT instruction links a new frame into the list before the first frame that has a numerically higher priority number, or as the last frame if no frame of greater priority is found.

B1, B2 and R5 are used the same as in the QOH instruction.

The hardware assumes a monotonic queue structure. The following examples best illustrate the point.

a. QOT a frame with priority 3 into a monotonic queue structure:

```
Frame Number -      1 2 3 4 5 6   7 8 9 10 11
Priority Number -   1 1 1 2 3 3 3 4 6 7 7 8
```

Note that the priority numbers of frames in the queue are linked in such a way that the priority number of frame n+1 is never less than that of frame n.

b. QOT a frame with priority 3 into a nonmonotonic queue structure:

```
Frame Number -      1 2 3 4 5 6   7 8 9 10 11 12
Priority Number -   1 1 1 2 3 3 3 4 3 4 2 7 8
```

Note that the frame is inserted between frames 6 and 7 rather than frames 8 and 9 since scanning takes place from queue head to queue tail.

Reference subsection 3.14.

Operation:

As above.

Indicator Conditions:

```
 Operation completed then C  <-- 1 else C <-- 0
                             (queue was unlocked)
                          B  Unchanged
                          I  Unchanged
                          G  Unchanged
                          L  Unchanged
                          U  Unchanged
                          OV Unchanged
```

Special Conditions:

1. QOT is interruptable.

2. The Lock word is addressed with a read-modify-write cycle.

5.9.15 Dequeue From Head, DQH

Format:

GE

Description:

The DQH instruction unlinks the first frame from the list whose priority value equals or is numerically greater then the contents of R5. B2 points to the lock word, and B1 is returned containing a pointer to the priority word of the unlinked frame. I(G) and I(L) indicate the hit conditions.

I(G)	I(L)	Condition
0	0	Unlinked frame was first whose priority equaled [R5]
1	0	Unlinked frame was first whose priority exceeded [R5]
0	1	No frame was unlinked, B1 is unchanged, no priority found equal to or greater than [R5]

Reference subsection 3.14.

Operation:

As above.

Indicator Conditions:

```
 Operation completed then C  <-- 1 else C <-- 0
                             (queue was locked)
                          B  Unchanged
                          I  Unchanged
                          G  Unchanged
                          L  Unchanged
                          U  Unchanged
                          OV Unchanged
```

Special Conditions:

1. DQH is interruptable.

2. The Lock word is addressed with a read-modify-write cycle.

5.9.16 Dequeue On Address, DQA

Format:

GE

Description:

The DQA instruction unlinks a frame whose priority word address exactly matches B1. B2 points to the lock word.

Reference subsection 3.14.

Operation:

As above.

Indicator Conditions:

```
 Operation completed then   C  <-- 1 else C <-- 0
                             (queue was locked)
                            B  Unchanged
                            I  Unchanged
                            G  <-- 0
 If frame was unlinked then L  <-- 0 else L <-- 1
                            U  Unchanged
                            OV Unchanged
```

Special Conditions:

1. DQA is interruptable.

2. The Lock word is addressed with a read-modify-write cycle.

5.9.17 Search Queue From Head, SQH

Format:

GE

Description:

The SQH instruction searches for the first frame in the queue whose priority value equals or is numerically greater than the contents of [R5]. [B2] points to the lock word. The indicator I(C), I(G), and I(L) report the outcome of the search as follows:

I(C)	I(G)	I(L)	
0	UC*	UC*	Queue was locked. No search was performed.
1	0	0	A frame was found whose priority equaled [R5]. Queue remains locked.
1	1	0	A frame was found whose priority exceeded [R5]. Queue remains locked.
1	0	1	No frame was found with priority equal to or greater than [R5]. [B1] is unchanged, and the queue remains locked.

If a suitable frame is found, [B1] is returned containing a pointer to the priority word of the frame found; the found frame is not unlinked.

Refer to subsection 3.14.

Operation:

Search takes place only if the queue is unlocked; i.e.,

- o If locked, then I(C) <-- 0 and instruction is terminated
- o If unlocked, then I(C) <-- 1 lock queue and instruction proceeds as described above.

Indicator Conditions:

C	As above
B	Unchanged
I	Unchanged
G	As above
L	As above
U	Unchanged
OV	Unchanged

* UC = Unchanged

Special conditions:

1. SQH is interruptable.

2. The Lock word is addressed with a read-modify-write cycle.

3. The SQH instruction is supported only in the M6X and M6XE.

5.9.18 Search Queue By Address, SQA

Format

'GE

Description:

The SQA instruction searches the queue for a frame whose priority word address exactly matches [B1]. [B2] points to the lock word. The indicators I(C), I(G), and I(L) report the outcome of the search as follows:

I(C)	I(G)	I(L)	
0	UC*	UC*	Queue was locked. No search was performed.
1	0	0	Frame was found. Queue remains locked.
1	0	1	Frame was not found. Queue remains locked.

If a suitable frame is found, the frame is not unlinked.
Refer to subsection 3.15.

Operation:

Search takes place only if the queue is unlocked; i.e.,

- o If locked, then I(C) <-- 0 and instruction is terminated

- o If unlocked, then I(C) <-- 1 lock queue and instruction proceeds as described above.

Indicator Conditions:

C	As above
B	Unchanged
I	Unchanged
G	As above
L	As above
U	Unchanged
OV	Unchanged

*UC = Unchanged

Special Conditions:

1. SQA is interruptable.

2. The Lock word is addressed with a read-modify-write cycle.

3. The SQA instruction is supported only in the M6X and M6XE.

5.9.19 Store Remote Descriptor Base Register, SRDB

Format:

GE

Description:

The contents of the Remote Descriptor Base Register (RDBR) are stored in B3.

Operation:

[B3] <-- [RDBR]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

5.9.20 Load Remote Descriptor Base Register, LRDB

Format:

GE

Description:

The contents of B3 are loaded into the Remote Descriptor Base Register (RDBR).

Operation:

[RDBR] <-- [B3]

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

None.

5.9.21 Load Stack Address Register, LDT

Format:

GE

Description:

The contents of a specified base register, Bn, are stored into the stack address register, T. This instruction is distinguished from the other stack instructions by the contents of the second word of the instruction:

Second Word Format:

Bits 0:7	RFU
Bits 8, 12	Zero
Bits 9:11	Specifies Bn, $0 < n \leq 7$
Bits 13:15	Zero

Refer to subsection 3.4.

Operation:

[T] <— [Bn]

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

1. If bit 12 of the second word ≠ Zero, then Trap TV16 is posted.

2. If current privilege is not sufficient to read the new stack header, then Trap TV14 (M6X or M6XE only).

5.9.22 Store Stack Register, STT

Format:

GE

Description:

The contents of the stack address register, T, are stored into B7. The instruction is distinguished from the other stack instructions by the contents of the second word of the instruction:

Second Word Format:

Bits 0:7	RFU
Bits 8:15	Zero

Refer to subsection 3.4.

Operation:

```
[B7] <-- [T]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If bit 12 of the second word ≠ Zero, then Trap TV16.

5.9.23 Acquire Stack Space, ACQ

Format:

GE

Description:

The ACQ instruction acquires a portion of the remaining available stack space. The amount of space in words to be acquired is specified by Rm, and Bn is loaded with the left-most address of the newly acquired space. CW is updated to the new stack length. This instruction is distinguished from the other stack instructions by the contents of the second word of the instruction:

Second Word Format:

Bits 0:7	RFU
Bits 8, 12	Zero
Bits 9:11	Specifies Rm, $0 < m \leq 7$
Bits 13:15	Specifies Bn, $0 < n \leq 7$

where:

CW = Current stack length in words
MW = Maximum stack length in words

Refer to subsection 3.4.

Operation:

```
If MW < CW + [Rm] + 1, TV10 (Stack Overflow)
Else [Bn] <-- [T] - CW - [Rm]
[[Bn] - 1] <-- [Rm]
CW <-- CW + [Rm] + 1
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. If bit 12 of the second word ≠ Zero, then Trap TV16.

2. If T = NULL then Trap TV16.

5.9.24 Relinquish Stack Space, RLQ

Format:

GE

Description:

The RLQ instruction converts a consumed stack frame into available space by updating the current length in words (CW) in the stack header. Bn is loaded with the leftmost address of the frame at the top of the stack. This instruction is distinguished from the other stack instructions by the contents of the second word of the instruction.

Second Word Format:

Bits 0:7	RFU
Bits 8:12	Zero
Bits 13:15	Specifies Bn, $0 < n \leq 7$

Refer to subsection 3.4.

Operation:

```
If CW = 0, Trap TV16
Else
CW <-- CW - [[T] - CW] - 1
If 0 < CW, then [Bn] <-- [T] - CW + 1
Else if CW = 0, then Trap (TV09) Stack Underflow
Else Trap TV16
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. If bit 12 of the second word ≠ Zero, then Trap TV16.

2. If T = NULL then Trap TV16.

5.9.25 Modify Frame Length, MFL

Format:

GE

Description:

The MFL instruction allows the current active frame to be either extended ([Rm] > 0) or shortened ([Rm] < 0) by the magnitude of the contents of the register Rm. The range of [Rm] is $-2^{15} \leq [Rm] \leq 2^{15}-1$.

Refer to subsection 3.4.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. The stack header in memory must not be accessed by software following the execution of MFL instruction or usage of a stack related address syllable since the CSS does not update the header in memory but rather hidden CSS registers only.

2. The stack header in main memory may be accessed by the software after it is made current by the CSS following a reload of T.

3. If [Rm] < 0 and the absolute value of [Rm] is greater than the size of the active frame, then a Trap TV16 (program error) occurs.

4. If [Rm] = 0, the instruction is equivalent to a NOP.

5. If [Rm] < 0 and the absolute value of [Rm] is equal to the size of the active frame, then the frame is reduced to Zero words. The frame remains the active frame.

6. If [Rm] > 0 and sufficient space is not available in the current stack area, then Trap TV10 (stack overflow).

7. The MFL instruction is supported only in the M6X and M6XE.

5.9.26 Bit String Search, BSRCH

Format:

GE

Description:

- o The BSRCH instruction searches a string for a search argument consisting of n all Zeros or n all Ones. If a match is found, then I(B) <-- 1 and R2 is incremented and R6 is decremented by the number of bits preceding the beginning of the match (B2 + R2 points to the beginning of the match).

- If no match is found, then I(B) <-- 0; R2 and R6 are updated as a function of whether a partial match is found or not. For no partial match found then R2 is incremented by the value of R6, and R6 is cleared to 0. For partial match found then R2 is incremented by x such that it points to the beginning of the last partial match and R6 is decremented by x.

- [B2] + [R2] points to a bit string, with [R2] being a 16-bit signed bit index.

- [R6] = size of string in bits, where $0 \leq [R6] \leq 2^{15}-1$.

- [R7(0)] = 0 specifies the search argument is n Zeros.

- [R7(0)] = 1 specifies the search argument is n Ones.

- [R7(1:7)] = RFU.

- [R7(8:15)] = size n of the search argument in bits where $0 \leq n \leq 255$.

Operation:

As above.

Indicator Conditions:

	C	Unchanged
If match then	B	<-- 1 else B <-- 0
	I	Unchanged
	G	Unchanged
	L	Unchanged
	U	Unchanged
	OV	Unchanged

Special Conditions:

1. If [R6] = 0, I(B) is cleared and exit.
2. If [R6(0)] = 1, then Trap TV16, negative string size.
3. If [R6] + [R2] > $2^{15}-1$, then Trap TV16 (i.e., B2 would have to be incremented when R2 reaches FFFF, to continue addressing the bit string).
4. If [R7(8:15)] = 0, I(B) is set and exit.
5. The BSRCH instruction is supported only in the M6X and M6XE.

5.9.27 Main Memory Diagnostic, MEMD

Format:

GE

Description:

The MEMD instruction selects a memory controller, passes it a memory command, and stores the memory controller's response.

NOTES

1. The implementation of this instruction is model dependent.

2. Memory controller responses to the MEMD are memory controller dependent.

3. Since base registers vary in width among the various models, the Megabus nomenclature, will also be given in describing the MEMD commands. In Megabus nomenclature, BSAD19 through BSAD22 correspond to the least significant (rightmost) hexadecimal digit of the logical address in a base register.

4. An input memory ID command should always be the first command used in order to identify the memory controller type.

For more information on the MEMD, refer to Appendix A in this EPS-1.

Operation:

o Model CR41E

Its parameters are in dedicated registers as follows:

o B5 contains a logical address which, after being converted into a physical address, provides the memory diagnostic command code.

o B6 contains a logical address which, after being converted into a physical address, defines the memory location to be read. B6 is only used by command code 2 in B5.

o R6 is used to receive memory data or status as a function of the memory diagnostic command.

The supported commands are given in Table 5-18.

o Model M5X and M5XE

Its parameters are in dedicated registers as follows:

o B5 contains a logical address which, after being converted into a physical address, defines the memory controller to be addressed and provides the memory diagnostic command code.

o R6 is used to receive memory data or status as a function of the memory diagnostic command.

o Model M6X and M6XE

Its parameters are in dedicated registers as follows:

o B5 contains a logical address which, after being converted into a physical address, defines the memory controller to be addressed. It also provides the memory diagnostic command code if [B5] is even.

- o K6 contains the memory diagnostic command which is sent to the memory controller if [B5] is odd.
- o R6 is used to receive memory data or status as a function of the memory diagnostic command.
- o K5 is used in some command sequences to supply the data to be written into memory.

The supported commands are given in Table 5-18 if [B5] is even and in Table 5-19 if [B5] is odd.

Indicator conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Privileged instruction. If not in ring 0, then Trap TV13 in lieu of execution.

Table 5-18 [B5] Functionality (Sheet 1 of 3)
([B5] is an even word address)

BSAD 19-22	F U N C T I O N
[B5] (16-19)	CR41E
0 0 0 0	R6 <-- Mem. ID/Status
0 0 1 0	Clear Mem. ID/Status; Read contents of location addressed by [B5]; R6 <-- Mem. ID/Status
0 1 0 0	Set EDAC Mode
0 1 1 0	Reset EDAC Mode
X X X 1	TV16
1 X X X	TV16
[B5] (19-22)*	M5X and M5XE
0 0 0 X	R6 <-- Mem. ID/Status of Mem. contr. specified by [B5].
0 0 1 X	RFU

Table 5-18 [B5] Functionality (Sheet 2 of 3)
([B5] is an even word address)

BSAD 19-22	F U N C T I O N
[B5] (19-22)*	M5X and M5XE (cont.)
0 1 0 X	Set EDAC mode in Mem. controller specified by [B5]. R6 <-- contents of location addressed by [B5].
0 1 1 X	Reset EDAC mode in Mem. controller specified by [B5]. R6 <-- contents of location addressed by [B5].
1 0 0 X	RFU
1 0 1 X	Set/Clear "Here-I-Am" indicator, per BSAD15, in Mem. controller specified by [B5].

1 1 0 X	Set/Reset Alpha refresh control, per BSAD14-15:
	BSAD14-15 = 00 Alpha refresh on ; normal cycle
	BSAD14-15 = 01 Alpha refresh off; normal cycle
	BSAD14-15 = 10 Alpha refresh on ; high speed cycle
	BSAD14-15 = 11 Alpha refresh off; high speed cycle

1 1 1 X	RFU
[B5] (19-22)*	M6X and M6XE
0 0 0 0	R6 <-- Mem. ID/Status of Mem. contr. specified by [B5]. If configured Mem. contr. is a BF8MXG, then [B5(7, 16, 17)] may be used to select the ID word # as follows: B5(7) specifies Mem. contr. state; 0 = online and 1 = offline. B5(16, 17) select ID word #; 00 = word #0 11 = word #3 If B5(7, 16, 17) contain values not defined above then R6 is loaded with the content of the memory location specified in B5.
0 0 1 0	R6 <-- Mem. Status (from most recently detected error) of Mem. contr. specified by [B5].
0 1 0 0	TV16
0 1 1 0	TV16
1 0 0 0	Reconfigure Mem. controller, per BSAD13-15; R6 <-- contents of location addressed by [B5].

Table 5-18 [B5] Functionality (Sheet 3 of 3)
([B5] is an even word address)

BSAD 19-22	FUNCTION
[B5] (19-22)*	M6X and M6XE (cont.)
1 0 1 0	Set/Clear "Here-I-Am" indicator, per BSAD15, in Mem. controller specified by [B5].

1 1 0 0	Set/Reset Alpha refresh control, per BSAD14-15:
	BSAD14-15 = 00 Alpha refresh on ; normal cycle
	BSAD14-15 = 01 Alpha refresh off; normal cycle
	BSAD14-15 = 10 Alpha refresh on ; high speed cycle
	BSAD14-15 = 11 Alpha refresh off; high speed cycle

1 1 1 0	TV16
X X X 1	Command specified by K6.

* [B5] (16-19) for M5X and M6X.

Table 5-19 [K6] Functionality (Sheet 1 of 2)
([B5] is an odd word address)

K6	FUNCTION
00000000	R6 <-- Mem. ID of Mem. controller specified by [B5]
00000002	R6 <-- Status word of Mem. controller specified by [B5], containing most recently detected error. Clear status word of Mem. controller specified by [B5].
00000004	Clear Status word of Mem. controller specified by [B5]. K5 <-- double word specified by [[B5]-1]. R6 <-- Status word of Mem. controller specified by [B5]; i.e., status of double word just read.
00000006	Place Mem. controller specified by [B5] in EDAC mode. [K5(16:31)] --> word specified by [[B5]-1] with all EDAC bits =0. Reset EDAC mode in Mem. controller specified by[B5].
00000007	Place Mem. controller specified by [B5] in EDAC mode. [K5(16:31)] --> word specified by [B5] with all EDAC bits =0. Reset EDAC mode in Mem. controller specified by[B5].
0000XXX8	Reconfigure Memory per [K6].
0000000A	Turn OFF "Here-I-Am" Light in Mem. controller specified by[B5]. R6 <-- word specified by [[B5]-1].
0000008A	Turn ON "Here-I-Am" Light in Mem. controller specified by[B5]. R6 <-- word specified by [[B5]-1].
0000000C	Set Soft Error Rewrite ON, Normal Cycle in Mem. controller specified by[B5]. R6 <-- word specified by [[B5]-1].
0000008C	Set Soft Error Rewrite OFF, Normal Cycle in Mem. controller specified by[B5]. R6 <-- word specified by [[B5]-1].
0000010C	Set Soft Error Rewrite ON, High Speed in Mem. controller specified by[B5]. R6 <-- word specified by [[B5]-1].
0000018C	Set Soft Error Rewrite OFF, High Speed in Mem. controller specified by[B5]. R6 <-- word specified by [[B5]-1].

Table 5-19 [K6] Functionality (Sheet 2 of 2)
([B5] is an odd word address)

NOTES

1. K6 (0-21) and K6 (25-27) must be zero else TV16. Other unlisted codes are RFU and yield unspecified results.

2. K6 (22-23) are subject to modification by the MMU during address translation, as a function of bits 14 and 15 of the first word of the segment descriptor.

5.9.28 Memory Management Unit Diagnostic, MMUD

Format:

GE

Description:

This instruction is used to test the MMU and cache hardware. The register usage is as follows:

o R5 supplies, in bits 0 and 13 through 15, a four-bit function code that specifies which diagnostic action is to be performed. Bits 1 through 12 are currently ignored but should be made Zeros, since they are RFU. When the MMUD is executed in an M6X or M6XE, the content of R5 is replaced with the 16-bit "syndrome" of the most recent "disaster". When the MMUD is executed in an M5XE, the content of R5 is unchanged.

o R6 and R7 supply a 32-bit operand (left unchanged) for function codes 3, 5 and 7. R6 and R7 receive a 32-bit result for function codes 8000, 1, 2, 4, and 6 (initial values ignored).

o B5 supplies a virtual address for function codes 1, 2 and 3 (only the segment number field is used by function codes 0 and 3. B5 is ignored for function codes 8000, 4, 5, 6 and 7.

The content of B5 is unaltered by the MMUD instruction.

The MMU diagnostic instruction (MMUD) provides functionality to:

o Read/load a segment descriptor from/into the MMU SD storage facility.

o Convert a LA into PA.

o Read/load the MMU mode register (this register is an internal MMU register).

o Read/load the level register.

o Read/load TSTB and TSTL.

For more information on the MMUD, refer to Appendix B in this EPS-1.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. The MMUD instruction is supported only in the M5XE, M6X and M6XE.

2. Privileged instruction. If not in ring 0, then Trap TV13 in lieu of execution.

5.9.29 Activate System Segment Table, ASST

Format:

GE

Description:

The ASST instruction activates a new system address space and switches the MMU mode of operation between SMMU mode and EMMU mode, according to the contents of registers B5 and R6.

Operation:

o [B5] = Null

Activate SMMU mode of operation, generate a trivial address map (one for one mapping, where LAs = PAs) in the 31 Segment Descriptors. All subsequent references are made through the 16 small SDs and the 15 large SDs.

Note that R6 is ignored for this case.

o [B5] ≠ NULL

INRUSH the contents of the SST, which is pointed to by the LA contained in B5, starting at SD#00, up to and including the SD indicated in the System Segment Table Limit (SSTL) contained in R6. All invalid SDs within the range of SSTL and all SDs beyond SSTL are marked INVALID in the EMMU Storage Array. The Task Segment Table Limit (TSTL) register is set to 7F (no valid task address space assigned) and EMMU mode is activated. All subsequent references are mapped through the SDs in the extended mode of operation.

The format of R6, where 00 < SSTL $\leq$ 7F, is shown below:

```
0         8 9       15
--------------------
|  MBZ     |  SSTL  |
--------------------
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. Privileged instruction. If not in ring 0, then Trap (TV13) in lieu of execution.
2. R6(0:8) must be set to zero else unspecified results will occur.
3. The SST must start in a valid segment, else Trap (TV15). The SST is assumed to be contained in consecutive PA locations (no checks are performed).
4. An SST must be contained within the physical memory space, else Trap (TV15). State of the EMMU storage array, after this trap, is undefined.
5. The ASST instruction is supported only in the M5XE and M6XE.

5.9.30 Activate Task (User) Space Table (ATST)

Format:

GE

Description:

The ATST instruction activates a new task address space according to the contents of registers B5 and R6. This instruction is equivalent to an EMMU context switch with B5 supplying the ASV and R6 the TSTL.

Operation:

Convert the LA contained in B5 (which contains the pointer to the Task Segment Table (TST)) into a PA. Then load the computed PA and [R6] (which contains the Task Segment Table Limit (TSTL) value) into MMU registers. Subsequent references to the task address space will be made dynamically through on-demand loading of the MMU task cache.

Note that the TST must be memory resident whenever the task is a candidate for execution.

The format of R6, where 00 < TSTL $\leq$ FF is shown below:

```
0        7 8      15
--------------------
|  MBZ    |  TSTL  |
--------------------
```

Indicator Conditions:

C Unchanged
B Unchanged
I Unchanged
G Unchanged
L Unchanged
U Unchanged
OV Unchanged

Special Conditions:

1. Privileged instruction. If not in ring 0, then Trap (TV13) in lieu of execution.

2. If ATST is executed in SMMU mode, then Trap (TV16).

3. The TST must start in a valid segment, else Trap (TV15). The TST is assumed to be contained in consecutive PA locations (no checks are performed).

4. If TSTL is specified to be less than 7F ([R6] as well as the ISA contents), the Task Address Space is treated as unassigned, i.e., TSTL is set to 7F and subsequent references to the Task Address Space will cause a Trap (TV15).

5. The ATST instruction is supported only in the M5XE and M6XE.

6. The TST, in a M5XE, must be double word aligned else Trap (TV16).

5.9.31 Who Goes There, WGT

Format:

GE

Description:

The Firmware Revision Level of the processor is loaded (right justified) into R7.

Operation:

[R7] <-- CSS Firmware Revision Level

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None

5.9.32 Dump Hardware Registers, DHR

Format:

GE

Description:

The DHR instruction results in the CSS reading most of the CSS hardware registers and placing them in memory beginning at the memory location pointed to by the base register B1.

Operation:

CSS Registers --> Memory field; [B1] points to leftmost word of memory field.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. Privileged instruction. If not in ring 0, then Trap (TV13) in lieu of execution.

2. The DHR instruction is supported only in the M6X and M6XE.

5.9.33 Rescan Configuration, RSC

Format:

GE

Description:

For CR41E, M5X and M5XE:

The RSC instruction causes the contents of R7 to be transmitted on the Megabus to an optional processor whose channel number is contained in R6(0:9), with the function code to be sent contained in R6(10:15).

For M6X and M6XE:

TBD

This instruction is used in conjunction with the ability of the SIP, CIP, and cache to alter its channel number. RSC also forces the CSS to determine whether an SIP, CIP, or cache is currently assigned to it.

Operation:

As above.

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. Privileged instruction. If not in ring 0, then Trap (TV13) in lieu of execution.

2. The function code specified in R6(10:15) must be odd (I/O output).

5.9.34 CPU Self Identification, CPID

Format:

GE

Description:

The CPID instruction places the CPU identification code and optional information in registers R6 and R7.

The format of the values returned in registers R6 and R7 are as follows:

CPU	R 6		R 7
	BITS 0-7 CPU TYPE	BITS 8-15 MODIFIER	BITS 0-15 CPU SPECIFIC
CR41	01	00	0000
CR41E	01	00	0001
HERCULES	02	00	0000
HELIOS	03	00	0000
M5XE	05	00	0000

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

1. Privileged instruction. If not in ring 0, then Trap (TV13) in lieu of execution.

2. The CPID instruction is supported only in the CR41, CR41E and 5XE.

SECTION 6 EXTENDED INTEGER INSTRUCTION SET DEFINITION

6.1 EXTENDED INTEGER INSTRUCTION (EII) SET

This section describes the CSS EII instruction set. Refer to Table 1-2 for a list of the various conventions and definitions used to describe these instructions. These instructions are available on the M6X and M6XE only.

These instructions are classified as follows:

o Double Operand	- Subsection 6.2
o Single Operand	- Subsection 6.3
o Short Value Immediate	- Subsection 6.4
o Branch on Register	- Subsection 6.5
o Shift	- Subsection 6.6

This set of instructions provides the following functionality:

o Access to seven 32-bit integer registers, K(1:7)

o Usage of a data descriptor to define the Operand

o Use of mode register M2.

Mode register (M2) is defined to contain overflow masks for each of the seven K registers. See subsection 3.3.4.6

For convenience the instructions are summarized in Appendix F.

6.1.1 Data Descriptors

Data Descriptors (DD) are used to specify the type and location of an operand (second word of an EII).

Two types of DDs are defined:

o For all EII instructions <u>except</u> for Load Index and Address (LXA) instruction the following applies:

```
0        4 5        7  8   9     15
|          |          |      |        |  DATA
| BS LENGTH | DATA TYPE | MAP # |   AS   |  DESCRIPTOR
|__________|__________|______|________|
```

or

```
0     3 4           7  8   9     15
|      |              |      |        |  DATA
| R F U |   DATA TYPE   | MAP # |   AS   |  DESCRIPTOR
|______|______________|______|________|
```

Where:

- BS LENGTH specifies the length of the bit string when data type is Bit String. If data is Bit String and BS length equals Zero, then use contents of R1 (11:15). If [R1(11:15)] = 0, then the instruction is treated as a NOP.

- DATA TYPE specifies the atom size of the operand as defined in Table 6-1.

Table 6-1 EII Data Types (Sheet 1 of 2)

DATA TYPE BITS 4 5 6 7	MEMORY OPERAND	IMO	= Kn	REG
X 0 0 0	U Bit String	Illegal (TV16)	Illegal (TV16)	Illegal (TV16)
X 0 0 1	S Bit String	"	"	"
0 0 1 0	Illegal (TV16)	"	"	"
0 0 1 1	"	"	"	"
0 1 0 0	U Half Word	"	"	"
0 1 0 1	S Half Word	"	"	"

Table 6-1 EII Data Types (Sheet 2 of 2)

DATA TYPE BITS	MEMORY OPERAND	IMO	= Kn	REG
0 1 1 0	U Word	U Word	Illegal (TV16)	U Word
0 1 1 1	S Word	S Word	"	S Word
1 0 1 0	Illegal (TV16)	Illegal (TV16)	"	Illegal (TV16)
1 0 1 1	S Double Word	S Double Word	S Double Word	S Double Word
1 1 0 0	Illegal (TV16)	Illegal (TV16)	Illegal (TV16)	Illegal (TV16)
1 1 0 1	"	"	"	"
1 1 1 0	Address	Address	Address	Address (B Reg)
1 1 1 1	Illegal (TV16)	Illegal (TV16)	Illegal (TV16)	Illegal (TV16)

where:
- o X = least significant bit of BS length field
- o U = unsigned
- o S = signed

- MAP # specifies the AS Map as a function of the AS in Word 1 (i.e., EII1 or EII23). See Figure 3-11.

AS IN WORD 1	MAP # IN WORD 2	AS MAP #
EII1	0	RFU (TV05)
EII1	1	1
EII23	0	2
EII23	1	3

o For LXA instruction, the following applies:

0	1 3	4 7	8	9 15	
RFU	B#	DATA TYPE	MAP#	AS	DATA DESCRIPTOR

where:

o B# defines in which B register the word portion of the address is to be stored, right-justified.

Bits 1 through 3 of word 1 of the instruction, define in which R register the atom index portion of the address is to be stored, right-justified.

o DATA TYPE specifies the atom size of the data being pointed to:

DATA TYPE BITS				ATOM SIZE
4	5	6	7	
X	0	0	X	Bit
0	0	1	0	Digit (4 bits)
0	0	1	1	Illegal (TV16)
0	1	0	X	Byte
0	1	1	X	Word
1	0	1	X	Double Word
1	1	0	0	Illegal (TV16)
1	1	0	1	Quad Word
1	1	1	0	Address
1	1	1	1	Illegal (TV16)

The atom index is stored in R# when a bit, digit or byte atom size is selected.
For all the other atom sizes, R# is set to Zero.

o MAP # specifies the AS Map as a function of the AS in the instruction word. See definition of this bit in previous DD description.

6.1.2 General Rules

The following general rules apply to the Extended Integer Instructions:

- If the DD specifies a signed operand, and the operands have unequal lengths, then:

  - Right-justify and sign-extend to the left the shorter operand and/or

  - Store the appropriate low order bits of the result; set the OV indicator if the truncated bits are not all equal to the sign bit retained.

- If the DD specified an unsigned operand, and the operands have unequal lengths, then:

  - Right-justify and zero-fill to the left the shorter operand and/or

  - Store the appropriate low order bits of the result; set the OV indicator if the truncated bits are not all Zeroes.

- If an OV condition occurs and the result is being stored into a K#, then as a function of [M2], either trap or complete the instruction. Refer to subsection 3.3.4.6.

- The BS LENGTH field is unsigned and specifies the length of a bit string. The left-most bit of the string is pointed to by the Effective Address. If the AS does not specify indexing, the string begins on a word boundary and continues to the right. Bit strings may cross word boundaries.

- EII instructions perform their operations with 32 bit operands. If an instruction specifies an operand with size less than 32 bits, then the operand is converted in to a 32 bit operand before instruction execution.

6.2 EII DOUBLE OPERAND

EII double operand instructions are encoded using the format of the general double operand instructions with their AS field equal to EII1 or EII23. The following formats apply:

0	1 3	4 8	9 15	
1	K#	OP	EII1 or EII23	WORD 1
DATA DESCRIPTOR USING AS FROM MAP 1, 2 or 3				WORD 2

where:		
K#	=	Selects one of the seven double-word registers (K)
OP	=	Opcode field
EII1	=	Defines an EII instruction using an AS from Map 1
EII23	=	Defines an EII instruciton using an AS from Map 2 or 3
Data Descriptor	=	Specifies the type and location of an operand. Refer to subsection 6.1.1.

Refer to Figure 3-15.

The above format applies to all EII double operand instructions except for the LXA instruction, for which the format is given in subsection 6.2.12.

Depending upon whether the AS specifies RAS, MAS, or IMO form, the double operand instructions are defined to have the following format respectively:

o RR - Register to register
o RM - Register to memory
o RI - Register immediate.

These instructions are summarized in Table 6-2; their numerical representation is given in Table 6-3.

Table 6-2 EII Double Operands

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
6.2.1	KLD	Load Register K	[K#] <-- [EA]		
6.2.2	KST	Store Register K	[EA] <-- [K#]	OV	
6.2.3	KCM	Compare with Register K	[K#] :: [EA]	G, L, U	
6.2.4	KSW	Swap Register K	[K#] <---> [EA]	OV	
6.2.5	KADD	Add to Register K	[K#] <-- [K#] + [EA]	C, OV	
6.2.6	KSUB	Subtract from Register K	[K#] <-- [K#] - [EA]	C, OV	
6.2.7	KMUL	Multiply Register K	[K#] <-- [K#] * [EA]	OV	
6.2.8	KDIV	Divide Register K	[K#] <-- [K#] / [EA]	OV	Note 1
6.2.9	KOR	OR with Register K	[K#] <-- [K#] \/ [EA]		
6.2.10	KXOR	Exclusive OR with Register K	[K#] <-- [K#] ⊕ [EA]		
6.2.11	KAND	AND with Register K	[K#] <-- [K#] /\ [EA]		
6.2.12	LXA	Load Index and Address	[B#] <-- EA (address) [R#] <-- EA (atom index)		

Note

1 M6X and M6XE do not change the state of I(C) during the execution of this instruction.

Table 6-3 Numerical Representation of Word 1 of EII Double Operand Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
6.2.12	8+R#	8	0+m	n	LXA	A function of the data type in the Data Descriptor
6.2.2	8+K#	8	8+m	n	KST	
6.2.11	8+K#	9	0+m	n	KAND	
6.2.4	8+K#	9	8+m	n	KSW	
6.2.10	8+K#	A	0+m	n	KXOR	
6.2.6	8+K#	A	8+m	n	KSUB	
6.2.9	8+K#	B	0+m	n	KOR	
6.2.8	8+K#	B	8+m	n	KDIV	
6.2.1	8+K#	C	8+m	n	KLD	
6.2.3	8+K#	D	8+m	n	KCM	
6.2.5	8+K#	E	8+m	n	KADD	
6.2.7	8+K#	F	8+m	n	KMUL	

where: m, n = 6, C for EII1
       m, n = 7, C for EII23

6.2.1 Load Register K, KLD

Format:

RR, RM, RI

Description:

The contents of the location specified by the AS of the Data Descriptor are loaded into the designated K register.

Operation:

[K#] <-- [EA]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

6.2.2 Store Register K, KST

Format:

RR, RM, RI

Description:

The contents of the designated register K are stored in the location specified by the AS of the Data Descriptor.

Operation:

```
[EA] <-- [K#]
```

Indicator conditions:

```
                     C     Unchanged
                     B     Unchanged
                     I     Unchanged
                     G     Unchanged
                     L     Unchanged
                     U     Unchanged
   If [K#] > [EA] OV <-- 1 else OV <-- 0.
```

Special Conditions:

The use of IMO address form may cause alteration of procedure.

6.2.3 Compare With Register K, KCM

Format:

RR, RM, RI

Description:

The contents of the designated K register and the contents of the location specified by the AS of the Data Descriptor are compared (unsigned).

Operation:

```
I(G), I(L), I(U) <-- [K#] :: [EA]
```

Indicator Conditions:

```
                             C   Unchanged
                             B   Unchanged
                             I   Unchanged
If [K#] > [EA],        then G <--1 Else G <-- 0
If [K#] < [EA],        then L <--1 Else L <-- 0
If [K#(0)] ≠ [EA(0)],  then U <--1 Else U <-- 0
                             OV  Unchanged
```

Special Conditions:

None.

6.2.4 Swap Register K, KSW

Format:

RR, RM, RI

Description:
The contents of the designated K register are swapped with the contents of the locations specified by the AS of the Data Descriptor.

Operation:

[K#] <--> [EA]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged

If significant bits
are truncated, then OV <-- 1 Else OV <-- 0

Special Conditions:

Use of IMO address form may cause alteration of procedure.

6.2.5 Add to Register K, KADD

Format:

RR, RM, RI

Description:

The sum of the contents of the designated K register and the contents of the location specified by the AS of the Data Descriptor is loaded into the K register.

Operation:

[K#] <-- [K#] + [EA]

Indicator Conditions:

If carry, then C<--1 Else C<--0

B	Unchanged
I	Unchanged

```
                    G   Unchanged
                    L   Unchanged
                    U   Unchanged
   If overflow, then OV <-- 1 Else OV <--0
```

Special Conditions:

If an overflow occurs and M2(#) = 1, then a Trap TV06 occurs.

6.2.6 Subtract From Register K, KSUB

Format:

RR, RM, RI

Description:

The difference of the contents of the designated K register and the contents of the location specified by the AS of the Data Descriptor is loaded into the K register.

Operation:

[K#] <-- [K#] - [EA] (using 2's complement arithmetic)

Indicator Conditions:

```
        If carry, then C <-- 1   Else C <--0
                       B   Unchanged
                       I   Unchanged
                       G   Unchanged
                       L   Unchanged
                       U   Unchanged
   If overflow, then  OV <--1  Else OV <--0
```

Special Conditions:

If an overflow occurs and M2(#) = 1, then a Trap TV06 occurs.

6.2.7 Multiply Register K, KMUL

Function:

RR, RM, RI

Descriptor:

The product of the contents of the designated K register and the contents of the location specified by the AS of the Data Descriptor is loaded into K.

Operation:

[K#] <-- [K#] * [EA]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged

If overflow, then OV <-- 1 Else OV<--0. All operands remain unchanged if OV --> 1.

Special Conditions:

If an overflow occurs and M2(#) = 1, then a Trap TV06 occurs.

6.2.8 Divide Register K, KDIV

Format:

RR, RM, RI

Description:

The designated K register contents are divided by the contents of the location specified by the AS of the Data Descriptor and the resulting quotient is loaded into K.

If the contents of the location specified by the AS (divisor) is zero, or if the resulting quotient (Q) is $Q < -2^{31}$ or $Q > 2^{31} -1$, then the overflow condition is set, and the contents of the selected register are unchanged.

Operation:

[K#] <-- [K#] /[EA]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged

If divisor is Zero, or if the quotient causes Overflow, then all operands remain unchanged, and OV <-- 1, else OV <-- 0

Special Conditions:

1. If an overflow occurs and M2(#) = 1, then a Trap TV06 occurs.

2. This CSS does not change the state of I(C) during the execution of this instruction.

6.2.9 OR With Register K, KOR

Format:

RR, RM, RI

Description:

The inclusive OR of the contents of the designated K register and the contents of the location specified by the AS of the Data Descriptor is loaded into K.

Operation:

```
[K#] <-- [K#] \/ [EA]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

6.2.10 Exclusive OR With Register K, KXOR

Format:

RR, RM, RI

Description:

The exclusive OR of the contents of the designated K register and the contents of the location specified by the AS of the Data Descriptor is loaded into K.

Operation:

```
[K#] <-- [K#] ⊕ [EA]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged

L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

6.2.11 AND With Register K, KAND

Format:

RR, RM, RI

Description:

The logical AND of the contents of the designated K register and the contents of the location specified by the AS of the Data Descriptor is loaded into K.

Operation:

[K#] <-- [K#] /\ [EA]

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

6.2.12 Load Index and Address, LXA

Instruction format:

```
 0  1     3 4        8 9        15
 _________________________________
|  |      |          |           |
| 1|  R#  | 1 0 0 0 0|  EII1  or |     WORD 1
|  |      |          |  EII23    |
|__|______|__________|___________|
|                                |
| DATA DESCRIPTOR USING AS FROM MAP |  WORD 2
|            1, 2 or 3           |
|________________________________|
```

Format:

RM

Description:

The B register designated in the Data Descriptor is loaded with the word portion of the atom address. The designated R register is loaded with the atom index portion of the address.

An atom address is formed by using the atom size and AS defined by the Data Descriptor.

Operation:

```
[B#] <-- EA, and
[R#] <-- ATOM INDEX
```

If Data Descriptor specifies a word (or greater) atom size, then [R#] <-- 0

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

Example

The Data Descriptor specifies a byte atom size. The AS specifies B2 + D + R2 + O from AS Map 2, where:

```
[B2] = 1000
D    = 5
[R2] = 5
O (offset): not applicable to LXA
```

The atom address is then equal to 1007.1. If the selected B# = B3 and R# = R1 then the registers are loaded as follows:

```
[B3] <-- 1007
[R1] <-- 1
```

6.3 EII SINGLE OPERAND

EII single operand instructions are encoded using the format of the general single operand instructions with their AS field equal to EII1 or EII23. The following formats apply:

```
 0   1   3  4          8 9          15
 ________________________________________
|   |       |           |             |
| 1 | 0 0 0 |    OP     | EII1   or   |   WORD 1
|   |       |           | EII23       |
|___|_______|___________|_____________|
|                                     |
| DATA DESCRIPTOR USING AS FROM MAP   |   WORD 2
|             1, 2 or 3               |
|_____________________________________|
```

where OP	=	Opcode field
EII1	=	Defines an EII instruction using an AS from Map 1
EII23	=	Defines an EII instruction using an AS from Map 2 or 3
DATA DESCRIPTOR	=	Specifies the type and location of an operand. Refer to subsection 6.1.1.

Depending upon whether the AS specifies an RAS, MAS, or IMO form, these instructions are defined to have the following format:

R = Register only
M = Memory only
I = Immediate only.

These instructions are summarized in Table 6-4; their numerical representation is given in Table 6-5.

Table 6-4 EII Single Operand Instruction Summary

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
6.3.1	KINC	Increment*	[EA] <- [EA] + 1	OV, C	
6.3.2	KDEC	Decrement*	[EA] <- [EA] - 1	OV, C	
6.3.3	KNEG	Negate	[EA] <- 0 - [EA]	OV, C	
6.3.4	KCPL	Complement	[EA] <-- $\overline{[EA]}$		

*No memory interlock (RMW) is invoked during execution of this instruction.

Table 6-5 Numerical Representation of Word 1 of EII Single Operand Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
6.3.1	8	C	0+m	n	KINC	A function of data type in Data Descriptor
6.3.2	8	D	0+m	n	KDEC	
6.3.3	8	E	0+m	n	KNEG	
6.3.4	8	F	0+m	n	KCPL	

where: m, n = 6, C for EII1
m, n = 7, C for EII23

6.3.1 Increment, KINC

Format:

R, M, I

Description:

Increment by One the contents of the location specified by the AS in the Data Descriptor.

Operation:

[EA] <-- [EA] + 1

Indicator Conditions:

```
If carry, then      C <-- 1, Else C <-- 0
                    B    Unchanged
                    I    Unchanged
                    G    Unchanged
                    L    Unchanged
                    U    Unchanged
If overflow, then   OV <-- 1, Else OV <-- 0
```

The setting of OV for this instruction will not cause a Trap.

Special Conditions:

Use of IMO address form causes alteration of procedure.

6.3.2 Decrement, KDEC

Format:

R, M, I

Description:

Decrement by one the contents of the location specified by the AS of the Data Descriptor.

Operation:

[EA] <-- [EA] - 1

Indicator Conditions:

```
If carry, then      C <-- 1, Else C <-- 0
                    B     Unchanged
                    I     Unchanged
                    G     Unchanged
                    L     Unchanged
                    U     Unchanged
If overflow, then   OV <-- 1, Else OV <-- 0
```

The setting of OV for this instruction will not cause a Trap.

Special Conditions:

Use of IMO address form causes alteration of procedure.

6.3.3 Negate, KNEG

Format:

R, M, I

Description:

Two's complement the contents of the location specified by AS of the Data Descriptor.

Operation:

[EA] <-- 0 - [EA]

Indicator Conditions:

```
If [EA] = 0, then          C <-- 1, Else C <-- 0
                           B     Unchanged
                           I     Unchanged
                           G     Unchanged
                           L     Unchanged
                           U     Unchanged
  If [EA] = 8000 0000, then OV <-- 1  Else OV <-- 0
  The setting of OV for this instruction will not cause a Trap.
```

Special Conditions:

Use of IMO address form causes alteration of procedure.

6.3.4 Complement, KCPL

Format:

R, M, I

Description:

One's complement the contents of the location specified by the AS of the Data Descriptor.

Operation:

$[EA] \leftarrow [\overline{EA}]$

Indicator Conditions:

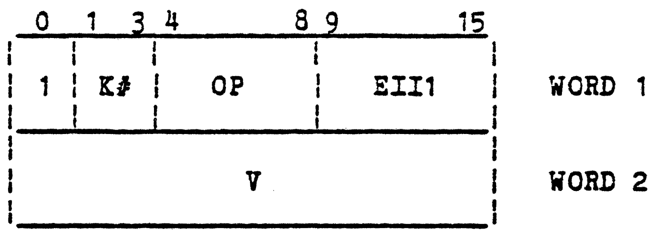
C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

Use of IMO address form causes alteration of procedure.

6.4 EII SHORT VALUE IMMEDIATE

The EII short value immediate (KSI) instructions have the following format:



where K# = Selects one of the seven double-word registers (K)

OP = Opcode field

EII1 = Defines an Extended Integer Instruction

V = Immediate operand value, sign extended range is $2^{15} \leq V \leq 2^{15} - 1$

These instructions operate on the K registers and perform the following operations: load, compare, add and multiply. These instructions are summarized in Table 6-6; their numerical representations are given in Table 6-7.

Table 6-6 EII Short Value Immediate Instructions

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
6.4.1	KLDV	Load Value*	[K#] <-- V		
6.4.2	KCMV	Compare Value*	[K#] :: V	G, L, U	
6.4.3	KADV	Add Value*	[K#] <-- [K#] + V	C, OV	
6.4.4	KMLV	Multiple by Value*	[K#] <-- [K#] * V	OV	

*The 16-bit V is sign extended to 32 bits

Table 6-7 Numerical Representation of EII Short Value Immediate Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
6.4.1	8+K#	0	E	C	KLDV	NOT APPLI-CABLE
6.4.2	8+K#	1	E	C	KCMV	
6.4.3	8+K#	2	E	C	KADV	
6.4.4	8+K#	3	E	C	KMLV	

6.4.1 Load Value, KLDV

Format:

KSI

Description:

Load the 16 bit Value field (sign extended) into the designated K register.

Operation:

```
[K#(16:31)] <-- [V(0:15)]; [K#(0:15)] <-- [V(0)]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

None.

6.4.2 Compare With Value, KCMV

Format:

KSI

Description:

The unsigned comparison of the contents of the designated K register and the 16-bit value field (sign extended) is used to set G, L, or U indicators.

Operation:

```
[TEMP(16:31)] <-- [V(0:15)]; [TEMP(0:15)] <-- [V(0)];
I(G), I(L), I(U) <-- [K#] :: [TEMP]
```

Indicator Conditions:

```
                         C    Unchanged
                         B    Unchanged
                         I    Unchanged
If [K#]>[TEMP] then      G <-- 1, Else G <-- 0
If [K#]<[TEMP] then      L <-- 1, Else L <-- 0
If K#(0) ≠ V(0), then    U <-- 1, Else U <-- 0
                         OV   Unchanged
```

Special Conditions:

None.

6.4.3 Add Value, KADV

Format:

KSI

Description:

The sum of the contents of the designated K register and the value field (sign extended) is loaded into K.

Operation:

```
[TEMP(16:31)] <-- [V(0:15)]; [TEMP(0:15)] <-- [V(0)];
[K#] <-- [K#] + [TEMP]
```

Indicator Conditions:

```
If Carry, then     C <-- 1, Else C <-- 0
                   B   Unchanged
                   I   Unchanged
                   G   Unchanged
                   L   Unchanged
                   U   Unchanged
If Overflow, then OV <-- 1, Else OV <-- 0
```

Special Conditions:

If an overflow occurs and M2(#) = 1, then Trap TV06 occurs.

6.4.4 Multiply by Value, KMLV

Format:

KSI

Description:

The product of the designated K register contents and the 16 bit value field (sign extended) is loaded into K.

Operation:

```
[TEMP(16:31)] <-- [V(0:15)]; [TEMP(0:15)] <-- [V(0)];
[K#] <-- [K#] * [TEMP]
```

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged

If Overflow, then OV <-- 1, Else OV <-- 0

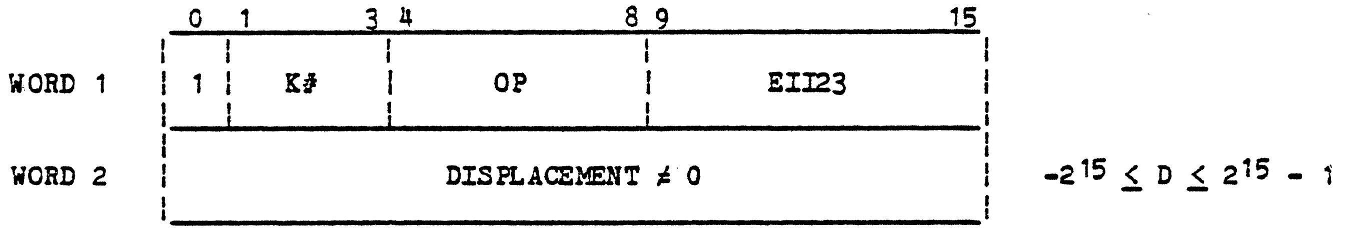
Special Conditions:

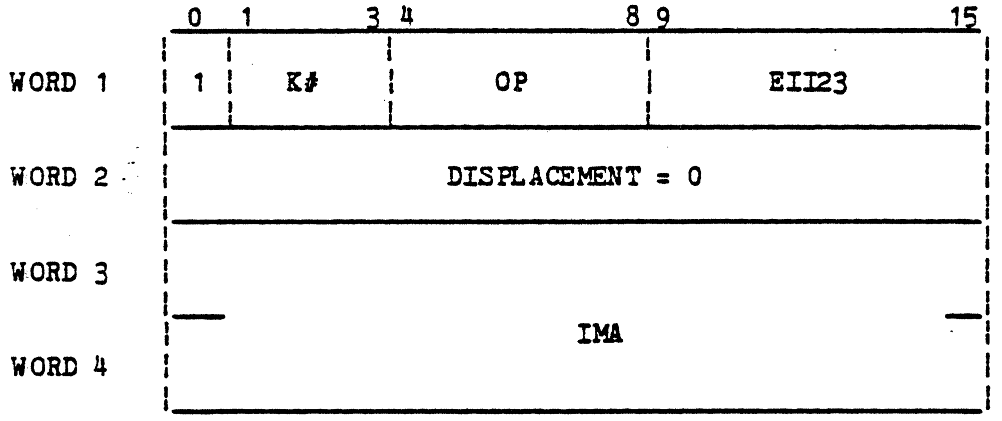
1. If OV <-- 1, [K#] is unchanged.

2. If an overflow occurs and M2(#) = 1, then Trap TV06 occurs.

6.5 EII BRANCH ON REGISTER

These instructions have the following formats (KBR):





where:	K#	= Selects one of seven operand registers (K)
	OP	= Opcode - determines the branch condition
	EII23	= Defines an EII instruction.
	DISPLACEMENT (D)	= Defines how to compute EA as follows:

- o For $D \neq 0$, EA = P + D
- o For $D = 0$, EA = IMA

These instructions enable branching on a selected K register, e.g., equal to Zero, less than Zero, etc. These instructions are defined in summary form in Table 6-8; their numerical representation is given in Table 6-9.

Table 6-8 EII Branch on Register Instructions

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
6.5.1	KBLZ	Branch if [K] is Less than Zero	If [K#(0)] = 1, then [P] <-- EA		In all branch operations, if branch condition is true and M1(J) = 1, then post a Trap TV02 after instruction is executed
6.5.2	KBGEZ	Branch if [K] is Greater than or Equal to Zero	If [K#(0)] = 0, then [P] <-- EA		
6.5.3	KBEZ	Branch if [K] is Equal to Zero	If [K#] = 0, then [P] <-- EA		
6.5.4	KBNEZ	Branch if [K] is Not Equal to Zero	If [K#] ≠ 0, then [P] <-- EA		
6.5.5	KBGZ	Branch if [K] is Greater than Zero	If ([K#] ≠ 0) /\ ([K#(0)] = 0), then [P] <-- EA		
6.5.6	KBLEZ	Branch if [K] is Less than or Equal to Zero	If ([K#(0)] = 1) \/ ([K#] = 0), then [P] <-- EA		
6.5.7	KBEVN	Branch if [K] is Even	If [K#(31)] = 0, then [P] <-- EA		
6.5.8	KBODD	Branch if [K] is Odd	If [K#(31)] = 1, then [P] <-- EA		

Table 6-9 Numerical Representation of Word 1 of EII Branch on Register Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
6.5.1	8+K#	0	7	C	KBLZ	
6.5.2	8+K#	0	F	C	KBGEZ	
6.5.3	8+K#	1	7	C	KBEZ	NOT
6.5.4	8+K#	1	F	C	KBNEZ	APPLI-
6.5.5	8+K#	2	7	C	KBGZ	CABLE
6.5.6	8+K#	2	F	C	KBLEZ	
6.5.7	8+K#	3	7	C	KBEVN	
6.5.8	8+K#	3	F	C	KBODD	

where $0 < K\# \leq 7$

6.5.1 Branch If [K] Less Than Zero, KBLZ

Format:

KBR

Description:

Branch to Effective Address if the contents of the designated K register are negative.

Operation:

If [K#(0)] = 1, then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If branch condition is true and M1(J) = 1, then a Trap TV02 occurs after the KBLZ instruction is executed.

6.5.2 Branch If [K] Greater Than or Equal to Zero, KBGEZ

Format:

KBR

Description:

Branch to the Effective Address if the contents of the designated K register are positive or Zero.

Operation:

If [K#(0)] = 0, then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true and M1(J) = 1, then a Trap TV02 occurs after the KBGEZ instruction is executed.

6.5.3 Branch If [K] Equal to Zero, KBEZ

Format:

KBR

Description:

Branch to the Effective Address if the contents of the designated K register are Zero.

Operation:

If [K#] = 0, then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true and M1(J) = 1, then a Trap TV02 occurs after the KBEZ instruction is executed.

6.5.4 Branch If [K] Not Equal to Zero, KBNEZ

Format:

KBR

Description:

Branch to the Effective Address if the contents of the designated K register are not Zero.

Operation:

If [K#] ≠ 0, then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
CV	Unchanged

Special Conditions:

If the branch condition is true and M1(J) = 1, then a Trap TV02 occurs after the KBNEZ instruction is executed.

6.5.5 Branch If [K] Greater Than Zero, KBGZ

Format:

KBR

Description:

Branch to the Effective Address if the contents of the designated K register are greater than Zero.

Operation:

If ([K#] ≠ 0) /\ ([K#(0)] = 0), then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged

I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true and M1(J) = 1, then a Trap TV02 occurs after the KBGZ instruction is executed.

6.5.6 Branch If [K] Less Than or Equal to Zero, KBLEZ

Format:

KBR

Description:

Branch to the Effective Address if the contents of the designated K register are less than or equal to Zero.

Operation:

If ([K#(0)] = 1) \/ ([K#] = 0), then [P] <-- EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true and M1(J) = 1 then a Trap TV02 occurs after the KBLEZ instruction is executed.

6.5.7 Branch If [K] Even, KBEVN

Format:

KBR

Description:

Branch to the Effective Address if the contents of the designated K register are even.

Operation:

If [K#(31)] = 0, then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true and M1(J) = 1, then a Trap TV02 occurs after the KBEVN instruction is executed.

6.5.8 Branch If [K] Odd, KBODD

Format:

KBR

Description:

Branch to the Effective Address if the contents of the designated K register are odd.

Operation:

If [K#(31)] = 1, then [P] <— EA

Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If the branch condition is true and M1(J) = 1, then a Trap TV02 occurs after the KBODD instruction is executed.

6.6 EII SHIFT OPERATIONS

EII shift instructions (KSHL) are encoded using a format that is similar to that used by the general shift long instructions.

```
 0   1  3  4       7  8    10 11       15
 _________________________________________
|   |     |         |      |             |
| 0 | K#  | 0 0 0 1 |  T   |      D      |   WORD 1
|___|_____|_________|______|_____________|
```

where K# = Selects one of seven double-word registers (K)

T = Identifies type and direction of the shift

D = Distance ($1 \leq D \leq 31$); if D = 0, substitute contents of R1(11:15) for the distance. If [R1(11:15)] = 0, then the register's contents are unchanged and the indicators relevant to the instruction are cleared.

These instructions are summarized in Table 6-10; Their numerical representation is given in Table 6-11.

Table 6-10 EII Shift Instructions (Sheet 1 of 2)

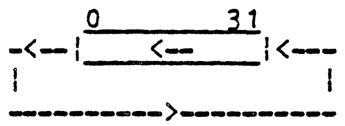
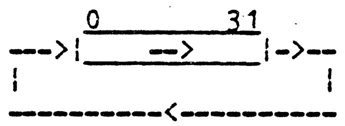
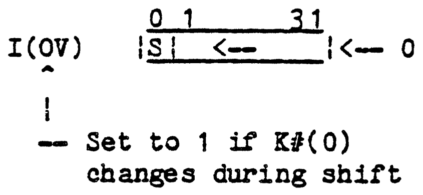
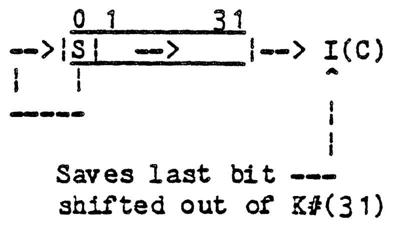
REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED
6.6.1	KCL	Shift Closed Left		
6.6.2	KCR	Shift Closed Right		
6.6.3	KAL	Shift Arithmetic Left	 0 1 31 I(OV) S 0 Set to 1 if K#(0) changes during shift	OV
6.6.4	KAR	Shift Arithmetic Right	 0 1 31 S I(C) Saves last bit shifted out of K#(31)	C

Table 6-10 EII Shift Instructions (Sheet 2 of 2)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED

6.6.5	KSOL	Shift Open Left	`0 31`
			`I(C)<--|___<--___|<-- 0`
			`^`
			`|`
			`--- Saves last bit shifted out of K#(0)`
				C

6.6.6	KSOR	Shift Open Right	`0 31`
			`0 -->|___-->___|-->I(C)`
			`^`
			`|`
			`Saves last bit ------ shifted out of K#(31)`
				C

Table 6-11 Numerical Representation of EII Shift Instructions

0 3 4 7 8 11 12 15

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
6.6.1	K#	1	2	D	KCL*	
6.5.1	K#	1	3	D - 16	KCL**	
6.6.2	K#	1	6	D	KCR*	NOT
6.6.2	K#	1	7	D - 16	KCR**	
6.6.5	K#	1	8	D	KSOL*	APPLI-
6.6.5	K#	1	9	D - 16	KSOL**	
6.6.3	K#	1	A	D	KAL*	CABLE
5.6.3	K#	1	B	D - 16	KAL**	
6.6.6	K#	1	C	D	KSOR*	
6.6.6	K#	1	D	D - 16	KSOR**	
6.6.4	K#	1	E	D	KAR*	
6.6.4	K#	1	F	D - 16	KAR**	

where $1 \leq K\# \leq 7$ in bits 1 through 3

D = Distance in bits, $1 \leq D \leq 31$

*For $D \leq 15$

**For $D > 15$

6.6.1 Shift [K], Closed, Left, KCL

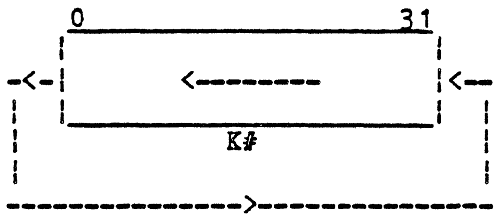
Format:

KSHL

Description:

The contents of the designated K register are shifted D bit positions to the left; bits shifted out of [K#(0)] replace bits vacating [K#(31)].

Operation:



Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If D = 0, the K registers' contents are unchanged.

6.6.2 Shift [K], Closed, Right, KCR

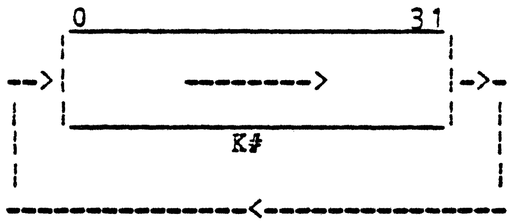
Format:

KSHL

Description:

The contents of the designated K register are shifted D bit positions to the right; bits shifted out of the [K#(31)] replace bits vacating [K#(0)].

Operation:



Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged
OV	Unchanged

Special Conditions:

If D = 0, then the K registers' contents are unchanged.

6.6.3 Shift [K], Arithmetic, Left, KAL

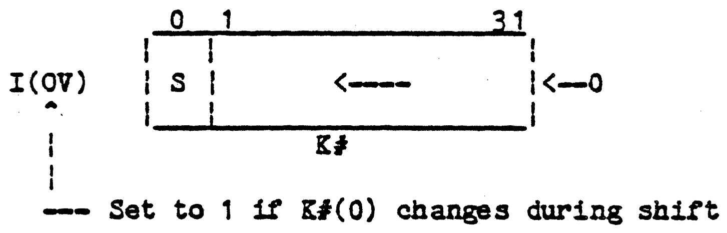
Format:

KSHL

Description:

The contents of the designated K register are shifted D bit positions to the left. If the sign bit K#(0) changed at any time during the operation, the OV indicator is set. Zero fill the d least significant bit positions of K#.

Operation:



Indicator Conditions:

C	Unchanged
B	Unchanged
I	Unchanged
G	Unchanged
L	Unchanged
U	Unchanged

If [K#(0)] changes, then OV <— 1, Else OV <— 0

Special Conditions:

1. If D = 0, then the K registers' contents are unchanged, and I(OV) is cleared.

2. If an overflow occurs and M2(#) = 1, then a Trap TV06 occurs.

6.6.4 Shift [K], Arithmetic, Right, KAR

Format:

KSHL

Description:

The contents of the designated K register are shifted D bit positions to the right. Bits equal to the sign bit [K#(0)] fill the d most significant bits of the K# register and the last bit shifted out of [K#(31)] is saved in the C indicator.

Operation:

```
         0  1                  31
        _________________________
        |   |                    |
    -->| S |        -->         |-->I(C)
    |   |___|____________________|      ^
    |     |           K#                |
    ------                              |
          Saves last bit shifted out ---
          of K#(31)
```

Indicator Conditions:

```
If D ≠ 0 and the last bit
shifted out of [K#(31)] = 1, then C<-- 1, Else C <-- 0
                                  B   Unchanged
                                  I   Unchanged
                                  G   Unchanged
                                  L   Unchanged
                                  U   Unchanged
                                  OV  Unchanged
```

Special Conditions:

If D = 0, then the K registers' contents are unchanged, and I(C) is cleared.

6.6.5 Shift [K#], Open, Left, KSOL

Format:

KSHL

Description:

The contents of the designated K register are shifted D bit positions to the left. Zero fill the d least significant bit positions of K# register. The least bit shifted out of [K#(0)] is saved in the C indicator.

Operation:

```
            0                31
            |                 |
  I(C)<--|        <---        |<-- 0
     ^      |_________________|
     |              K#
     |
     --- Saves last bit shifted out of K#(0)
```

Indicator Conditions:

```
If D ≠ 0 and if the last bit
shifted out of K#(0)= 1, then C <-- 1, Else C <-- 0
                                            B   Unchanged
                                            I   Unchanged
                                            G   Unchanged
                                            L   Unchanged
                                            U   Unchanged
                                            OV  Unchanged
```

Special Conditions:

If D = 0, then the K registers' contents are unchanged, and I(C) is cleared.

6.6.6 Shift [K#], Open, Right, KSOR

Format:

KSHL

Description:

The contents of the designated K register are shifted D bit positions to the right. Zero fill the d most significant bit positions of the K# register. The last bit shifted out of [K#(31)] is saved in the C indicator.

Operation:

```
          0                31
          |                 |
  0 -->|        --->        |-->I(C)
          |_________________|      ^
                                   |
  Saves last bit shifted out of ---
  K#(31)
```

Indicator Conditions:

If D ≠ 0 and if the last bit shifted out of [K#(31)] = 1, then	C	<-- 1 Else C <-- 0
	B	Unchanged
	I	Unchanged
	G	Unchanged
	L	Unchanged
	U	Unchanged
	OV	Unchanged

Special Conditions:

If D = 0, then the K registers' contents are unchanged, and I(C) is cleared.

This page is intentionally blank.

SECTION 7 COMMERCIAL INSTRUCTIONS

7.1 INTRODUCTION

This section describes the commercial instruction set. These instructions are part of the CSS instruction set and fall under the generic and branch groupings.

The commercial instructions are classified as follows:

o Numeric (subsection 7.3.1)
o Alphanumeric (subsection 7.3.2)
o Edit (subsection 7.3.3)
o Branch (subsection 7.3.4).

The commercial numeric, alphanumeric, and edit instructions use Data Descriptors (DDs) to specify their operands.

The commercial branch instructions are similar to the general branch on indicator instructions.

For convenience the instructions are summarized in Appendix G. They also appear in Appendix E as a function of their format.

7.2 DATA DESCRIPTORS

Data Descriptors (DDs) are used to define the operand type, size and location in memory. They can be In-Line DDs (IDs) or Remote DDs (RDs), but regardless of their location in memory, they have the same format.

IDs are part of a Commercial Instruction (see Figure 3-16).

RDs are defined by a label within the Commercial Instruction (see Figure 3-17). The label is a 12-bit positive integer which is used as an offset from the remote descriptor base address (contained in the Remote Descriptor Base Register, RDBR.

As a function of the instruction op code, a DD can be one of the following:

o Decimal DD
o Alphanumeric DD, or
o Binary DD.

7.2.1 Decimal Data Descriptors

Decimal DDs are implied by all numeric instructions and the Numeric Edit instruction. Decimal DDs can refer to either string decimal or packed decimal data. The format for Decimal DDs is shown in Figure 7-1.

```
   0    1    2  3     7   8  9              15
 |    |    |    |      |    |                 |
 | C1 | C2 | C3 |  L   | T  |       AS        |   WORD 1
 |____|____|____|______|____|_________________|
 |                                            |
 |              AS SPECIFIC                   |   WORD 2
 |____________________________________________|
```

where:

WORD 1:
- C1, C2, C3 = Control bits which specify byte or half-byte offset and sign information

- L= Specifies the length of the operand

- T= Specifies the data type:
  T = 0, data is string decimal;
  T = 1, data is packed decimal.

- AS= Specifies the address syllable (see subsection 3.11).

WORD 2:
- The contents of Word 2 is as defined by the AS (see subsection 3.11).

Figure 7-1 Decimal Data Descriptor Format

7.2.1.1 STRING DECIMAL DD

In a String Decimal DD the fields shown in Figure 7-1 have the following meanings:

o C1 (Bit 0) - Byte offset:

- When no indexing is specified, C1 specifies the offset within the addressed word:
  C1 = 0: operand starts in the leftmost byte of the addressed word.
  C1 = 1: operand starts in the rightmost byte of the addressed word.

- When indexing is specified, C1 contains a byte offset value which is added to the index value and the resulting sum is used to compute the EA.

o C2, C3 (Bits 1 and 2) - Sign control as shown below:

C2	C3	SIGN CONVENTION
0	0	Unsigned*
0	1	Trailing Overpunch
1	0	Leading Separate Sign
1	1	Trailing Separate Sign

*The operand is positive.

o L (Bits 3-7) - Specifies the length of the operand:
for L ≠ 0, then $1 \leq L \leq 31$
for L = 0, then escape to Rn (11-15), where:

n = 4 for DD1
n = 5 for DD2
n = 6 for DD3.

Rn (8-10) should be Zero else unspecified results occur.

For unsigned or signed overpunched operands, the length refers to digits.

For leading or trailing separate signed operand, the length refers to L-1 digits and a sign.

An Illegal Specification Trap TV26 results if an operand has either:

- a length of Zero, or
- a length of one and specifies separate sign (i.e., the operand consists only of a sign).

- o T (Bit 8) = Zero.
- o AS (Bits 9 - 15)- Specifies the address syllable (see subsection 3.11).
- o WORD 2 - The contents of Word 2 are as defined by the AS (see subsection 3.11).

7.2.1.2 PACKED DECIMAL DD

In a Packed Decimal DD the fields shown in Figure 7-1 have the following meaning:

- o C1, C2 (Bit 0, 1) - Half byte offset:
  - When no indexing is specified, then C1 and C2 specify the offset within the addressed word:

C1	C2	POSITION OF FIRST DIGIT WITHIN THE ADDRESSED WORD
0	0	digit 0 (bits 0 - 3)
0	1	digit 1 (bits 4 - 7)
1	0	digit 2 (bits 8 - 11)
1	1	digit 3 (bits 12 - 15)

  - When indexing is specified C1 and C2 contain a half-byte offset value which is added to the index value and the resulting sum is used to compute the EA.

- o C3 (Bit 2) - Sign control:
  - If C3 = 0, the operand is unsigned;
  - If C3 = 1, the operand is trailing signed.

  When unsigned, the operand is considered to be positive. When signed, only trailing sign is allowed.

- o L (Bits 3 - 7) - Specifies the length of the operand directly or indirectly. The rules in subsection 7.2.1.1.for L also apply here.
- o T (Bit 8) = One.
- o AS (Bits 9 - 15) - Specifies the address syllable (see subsection 3.11).
- o WORD 2 - The contents of Word 2 are as defined by the AS (see subsection 3.11).

7.2.2 Alphanumeric Data Descriptors

Alphanumeric DDs are implied by all Alphanumeric instructions and the Alphanumeric Edit instruction. The format of the DD is shown in Figure 7-2.

```
 0    1    2  3    7  8  9           15
 |    |    |  |       |  |              |
 | C1 | C2 | 0|   L   | 0|      AS      |   WORD 1
 |____|____|__|_______|__|______________|
 |                                      |
 |             AS SPECIFIC              |   WORD 2
 |______________________________________|
```

Figure 7-2 Alphanumeric Data Descriptor Format

o C1 (Bit 0) - Byte offset:

- When no indexing is specified, C1 specifies the offset within the addressed word:

  If C1 = Zero, operand starts with leftmost byte of the addressed word.
  If C1 = One, operand starts with the rightmost byte of the addressed word.

- When indexing is specified, C1 contains a byte offset value which is added to the index value and the resulting sum is used to compute the EA.

o C2 (Bit 1) - Fill Control. This bit is used in DD2 of an ALR or MAT. It specifies whether or not the receiving field is to be filled:

If C2 = Zero, do not fill;
If C2 = One, fill.

The fill character is either a blank or as specified in [R5(0:7)]. See definition of the L field.

In an ACM instruction, the same rules apply except that fill is always assumed, regardless of the state of C.

In DME and AME instructions, no fill is performed.

o Bit 2 - MBZ or unspecified results will occur.

o L (Bits 3 - 7) = Specifies the length of the operand either directly or indirectly. It also specifies the fill character. Refer to the specific instruciton for a description of the interpretation of the L field.

o Bit 8 - MBZ or an Illegal Specification Trap TV26 occurs.

o AS (Bits 9 - 15) - Specifies the address syllable (see subsection 3.11).

o WORD 2 - The contents of WORD 2 are as defined by the AS (see subsection 3.11).

7.2.3 Binary Data Descriptors

Binary DDs are implied by the CBD and the CDB instructions. Their format is shown in Figure 7-3.

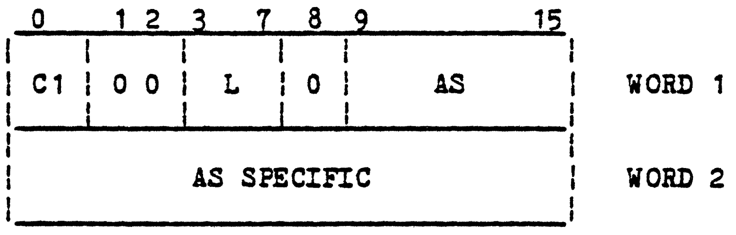


Figure 7-3 Binary Data Descriptor Format

o C1 (Bit 0) - Byte offset. Same definition as given for an alphanumeric DD applies.

o Bits 1, 2 - MBZ or an unspecified result occurs.

o L (Bits 3 - 7) - Defines the binary precision, either 16 bits or 32 bits.

- If L ≠ 0, then the L must be two or four, otherwise unspecified results occur.

- If L = 2, then binary operand is 16 bits (single precision).

- If L = 4, then binary operand is 32 bits (double precision).

- If L = 0, then the length is given by R4(11:15) for DD1 or R5(11:15) for DD2.

- Rn (11-15) can be set to two or four only, otherwise unspecified results occur. The same rules for L = 2 and L = 4, as described above, apply.

o Bit 8 - MBZ or an Illegal Specification Trap TV26 occurs.

o Bits 9 - 15 (AS) - Specifies the address syllable (see subsection 3.11).

- WORD 2 - The contents of WORD 2 are as defined by the AS (see subsection 3.11).

NOTE

A binary descriptor is a special case of an alphanumeric descriptor. Specifically, it is equal to an alphanumeric descriptor which has its C2 bit set to Zero (i.e., specifies no fill) and its length set to either two or four. Consequently, alphanumeric instructions can be used to operate on (e.g., move or compare) binary operands. The binary descriptor(s) in this case is interpreted as alphanumeric.

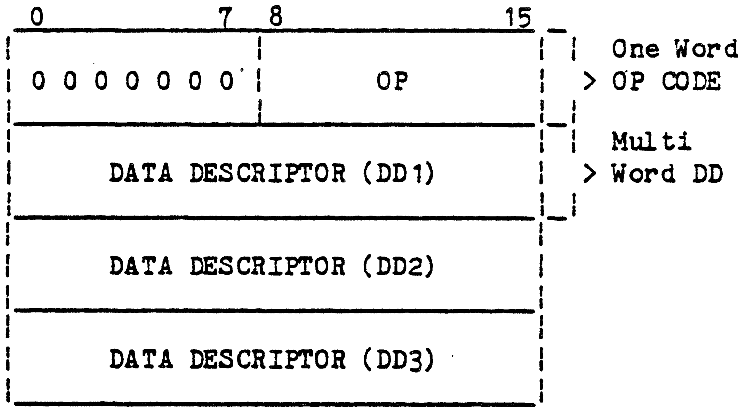
7.3 COMMERCIAL INSTRUCTION SET

The following subsections describe the Commercial instructions. These instructions are classified as follows:

- Numeric
- Alphanumeric
- Edit
- Branch.

The format for the numeric, alphanumeric, and edit instructions is given in Figure 7-4 (also see Figure 3-16). They are summarized in Table 7-1 and their numerical representation is given in Table 7-2.

The format for the branch instructions is given in Figure 7-7. They are summarized in Table 7-3 and their numerical representation is given in Table 7-4.



where:

OP - Opcode (002X)

DDn - Data Descriptor - Specifies the type, size and location of the operand. DD1 refers to the first operand, DD2 refers to the second and DD3 refers to the third.

Figure 7-4 Numeric, Alphanumeric and Edit Instruction Formats

Table 7-1 Commercial Numeric, Alphanumeric, and Edit Instruction Summary (Sheet 1 of 2)

REFERENCE	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS					TRAPS					NUMBER OF OPER-ANDS	COMMENTS
				OV	TR	SF	G	L	IS	IC	DZ	TR	OV		
NUMERIC INSTRUCTIONS															
7.3.1.1	DAD	Decimal Add	[DD2] + [DD1] --> [DD2]	X		X	X	X	X	X			X	2 (NN)	
7.3.1.2	DSB	Decimal Subtract	[DD2] - [DD1] --> [DD2]	X		X	X	X	X	X			X	2 (NN)	
7.3.1.3	DML	Decimal Multiply	[DD2] x [DD1] --> [DD2]	X		X	X	X	X	X			X	2 (NN)	
7.3.1.4	DDV	Decimal Divide	$\frac{[DD2]}{[DD1]}$ -> [DD3] R --> [DD2]	X		X	X	X	X	X	X		X	3 (NNN)	
7.3.1.5	DCM	Decimal Compare	[DD1] :: [DD2] --> IND				X	X	X	X				2 (NN)	Is DD1 G or L than DD2? Zero fill to the left supplied for smaller operand
7.3.1.6	DMC	Decimal Move and Convert	[DD1] converted --> [DD2]	X		X	X	X	X	X			X	2 (NN)	Combinations are S --> S, S --> P, P --> S, P --> P*
7.3.1.7	CBD	Convert Binary to Decimal	[DD1] converted --> [DD2]	X		X			X				X	2 (BN)	Binary operand is in 2s complement form
7.3.1.8	CDB	Convert Decimal to Binary	[DD1] converted --> [DD2]	X					X	X			X	2 (NB)	Binary result is a 2s complement number
7.3.1.9	DSH	Decimal Shift	Shift [DD1] Left "d"	X			X	X	X	X			X	1 (N)	"d" = shift distance; OV indicator and Trap, valid only for shift left. Optional rounding during a shift right.
			Shift [DD1] Right "d"				X	X	X	X					

*where S = String operand; P = Packed operand.

Definitions: N = Decimal or Numeric Operand
B = Binary Operand
A = Alphanumeric Operand
E = Edit

Table 7-1 Commercial Numeric, Alphanumeric, and Edit Instruction Summary (Sheet 2 of 2)

REFERENCE	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS					TRAPS					NUMBER OF OPER-ANDS	COMMENTS
				OV	TR	SF	G	L	IS	IC	DZ	TR	OV		
ALPHANUMERIC INSTRUCTIONS															
7.3.2.1	ALR	Alphanumeric Move (Interruptable)	[DD1] --> [DD2]		X				X			X		2 (AA)	Fill or do not fill to the right defined by DD2
7.3.2.2	ACM	Alphanumeric Compare (Interruptable)	[DD1] :: [DD2]				X	X	X					2 (AA)	If DD1(L) ≠ DD2(L), then extend shorter operand as specified by DD2
7.3.2.3	MAT	Alphanumeric Move and Translate (Interruptable)	[DD1] Translate --> [DD2] [DD3] specifies 256 byte Translate Table		X				X			X		3 (AAA)	Fill character defined by DD2 is used directly
7.3.2.4	SRCH	Alphanumeric Search (Interruptable)	[DD3] is searched using [DD1] as SL. Result --> G, L indicators and displacement, SA number --> [DD2]				X	X	X					3 (AAA)	SL = Search List SA = Search Argument
7.3.2.5	VRF	Alphanumeric Verify (Interruptable)	[DD3] is verified using [DD1] as VL. Result --> G,L indicators and displacement --> [DD2]				X	X	X					3 (AAA)	VL = Verify List
EDIT INSTRUCTIONS															
7.3.3.1.1	DME	Decimal Move and Edit (Interruptable)	[DD1] edited --> [DD2], DD3 specifies Micro-ops						X	X				3 (NAA)	DD1 = decimal desc.; DD2 = DD3 = alphanumeric desc.
7.3.3.1.2	AME	Alphanumeric Move and Edit (Interruptable)	[DD1] edited --> [DD2], DD3 specifies Micro-ops						X					3 (AAA)	DD1 = DD2 = DD3 = alphanumeric desc.

Table 7-2 Numerical Representation of Numeric, Alphanumeric, and Edit Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
7.3.2.5	0	0	2	0	VRF	Note 3
7.3.2.1	0	0	2	1	ALR	Note 3
7.3.2.2	0	0	2	2	ACM	Note 3
7.3.2.3	0	0	2	3	MAT	Note 3
7.3.3.1.2	0	0	2	4	AME	Note 3
7.3.1.6	0	0	2	5	DMC	Note 1
7.3.3.1.1	0	0	2	6	DME	Notes 1,3
7.3.1.7	0	0	2	7	CBD	Note 2
7.3.2.4	0	0	2	8	SRCH	Note 3
7.3.1.3	0	0	2	9	DML	Note 1
7.3.1.8	0	0	2	A	CDB	Note 2
7.3.1.4	0	0	2	B	DDV	Note 1
7.3.1.1	0	0	2	C	DAD	Note 1
7.3.1.2	0	0	2	D	DSB	Note 1
7.3.1.9	0	0	2	E	DSH	Note 1
7.3.1.5	0	0	2	F	DCM	Note 1

NOTES FOR TABLE 7-2

1. The atom size of each numeric operand may be either a byte or a digit as defined by the DD.

2. The atom size of each numeric operand may be either a byte or a digit as defined by the DD. The atom size of the binary operand must be either 2 or 4 bytes.

3. The atom size of each alphanumeric operand is a byte.

7.3.1 Numeric

The following general rules apply to all commercial numeric instructions:

1. EA always points to the leftmost digit or leading sign of the operand.

2. G, L indicators indicate value of result relative to Zero (except for CDB and CBD instructions) regardless of overflow.

3. If the result is shorter than the receiving field, the receiving field is Zero-filled to the left except for the CDB instruction where the result is sign extended instead.

4. (+) and (-) Zero are allowed on input but are treated as + Zero during instruction execution. All Zero results generated by the hardware are + Zero.

5. A string decimal Zero = 30_{16}; a packed decimal Zero = 0_{16}.

6. All signed results have hardware generated signs, regardless of the sign convention used by the operands at execution time. Refer to Tables 3-1 and 3-2.

7. Zone bits (leftmost bits of a string decimal number) are always ignored on input. Nevertheless, all results have the zone bits set to 3 hexadecimal.

8. Operands must not overlap each other, otherwise unspecified results occur (except as allowed by Rule #9).

9. Identical operands are allowed.

10. If a trap condition is disabled, the trap does not occur and the receiving field is altered.

11. Mixing of operands (i.e., string and packed decimal) is allowed.

12. Operands having different sign conventions are allowed.

13. Any operand having a Zero length or a length equal to just the sign character, and specifying separate sign, results in an IS Trap TV26 condition.

14. OV is set if the receiving field cannot accept all significant digits of the result.

15. If either operand has an illegal digit or an illegal sign, then an Illegal Character Trap TV27 results.

16. If a negative result is to be stored in an unsigned receiving field, then the sign fault indicator is set.

17. If DD2 specifies IMO in any instruction except a DCM, a Trap TV26 occurs.

18. If DD3 specifies IMO, a Trap TV26 occurs.

19. Truncation Traps can occur only during the execution of Alphanumeric instructions.

20. The original operands of a commercial numeric instruction are preserved if a Trap IS, IC, DZ, or OV occurs. For any other Trap type, the state of the operands are undefined.

21. If the receiving field cannot accept all significant digits of the result and M3 register specifies that no trap be generated, then the receiving field contains only low order digits (i.e., the high order digits are lost).

22. Comparison is algebraic.

23. The following defines the commercial numeric traps:

TRAP EVENT	MNEMONIC	TRAP NUMBER
Reference to protected memory	PV	14* or 32
Illegal character	IC	15* or 23
Divide by Zero	DZ	17* or 24
Illegal specification	IS	26
Illegal character	IC	27
Divide by Zero	DZ	25
Overflow	OV	29

* This trap number is used when the commercial instruction is executed by the CPU as opposed to an auxiliary porcessor.

7.3.1.1 DECIMAL ADD, DAD

Type:

Numeric.

Description:

[DD2] + [DD1] --> [DD2]

Indicator Conditions:

OV, SF, G, L

Trap Conditions:

IS, IC, OV

7.3.1.2 DECIMAL SUBTRACT, DSB

Type:

Numeric

Description:

[DD2] - [DD1] --> [DD2]

Indicator Conditions:

OV, SF, G, L

Trap Conditions:

IS, IC, OV

7.3.1.3 DECIMAL MULTIPLY, DML

Type:

Numeric

Description:

[DD2] x [DD1] --> [DD2]
(Multiplicand x Multiplier = Product)

Indicator Conditions:

OV, SF, G, L

Trap Conditions:

IS, IC, OV

7.3.1.4 DECIMAL DIVIDE, DDV

Type:

Numeric

Description:

$\frac{[DD2]}{[DD1]}$ --> [DD3]

Remainder --> [DD2]

Indicator Conditions:

OV, SF, G, L

Trap Conditions:

IS, IC, DZ, OV

Special Conditions:

1. OV is set if quotient is larger than DD3(L) or a divide by Zero is attempted.

2. Sign of quotient is determined by rules of algebra. Specifically:

   o If sign of [DD1] = that of [DD2], then quotient = (+).

   o If sign of [DD1] ≠ that of [DD2], then quotient = (-).

   o Sign of remainder is always equal to that of the dividend [DD2] unless the remainder is Zero.

3. If [DD1] > [DD2], then [DD2] --> [DD2] and 0 --> [DD3].

4. If divide by zero is attempted, then DZ Trap (TV25) and set CI(OV).

7.3.1.5 DECIMAL COMPARE, DCM

Type:

Numeric

Description:

[DD1] :: [DD2] --> Indicators
Compare takes place algebraically. Indicators will indicate if [DD1] is E, G, L than [DD2].

Indicator Conditions:

G, L

Trap Conditions:

IS, IC

Special Conditions:

1. Comparison is algebraic.

2. Leading Zero fill digits are supplied for the shorter operand.

3. (+) and (-) zero compares equal.

4. [DD1] and/or [DD2] may be IMOs.

7.3.1.6 DECIMAL MOVE AND CONVERT, DMC

Type:

Numeric

Description:

[DD1] is converted and moved to [DD2]

1. Valid combinations:

   String --> String
   String --> Packed
   Packed --> String
   Packed --> Packed

2. Data type specified by DD1 is converted to data type specified by DD2.

3. The result is right justified in the receiving field.

4. If DD2(L) > DD1(L), then the leftmost portion of the result is Zero-filled.

Indicator Conditions:

OV, SF, G, L

Trap Conditions:

IS, IC, OV

7.3.1.7 CONVERT BINARY TO DECIMAL, CBD

Type:

Numeric

Description:

[DD1] is converted and moved to [DD2]

1. Result is right justified.

2. DD1 is a Binary DD.

3. DD2 is a Decimal (String or Packed DD).

Indicator Conditions:

OV, SF

Trap Conditions:

IS, OV

Special Conditions:

1. If DD2(L) > the number of significant digits in the decidecimal equivalent of [DD1], then the leftmost position of DD2 is Zero-filled.

2. High order bit of Binary operand is its sign bit (2's complement notation).

3. If DD1 specifies an IMO, then unspecified results occur.

7.3.1.8 CONVERT DECIMAL TO BINARY, CDB

Type:

Numeric

Description:

[DD1] is converted and moved to [DD2]

1. Result is right justified.

2. DD1 is a decimal (String or Packed) DD.

3. DD2 is a binary DD.

4. Binary result is a 2's complement number, sign extended to fill [DD2].

Indicator Conditions:

OV

Trap Conditions:

IS, IC, OV

7.3.1.9 DECIMAL SHIFT, DSH

Type:

Numeric

Description:

Figure 7-5 gives the two possible formats of the shift instruction. Note that two shift control words are always required and they must be in line following DD1 or Label 1.

Special Conditions:

1. If DD1 specifies an IMO, an IS Trap results.

2. Illegal Character checking is performed only upon completion of the shift operation. Thus, any illegal characters shifted out do not affect the Illegal Character checking.

3. If d = 0, the instruction is treated as a no-operation instruction. Thus, there is no checking for illegal characters or sign normalization.

7.3.1.9.1 Decimal Shift - Shift Left, DSH

Type:

Numeric

Description:

For L/R = 0
Shift DD1 left "d" and append zeros to the right

Indicator Conditions:

OV, G, L

Trap Conditions:

IC, OV, IS

Special Conditions:

If a non-zero character is shifted out, set OV.

7.3.1.9.2 Decimal Shift - Shift Right, DSH

Type:

Numeric

Description:

For L/R = 1

Shift [DD1] Right "d" and round if specified. Zero fill to the left.

Indicator Conditions:

G, L

(Text continues after Figure 7-5)

Shift instruction with shift control information in line:

```
                               0              7 8             15
                               ____________________________________
                              |                 |                  |
                              | 0 0 0 0 0 0 0 0 | 0 0 1 0 1 1 1 0  |
                              |_________________|__________________|
                              |                                    |
                              |            DD1 or Label 1          |      Bit 7 of SCW1
                  __          |____________________________________|_     specifies that
                 |            |       |          | |               | |    shift control
Shift            | SCW1       |  RFU  | 0 0 0 0 |1| 0 1 1 1 1 0 0 0| |-   information is
Control         <             |_______|_________|_|________________|_|    in SCW2.
Words            |            | L |   |          |R|               |
                 | SCW2       | / |RFU|     d    |C|     R F U     |
                 |__          |_R_|___|__________|_|_______________|
                                0 1 2 3        7 8 9             15
```

OR

Shift instruction with shift control information in register R5:

```
                               0              7 8             15
                               ____________________________________
                              |                 |                  |
                              | 0 0 0 0 0 0 0 0 | 0 0 1 0 1 1 1 0  |
                              |_________________|__________________|
                              |                                    |
                              |            DD1 or Label 1          |      Bit 7 of SCW1
                  __          |____________________________________|_     specifies that
                 |            |       |          | |               | |    shift control
Shift            | SCW1       |  RFU  | 0 0 0 0 |0| 0 1 1 1 1 0 0 0| |-   information is
Control         <             |_______|_________|_|________________|_|    in register R5.
Words            |            |                                    |
                 | SCW2       |                R F U               |
                 |__          |____________________________________|
                                0 1 2 3        7 8 9             15


                  __           ____________________________________
                 |            |R|              |L|   |             |
CPU R5          <             |C|     R F U    |/|RFU|      d      |
                 |__          |_|______________|R|___|_____________|
                               0 1            7 8 9  11          15
```

where:

SCWn = Shift Control Word (SCW1 and SCW2) - They must be inline and take the position normally occupied by DD2. The SCWn is treated as a DD and thus the word must be encoded exactly as shown above. SCW1 bit 7 specifies where the shift control information is located. When the shift control information comes from R5, the left and right bytes of the shift control information are interchanged. The sign (if any) is unaffected by the shift operation (i.e., it does not get shifted).

Figure 7-5 Shift Instruction Formats (Sheet 1 of 2)

L/R = Specifies the direction of the shift operation. For L/R = 0, shift left; for L/R = 1, shift right.

d = Shift distance: $(0 \leq d \leq 31)$

RC = Round Control - This is applicable only to a shift right. For RC = 0, do not round; for RC = 1, round after shifting right.

Figure 7-5 Shift Instruction Formats (Sheet 2 of 2)

(Text continued from sheet 7-17)

Trap Conditions:

IC, IS

Special Conditions:

If rounding is performed, the following rules apply:

o Shift digits to the right

o If the last digit (Ld) shifted out equals: $0 \leq Ld \leq 4$, then the result remains unchanged.

o If the last digit (Ld) shifted out equals: $5 \leq Ld \leq 9$, then absolute value of the result is incremented by one (i.e., +12 --> +13; -12 --> -13.)

7.3.2 Alphanumeric Instructions

The following general rules apply to all Commercial alphanumeric instructions:

1. EA always points to the leftmost character of the operand.

2. If sending field length of an ALR or MAT instruction is shorter than receiving field length, then fill or not (as specified by DD2) the remaining characters, at the right end of the receiving field. ACM always fills (independent of DD2-C2).

3. During the execution of an alphanumeric compare instruction, the shorter field is always filled (as specified in DD2) to ensure that both operands have the same length.

4. When fill is specified by DD2, then DD2 also specifies the fill character:

   o Either an ASCII Blank, or

   o Character contained in [R5 (0:7)].

5. Operands with Zero length are allowed except in SRH and VRF instructions.

6. The TR indicator is set if the receiving field cannot accept all the characters of the result.

7. Operands must not overlap except as allowed by rule #8.

8. Identical operands are allowed.

9. If DD2 specifies an IMO in any instruction except an ACM, a Trap TV26 occurs.

10. If DD3 specifies an IMO, a Trap TV26 occurs.

11. The following defines the commercial alphanumeric instruction traps:

TRAP EVENT	MNEMONIC	TRAP NUMBER
Reference to protected memory	PV	14* or 32
Reference to unavailable resource	UR	15* or 23
Memory or bus error	BE	17* or 24
Illegal specification	IS	26
Truncation	TR	28

* This trap number is used when the commercial instruction is executed by the CPU as opposed to an auxiliary processor.

12. Whenever a trap occurs, the state of the indicators affected by the trapped instruction are unspecified.

13. If an IS Trap occurs, then the operands are preserved except for the edit instruction.

14. If the receiving field cannot accept all the characters of the result, then only the lefthand portion of the result is found in the receiving field (i.e., the right-most character(s) are discarded).

7.3.2.1 ALPHANUMERIC MOVE, ALR

Type:

Alphanumeric

Description:

[DD1] --> [DD2]

1. If DD2(L) = Zero, then instruction is aborted, the TR bit is set, and a TR Trap results.

2. If DD1(L) = Zero, then fill or not as specified by DD2.

Indicator Conditions:

TR

Trap Conditions:

IS, TR

Special Conditions:

1. For L $\neq$ Zero:

   DD1, $1 \leq L \leq 31$

   DD2, $1 \leq L \leq 31$ and fill character (if specified by C2) is an ASCII Blank (20 hexadecimal).

2. For L = Zero, then escape to [Rn (8:15)] for L:

   DD1, $0 \leq L \leq 255$

   DD2, $0 \leq L \leq 255$ and fill character (if specified by C2) is contained in [R5 (0:7)].

3. ALR instruction is interruptable (see subsection 3.5.2.3, H-bit).

7.3.2.2 ALPHANUMERIC COMPARE, ACM

Type:

Alphanumeric

Description:

[DD1] :: [DD2] —> Indicators

1. If DD1(L) ≠ DD2(L), then unconditionally extend the shorter field to the right with fill characters as specified by DD2. The extension of the shorter operand does not take place in main memory but only in the CSS.

2. If DD1(L) = DD2(L) = Zero, then compare is as for equal operands.

3. [DD1] is determined to be G or L than [DD2] by comparing characters left to right. When a non-compare results, then the binary values of the two unlike characters determine whether [DD1] is G or L than [DD2].

Indicator Conditions:

G, L

Trap Conditions:

IS

Special Conditions:

1. Indicators indicate that [DD1] is G, L, than [DD2].

2. [DD1] or [DD2] may be IMOs.

3. For L ≠ Zero:

   DD1, $1 \leq L \leq 31$

   DD2, $1 \leq L \leq 31$ and fill charracter is an ASCII blank (20 hexadecimal).

4. For L = Zero, then escape to [Rn (8-15)] for L.

   DD1, $0 \leq L \leq 255$
   DD2, $0 \leq L \leq 255$ and fill character is contained in [R5(0-7)].

5. ACM instruction is interruptable (see subsection 3.5.2.3, H bit).

7.3.2.3 ALPHANUMERIC MOVE AND TRANSLATE, MAT

Type:

Alphanumeric

Description:

[DD1] is translated and moved to [DD2].

DD3 specifies a 256 byte translate table.

Each character in [DD1] is used as a displacement from the base of the table defined by DD3 and the referenced character from the table is stored in [DD2].

1. If DD2(L) = Zero, then instruction is aborted and a TR Trap (TV28) results.

2. If DD1(L) = Zero, then fill or not as specified by DD2.

Indicator Conditions:

TR

Trap Conditions:

IS, TR

Special Conditions:

1. The CSS takes the length of [DD3] to be 256 bytes.

2. Fill character specified by DD2 is used directly (i.e., it is not translated).

3. For $L \neq$ Zero:

   DD1, $1 \leq L \leq 31$

   DD2, $1 \leq L \leq 31$ and fill character (if specified by C2) is an ASCII blank (20 hexadecimal).

   DD3, L is ignored and DD3(L) is assumed to be 256.

4. For L = Zero, then escape to [Rn(8:15)] for L:

   DD1, $0 \leq L \leq 255$

   DD2, $0 \leq L \leq 255$ and fill character (if specified by C2) is contained in [R5(0:7)].

   DD3, L is ignored and DD3(L) is assumed to be 256.

5. MAT instruction is interruptable (see subsection 3.5.2.3, H bit).

7.3.2.4 ALPHANUMERIC SEARCH, SRCH

Type:

Alphanumeric

Description:

Search the operand string defined by DD3 for a character sequence (substring) which matches any of one or more argument strings described by DD1. The operation terminates when the scan of the operand string is completed or a match is found. If a match is found, the indicator bits are set for comparison equality (G = 0, L = 0) and the two-word field addressed by DD2 is set to indicate both the operand substring position of the match and the identifier number index of the matched argument. If there is no match, the indicator bits are set to indicate inequality (G = 0, L = 1) and the field addressed by DD2 is left unaltered.

DD1, the descriptor for the argument list has the folloiwng significance:

If L = 1 through 31: Single argument string of length L

If L = 0: Register R4 contains the dimensions of a list (array) of argument strings. Bits 11 through 15 specifiy the total list length (up to 31 bytes) and bits 3 through 7 specify the length of each argument. The number of arguments is equal to the largest integer multiple of the argument length contained in the list length. Any residue is ignored.

DD3, the descriptor for the string being searched, denotes the following:

If L = 1 through 31: The operand is a field (string) of length L. All substrings equal in length to that of the argument(s) are match candidates.

If L = 0: Register R6 bits 8 through 15 specify the operand string length (up to 255 bytes). Register R6 bits 0 through 7 specify a value referred to as the operand element length or cursor increment. This coding of L in DD3 causes selected substrings of the operand to be tested for an argument match. The candidate substring selection is controlled by a hardware-managed cursor. During execution, the cursor contains the byte displacement (fom the left end of the operand) of the first (leftmost) byte of each candidate substring as it is tested. After each no-match scan of the argument list, the cursor is incremented by the value in bits 0 through 7 to select the succeeding candidate substring.

The operation proceeds under the control of two hardware maintained values: operand scan cursor and argument list index. Both are initially set to Zero which always, therefore, starts the search at the leftmost character of the operand string and the first argument string in the argument list. The argument index is incremented to select successive arguments for comparison against the operand string positions selected by the current setting of the cursor. If there is no match after the argument index has reached its maximum value (one less than the number of arguments since the range starts at Zero), the cursor is incremented by the scan cursor increment value to select a new operand substring. The argument index is reset to Zero and the arguments are scanned again for a match. The instruction is terminated when a match is found or the cursor is so far to the right that the remaining substring is either (1) shorter than the argument length or (2) shorter than the operand element length.

If a match occurs, the first word of the two addressed by DD2 is set to the index of the search argument which was matched. The second word is set to the value of the operand scan cursor which indicates the relative character position, in the operand, of the leftmost character of the matched substrings.

Depending on the relative magnitudes of the cursor increment and argument length, the following ralationships between substring of the operand are possible:

- o Cursor increment less than argument length - Tested operand substrings overlap.
- o Increment equal to argument length - Substrings are concatenated.
- o Increment greater than argument length - Untested characters embedded in operand string between tested substrings.

Indicator Conditions:

G, L

Trap Conditions:

IS

Special Conditions:

1. Unspecified results occur if argument length or search list length is not less than or equal to 31 (bytes).
2. IS trap (TV26) if argument longer than search list or cursor increment (operand element length) larger than operand string length.
3. An IS trap (TV26) is posted if argument string length or cursor increment is Zero.
4. For L $\neq$ Zero:

   DD1, $1 \leq L \leq 31$. This implies that SAL = 1 and $1 \leq L = SLL \leq 31$.

DD2, L is ignored and DD2(L) is assumed to be two words.

DD3, $1 \leq L \leq 31$. This implies that OEL = 1 and $1 \leq L = OL \leq 31$.

5. For L = Zero, then escape to Rn for L:

   DD1, $1 \leq L = SLL \leq 31$ in [R4(11:15)] and SAL is contained in [R4(3:7)].

   DD2, L is ignored and DD2(L) is assumed to be two words.

   DD3, $1 \leq L = OL \leq 255$ in [R6(8-15)] and OEL is contained in [R6(0-7)]

6. SRCH instruction is interruptable (see subsection 3.5.2.3, H bit).

NOTE

Due to the complexity of the Search instruction, examples are given in Appendix C. The following abbreviations are used: SA - search argument, SAL - search argument length, OEL - operand element length (cursor increment), OL - operand string length, and SLL - search list length.

7.3.2.5 ALPHANUMERIC VERIFY, VRF

Type:

Alphanumeric

Description:

Verify that each of the characters or selected substrings of the operand defined by DD3 is a member of the set of verify arguments contained in the verify list defined by DD1.

If at least one character or substring of the operand does not match any one of the verify arguments, then a nonequal indication is posted (G = 0, L = 1). Also, the word addressed by DD2 is set to the relative displacement (from the leftmost operand character) of the unmatched substring. If substrings of the operand are tested rather than individual characters, this value is the displacement of the leftmost character of the substring, not that of the specific character which could not be matched. If all tested operand items (characters or substrings) have a corresponding verify list entry, then an equal comparison indication (G = 0, L = 0) is set and the word addressed by DD2 is unaltered.

DD1, the descriptor for the verify argument list, has the following significance:

If L = 1 thorugh 31: Single argument string of length L.

If L = 0: Register R4 contains the dimensions of a list of verify arguments. Bits 11 through 15 specify the total list length (up to 31 bytes) and bits 3 through 7 specify the length of each argument. The number of arguments is equal to the largest integer multiple of the list length divided by the argument length. Any residue is ignored.

DD3, descriptor for the string being verified, denotes the following:

If L = 1 through 31: Operand string of length L. The scan cursor is incremented by one character position after each verify/match. All character positions are therefore tested for a corresponding character or substring entry in the argument list.

If L = 0: Bits 8 thorugh 15 of register R6 specify the operand string length (up to 255 bytes). Bits 0 through 7 specify a scan cursor increment value referred to as operand element length. As in the search instruction, the cursor indicates the left-most character position of the operand and substring to be verified. The cursor is incremented automatically after each verification to select the subsequent character or substring to be tested for an argument match. The number of characters tested for a single argument match is equal to the argument length.

As in the search instruction, scanning of the operand is under the control of a cursor which selects a character (verify argument length of one byte) or substring. After a match is found in the verify argument list, the cursor is incremented to select the next operand character(s) to be tested. The cursor is initialized to a character displacement of Zero (first character) and incremented so that verification always includes and starts at the first character and ends when a mismatch is found or an increment would cause the cursor to fall beyond the operand string end. If the cursor becomes positioned such that remaining characters of the operand would form a test compare result that is truncated shorter than the argument length, the verification fails. There is no padding of the operand during matching comparisons.

Indicator Conditions:

G, L

Trap Conditions:

IS

Special Conditions:

1. Verify argument list length must be equal to or less than 31 bytes else unspecified results occur.

2. Verify argument length (VAL) must be equal to or less than verify list length (VLL) and operand element length (OEL) (cursor increment) must be less than operand length (OL) else IS Trap (TV26).

3. If VAL = 0 or operand element length (OEL) = 0, then IS Trap (TV26).

4. If any of the length paramenters is zero, then IS Trap (TV26).

5. For L $\neq$ Zero

   DD1, $1 \leq L \leq 31$. This implies that VAL = 1 and $1 \leq L = VLL \leq 31$.

   DD2, L is ignored and DD2(L) is assumed to be one word.

   DD3, $1 \leq L \leq 31$. This implies that OEL = 1 and $1 \leq L = OL \leq 31$.

6. For L = Zero, then escape to [Rn(8-15)] for L:

   DD1, $1 \leq L = VLL \leq 31$ in [R4(11:15)] and SAL is contained in [R4(3:7)].

   DD2, L is ignored and DD2(L) is assumed to be one word.

   DD3, $1 \leq L = OL \leq 255$ in[R6(8:15)] and OEL is contained in [R6(0:7)]

7. VRF instruction is interruptable (see subsection 3.5.2.3, H bit).

NOTE

Due to the complexity of the verify instruction, examples are given in Appendix D.

7.3.3 Edit Instructions

Two types of edit instructions are supported:

- o Decimal Move and Edit (DME)
- o Alphanumeric Move and Edit (AME).

Their DDs are interpreted as follows:

- o DD1 = Sending field descriptor, and is:
  - A decimal DD for a DME with DD1(L) $\leq$ 31, or
  - An alphanumeric DD for a AME with DD1(L) $\leq$ 255.

- o DD2 = Receiving field descriptor (alphanumeric). DD2 (L) $\leq$ 255.
- o DD3 = Micro-op string descriptor (alphanumeric). DD3(L) $\leq$ 255.

Edit instructions are used to move and edit data from a sending field to a receiving field as specified by micro-op's contained in a micro-op string. The number of edited characters stored in [DD2] can be either less or more than contained in the sending field. Less characters result in the receiving field when one or more of the sending field characters are skipped over. More characters than were originally in the sending field result in the receiving field when one or more characters are inserted by the micro-ops.

Insertion characters normally come from an Edit Insertion Table (EIT) that is hardware managed and contains special characters useful during edit operations.

The editing is terminated normally when the receiving field length is exhausted. This normal termination occurs irrespective of the current string length of the sending field or the micro-op string or both. If the receiving field length is not exhausted, but either there are no more micro-ops or the sending field is exhausted, then an IS Trap (TV26) occurs.

7.3.3.1 EDIT INSTRUCTION DESCRIPTION

The following general rules apply to the edit instructions:

1. EA always points to the leftmost character of the operand.
2. All operations take place left to right.
3. Instructions terminate as described in subsection 7.3.3.
4. Plus or minus Zero sending fields are considered positive (i.e., SN flag is set Off).
5. Any operand having a Zero length causes IS Trap (TV26).
6. Sending field count defines the number of atoms in the sending field. It is decremented every time an atom in the sending field is either moved to the receiving field or skipped.
7. Receiving field count defines the number of byte positions in the receiving field. It is decremented every time a byte is moved into the receiving field.
8. The EIT is always initialized at Edit instruction initiation. See subsection 7.3.3.3 for EIT's initial state values.
9. The Edit Flags are always initialized at Edit instruction initiation. See subsection 7.3.3.4.
10. A digit is assumed to be equal to zero if its low order four bits are Zero.

11. The following defines the Edit instruction Traps:

TRAP EVENT	MNEMONIC	TRAP NUMBER
Reference to protected memory	PV	14* or 32
Reference to unavailable resource	UR	15* or 23
Memory or bus error	BE	17* or 24
Illegal specification	IS	26
Illegal character	IC	27

* This trap number is used when the commercial instruction is executed by the CPU as opposed to an auxiliary processor.

12. Micro-ops in the array subsequent to the one which results in the termination of the instruction are not necessarily subject to IS Trap condition checks.

13. Illegal character checks (for non-numeric values) are not performed on portions of DME sending fields which are ignored or unprocessed. Sign representations, however, must be valid.

7.3.3.1.1 Decimal Move And Edit, DME

Type:

Edit

Description:

[DD1] Edited and Moved to [DD2]

[DD3] specifies the micro-ops

1. DD1 is a numeric DD; $L \leq 31$

2. DD2 and DD3 are alphanumeric DDs; $L \leq 255$.

Indicator Conditions:

None.

Trap Conditions:

IS, IC

Special Conditions:

1. For L $\neq$ Zero:

   DD1, $1 \leq L \leq 31$
   DD2, $1 \leq L \leq 31$
   DD3, $1 \leq L \leq 31$.

2. For L = Zero, then length is given by:

   DD1, $1 \leq [R4(8:15)] \leq 31$
   DD2, $1 \leq [R5(8:15)] \leq 255$
   DD3, $1 \leq [R6(8:15)] \leq 255$.

3. DME instruction is interruptable (see subsection 3.5.2.3, H bit.).

7.3.3.1.2 Alphanumeric Move and Edit, AME

Type:

Edit

Description:

[DD1] Edited and moved to [DD2]
[DD3] specifies the micro-ops
All DDs are alphanumeric DDs; $L \leq 255$.

Indicator Conditions:

None.

Trap Conditions:

IS

Special Conditions:

1. For L $\neq$ Zero:

   DD1, $1 \leq L \leq 31$
   DD2, $1 \leq L \leq 31$.
   DD3, $1 \leq L \leq 31$.

2. For L = Zero, then length is given by:

   DD1, $1 \leq [R4(8:15)] \leq 255$
   DD2, $1 \leq [R5(8:15)] \leq 255$
   DD3, $1 \leq [R6(8:15)] \leq 255$

3. AME instruction is interruptable (see Subsection 3.5.2.3, H bit).

7.3.3.2 Micro-Ops

DD3 points to an array of eight-bit micro operations to be performed during an Edit instruction. Each eight-bit byte defines a micro operation and has the following formats:

```
 0   3 4   7    Bit Position Within a Byte
 _____________
|      |      |
| MOP  |  IF  |
|______|______|
 0   3 0   3    Bit Position Within a Field
```

where MOP = Micro-op; IF = Information Field

A micro-op defines the micro operation to be performed. The following micro-ops are supported:

BIT POSITIONS IN MOP FIELD				
b0 b1	b2 b3 0 0	b2 b3 0 1	b2 b3 1 0	b2 b3 1 1
0 0	CHT	ENF	IGN	INSA
0 1	INSB	INSM	INSN	INSP
1 0	MFLC	MFLS	RFU	RFU
1 1	MVC	MVZA	MVZB	SEF

The information field (IF) indicates one of the following, depending upon the micro-op:

1. The number of source digits to be manipulated (1-16).

2. The number of the particular entry in the Edit Insertion Table (EIT) to be used.

3. Which edit flags are to be set.

7.3.3.3 EDIT INSERTION TABLE

During the execution of an Edit instruction, a hardware table of eight 8-bit bytes holds insertion information. This table is called the Edit Insertion Table (EIT). It is initialized to the values given below at the start of each Edit instruction (each symbol shown refers to the corresponding standard ASCII character):

EIT Entry Number	1	2	3	4	5	6	7	8
Graphic Value	Ƀ	*	+	-	$	,	.	Ø
Hexadecimal Value	20	2A	2B	2D	24	2C	2E	30

where Ƀ = Blank; Ø = The digit Zero.

One or all of the table entries can be changed by the Change Table (CHT) micro-op to provide different insertion character(s).

7.3.3.4 EDIT FLAGS

The following edit flags are provided for recording or setting or both (as "ON" or "OFF") certain specific "editing condition." The current status of some of these edit flags can be interrogated by conditional micro-ops, thereby providing a mechanism for alternative editing operations. The status of the other edit flags is interrogated at the termination of the micro-op sequence, thereby causing certain "post-edit/editing" operations to be performed.

1. The End Suppression (ES) Flag - The ES flag is initially set "OFF". It is typically set "ON" when Zero suppression ends (e.g., when a non-zero digit is processed from the sending field). The ES flag can be interrogated and set "ON" or "OFF" by certain micro-ops (e.g., SEF).

2. The Sign (SN) Flag - The SN flag is initially set "OFF" if the sending field has an alphanumeric descriptor (i.e, in an AME).

   If the sending field has a numeric descriptor (i.e., in an DME), the sign of the sending field is initially "read" from the sending field according to the sign type information in the sending field descriptor, and:

   o SN is initially set "OFF" if the sending field is positive, unsigned or Zero.

   o SN is initially set "ON" if the sending field is negative and non Zero.

   The SN flag can be interrogated by certain micro-ops, but cannot be altered by any micro-op.

When an operand specifies a separate sign, the sign is is skipped (i.e., it is not considered to be part of the sending field). Similarly, when an overpunch sign is specified, the overpunch is converted into a decimal digit (i.e., sign information is removed).

3. The Zero (Z) Flag - The Z flag is initially set "ON". It is automatically set "OFF" whenever a non-zero atom from a sending field is moved to the result field. (Once the Z flag has been set "OFF", it remains "OFF" for the duration of the Edit instruction.)

   The Z flag can be neither interrogated nor altered by any micro-op. The Z flag is interrogated and resulting alternative editing can occur in the post-edit/edit (see BZ and AZ Flags).

4. The Blank-When-Zero (BZ) Flag - The BZ flag is initially set "OFF". It can be set "ON" by either the ENF (End Floating Suppression) or SEF (Set Edit Flags) micro-op. (Once set "ON", the BZ flag remains so for the duration of the Move/Edit instruction.)

   If, at the termination of the micro-op sequence, BOTH the Z and BZ flags are "ON", then the ENTIRE result field is force-filled with the contents of EIT (1) - that is, Edit Insertion Table entry one [which normally contains a blank ("b̸") character].

   Note that this "post-edit/editing" operation completely "blanks out" whatever character string had been move/edited into the result field by "executing" the micro-op sequence.

5. The Asterisk-When-Zero (AZ) Flag - The AZ flag is initially set "OFF". It can be set "ON" by the SEF micro-op.

   If, at the termination of the micro-op sequence, the Z Flag is "ON" AND the BZ flag is "OFF" AND the AZ flag is "ON," then the ENTIRE result field EXCEPT any character corresponding to EIT 7 (normally a decimal point, ".") will be forcefilled with the contents of EIT 2 (which normally contains an asterisk ("*").

   The "post-editing/editing" operation completely fills with asterisks (*) except for the character currently in EIT 7 - the character string that had been move/edited into the result field by "executing" the micro-op sequence. The BZ and AZ flags (hence, the blank-when-Zero and the asterisk-when-Zero operation) are mutually exclusive. Further, the BZ flag is interrogated before the AZ flag.

6. The Plus/Minus (PM) Flag - The PM flag is initially set "OFF." It can be set "ON" by the SEF micro-op.

   The PM flag can be interrogated by the floating (MFLS) and fixed (INSP) sign insertion micro-ops as well as the End Floating Suppression (ENF) micro-op.

   If the PM flag is "ON", then the floating sign micro-ops (MFLS, ENF) and the insert on positive fixed sign insertion micro-op (INSP) operate as follows (refer to subsection 7.3.3.6.2):

- o If the SN flag is "OFF", then float or insert the edit insertion character contained in EIT(3) - normally a plus sign ("+").

- o If the SN flag is "ON", then the float or insert the insertion character contained in EIT (4) - normally a minus sign ("-").

If the PM flag is "OFF", then the floating sign micro-ops (MFLS, ENF) and BOTH fixed sign insertion micro-ops (INSN and INSP) utilize:

- o EITHER - The appropriate non-blank edit insertion character (i.e., either the appropriate sign EIT (3) or EIT (4) or the approariate credit or debit insertion character from the micro-op string);

- o OR - The blank edit insertion character from EIT (1); depending on which micro-op is used and the setting of the SN flag (see the write-up on the particular micro-ops for specific details).

7.3.3.5 DIFFERENCE BETWEEN A DME AND AN AME

The only differences between a DME and a AME are:

1. In the type of the sending field; where:

   a. For DME, DD1 is a decimal DD.
   b. For AME, DD1 is an alphanumeric DD; and

2. In the intial state of the edit flags as specified in subsection 7.3.3.4.

Micro-op execution is independent of instruction type.

7.3.3.6 MICRO-OP DESCRIPTION

Descriptions of the micro-ops (MOPs) follows. The mnemonic and function performed is given for each MOP.

7.3.3.6.1 CHT (Change Table)

The EIT is modified as specified by the IF field. If a specific EIT entry is to be modified, then use the 8-bit byte immediately following the CHT MOP and do not execute it as a micro-op. If all the EIT entries are to be modified, then use the eight 8-bit bytes immediately following the CHT MOP and do not execute them as micro-ops.

IF CODE	OPERATION
0	Replace all 8 EIT Entries
1	Replace EIT Entry 1
2	Replace EIT Entry 2
3	Replace EIT Entry 3
4	Replace EIT Entry 4
5	Replace EIT Entry 5
6	Replace EIT Entry 6
7	Replace EIT Entry 7
8	Replace EIT Entry 8
9 - F	IS Trap (TV26)

7.3.3.6.2 ENF (End Floating Suppression)

The ENF micro-op is used to perform two functions as determined by the bits 0 and 1 of the IF:

ENF TABLE

ES	PM	SN	IF(0)	INSERTION CHARACTER
1	X	X	X	NONE
0	0	0	0	EIT(1) ƀ
0	X	1	0	EIT(4) -
0	X	X	1	EIT(5) $
0	1	0	0	EIT(3) +

1. The main function is to terminate either leading Zero suppression or floating insertion and to force the insertion of either the appropriate sign or currency symbol. IF(0) specifies the type of insertion required:

   a. IF(0) = 0, then do sign insertion;
   b. IF(0) = 1, then do currency symbol insertion.

2. The other function is either to set or not to set the BZ flag. IF(1) defines the following:

   a. IF(1) = 0, then do not set BZ;
   b. IF(1) = 1, then set BZ.

   Insertion is performed as a function of the state of the ES flag, the PM flag, the SN flag and bit 1 of IF:

   o For ES = OFF, then insertion if required as a function of the setting of IF(0), SN and PM.

   o For ES = ON, then no insertion is required.

   The following defines the insertion byte required when ES is OFF and PM is OFF:

   o For IF(0) = 0, SN determines the insertion byte:

      a. If SN = OFF, the EIT(1) is moved into the receiving field.

      b. If SN = ON, the EIT(4) is moved into the receiving field.

   o For IF(0) = 1, then EIT (5) is moved into the receiving field.

   The following defines the insertion byte required when ES is OFF and PM is ON:

   o For IF(0) = 0, SN determines the insertion byte:

      a. If SN = OFF, then EIT(3) is moved into the receiving field.

      b. If SN = ON, then EIT(4) is moved ito the receiving field.

   o For IF(0) = 1, then EIT(5) is moved into the receiving field.

   Whenever insertion takes place, ES is set ON.

   The sending field count is unaffected by the ENF micro-op and the receiving field count then:

   o Remains unaltered if no insertion is performed, or

   o Is decremented by one if insertion is performed.

7.3.3.6.3 IGN (Ignore Source Character)

The IGN micro-op is used to skip over n bytes of the sending field. The sending field count is reduced accordingly and no change is made to the receiving field count.

IF specifies the number of bytes to be ignored where:

$$IF = n, \text{ for } 1 \leq n \leq 15 \text{ and } IF = 0 \text{ specifies } n = 16.$$

7.3.3.6.4 INSA (Insert Asterisk On Suppress)

The INSA micro-op is used to insert a byte into the receiving field as a function of:

1. The state of ES flag, and

2. The value of IF.

The sending field count remains unchanged while the receiving field count is decremented by one.

ES determines the following:

- o For ES = OFF, then insert into the receiving field EIT(2) (normally an asterisk).

- o For ES = ON, then insert into the receiving field the insertion character specified by IF.

IF determines the following:

- o The insertion byte, when ES = ON:

  a. For IF = 0, insert the byte immediately following the INSA micro-op.

  b. For IF = 1-8, take the insertion byte from the EIT entry defined by IF.

  c. For IF > 8, set IS Trap (TV26).

- o Where the next micro-op is located:

  a. For IF = 0, the next micro-op is at INSA address +2 since INSA address +1 contains the insertion byte.

  b. For IF ≠ 0, the next micro-op follows INSA.

7.3.3.6.5 INSB (Insert Blank On Suppress)

The INSB micro-op is used to insert a byte into the receiving field. The same logic applies to this micro-op as defined for the INSA micro-op except that for ES = OFF, insert into the receiving field EIT(1) (normally a blank).

7.3.3.6.6 INSM (Insert Table Entry 1 Multiple)

The INSM micro-op is used to insert n EIT Entry 1 bytes into the receiving field. The sending field count remains unchanged while the receiving field count is decremented by n.

IF specifies the number of EIT(1) bytes to be inserted into the receiving field where:

- o IF n for $1 \leq n \leq 15$, and
- o IF = 0 specifies n = 16.

7.3.3.6.7 INSN (Insert on Negative)

The INSN micro-op is used to insert a byte into the receiving field as a function of:

1. The state of the SN flag and
2. The contents of IF.

Therefore, the sending field count remains unchanged while the receiving field count is decremented by one.

- o For SN = OFF, insert into the receiving field EIT(1) (normally a blank).
- o For SN = ON, insert into the receiving field the insertion byte specified by IF:

  a. For IF = 0, the insertion byte immediately follows the INSN micro-op.

  b. For IF = 1-8, the insertion byte is taken from the EIT entry defined by IF.

  c. For IF > 8, set IS.

  d. For IF = 0, the next micro-op is at INSN address +2 since INSN address +1 contains the insertion byte.

  e. For IF $\neq$ 0, the next micro-op follows INSN.

7.3.3.6.8 INSP (Insert on Positive)

The INSP micro-op is used to insert a byte into the receiving field as a function of:

- o The state of the SN flag
- o The state of the PM flag
- o The contents of IF.

Therefore, the sending field count remains unchanged while the receiving field count is decremented by one.

```
 0   3 4   7  _ _ 8          15
 |     |     |    |            |
 | MOP | IF  |    |    DATA    |     DD3 POINTER
 |_____|_____| _ _|____________|
```

MOP = Micro-op
 IF = Information Field.

(Reference subsection 7.3.3.2)

INSP TABLE			
PM	SN	IF	INSERTION CHARACTER
0	0	0	Data (DD3 Second Byte)
0	0	1-8	EIT (1-8)
0	0	9-F	Set Trap (TV26)
0	1	X	EIT(1) b
1	0	X	EIT(3) +
1	1	X	EIT(4) -

1. For PM = OFF, the following applies:

   a. For SN = OFF, then insert the insertion byte specified by IF into the receiving field.

   b. For SN = ON, then insert EIT(1) (normally a blank) into the receiving field.

2. For PM = ON, the following applies:

   a. For SN = OFF, then insert EIT(3) (normally a "+") into the receiving field.

   b. For SN = ON, then insert EIT(4) (normally a "-") into the receiving field.

IF determines the following:

o The insertion byte when PM = SN = OFF:

a. For IF = 0, the insertion byte immediately follows the INSP micro-op.

b. For IF = 1-8, the insertion byte is taken from the EIT entry defined by IF.

c. For IF > 8, an IS Trap (TV26) is set.

o Where the next micro-op is located:

a. or IF = 0, the next micro-op is at INSP address +2 since INSP address +1 contains the insertion byte.

b. For IF $\neq$ 0, the next micro-op follows INSP.

7.3.3.6.9 MFLC (Move With Float Currency Symbol Insertion)

The MFLC micro-op is used to float the appropriate Currency Symbol over a specified number of sending field digits as a function of the ES flag. The Currency digit is four bits in size for packed decimal; otherwise, it is one byte long.

IF specifies the number of sending field digits upon which the operation is to be performed, where:

o IF = n for $1 \leq n \leq 15$, and

o IF = 0 specifies n = 16.

Starting with the next available sending field digit, the next n digits are individually fetched and the following conditional actions occur:

1. For ES = OFF:

a. If sending digit is Zero, then move EIT(1) to receiving field in place of Zero. The ES flag remains OFF.

b. If digit is not a Zero, then move EIT(5) (currency symbol) to the receiving field followed by the non-Zero atom. The ES flag is set ON.

2. For ES = ON, just move the digits to the receiving field.

The number of bytes moved to the receiving field is data dependent:

o If ES = OFF and the entire sending field equals Zero, then n EIT (1) bytes are moved to the receiving field. ES remains OFF.

o If ES = ON at micro-op initiation, then n digits are moved to the receiving field.

- o If the sending field contains at least one leading Zero and one non-Zero digit, then n+1 bytes are moved to the receiving field. ES is set ON.

At the completion of the MFLC micro-op, the sending field count is decremented by n while the receiving field count is decremented by either n or n+1.

7.3.3.6.10 MFLS (Move With Float Sign Insertion)

The MFLS micro-op is used to float the appropriate Sign Symbol over the specified number of sending field digits as a function of:

- o The ES flag
- o The SN flag
- o The PM flag.

The sign digit is four bits for packed desimal data; otherwise, it is one byte long.

IF specifies the number of sending field digits upon which the operation is performed, where:

IF = n for $1 \leq n \leq 15$ and

IF = 0 specifies n = 16.

Starting with the next available sending field digit, the next n digits are individually fetched and the following conditional actions occur:

MFLS TABLE

ES	PM	SN	SENDING DIGIT	INSERTION CHARACTER
0	X	X	= 0	EIT(1) b
0	0	0	≠ 0	EIT(1) b
0	X	1	≠ 0	EIT(4) -
0	1	0	≠ 0	EIT(3) +

1. For ES = OFF:

   a. If digit is a Zero, then move EIT(1) to receiving field in place of Zero. If the entire sending field equals Zero, then n EIT(1) bytes are moved to the recieving field (ES remains OFF).

   b. If digit is not a Zero, then the SN and PM flags determine the sign to be inserted:

      o For PM = OFF:

        - and SN = OFF, insert EIT(1);

        - and SN = ON, insert EIT(4).

      o For PM = ON:

        - and SN = OFF, insert EIT(3);

        - and SN = ON, insert EIT(4).

   After inserting the sign, also move the non-Zero digit to the receiving field and set the ES flag ON.

2. For ES = ON, just move the digits to the receiving field. The number of digits moved to the receiving field is data dependent.

   a. If ES = ON at micro-op initiation, then n digits are moved to the receiving field.

   b. If the sending field contains at least one leading Zero and one non-Zero digit, then n+1 digits are moved to the receiving field. ES is set ON.

At the completion of the MFLS micro-op, the sending field count is decremented by n while the receiving field count is decremented by either n or n+1.

7.3.3.6.11 MVC (Move Source Character)

The MVC micro-op is used to move n sending field digits to the receiving field.

IF specifies the number of sending field digits upon which the operation is to be performed, where:

IF = n for $1 \leq n \leq 15$ and

IF = 0 specifies n = 16.

At the completion of the MVC micro-op both the sending and receiving field counts are decremented by n.

7.3.3.6.12 MVZA (Move With Zero Suppression and Asterisk Replacement)

The MVZA micro-op is used to replace Zero digits with asterisks over a specified number of sending field digits as a function of the ES flag.

IF specifies the number of sending field digits upon which the operation is to be performed, where:

IF = n for $1 \leq n \leq 15$ and

IF = 0 specifies n = 16.

Starting with the next available sending field digit, the next n digits are individually fetched and the following conditional actions occur:

1. For ES = OFF:

   a. If digit is a Zero, then move EIT(2) to receiving field in place of Zero digit;

   b. If digit is not Zero, then the digit is moved to the receiving field. The ES flag is set ON.

2. For ES = ON, just move the digit to the receiving field.

At the completion of the MVZA micro-op, both the sending and receiving field counts are decremented by n.

7.3.3.6.13 MVBZ (Move With Zero Suppression and Blank Replacement)

The MVZB micro-op is used to replace Zero digits with blanks over a specified number of sending field digits. The same logic applies to this micro-op as defined for MVZB micro-op except that if ES = OFF and the digit is Zero, then move EIT(1) to the receiving field.

7.3.3.6.14 SEF (Set Edit Flags)

The function of the SEF micro-op is to "control" the following edit flags:

o ES (End Suppression)

o BZ (Blank When Zero)

o AZ (Asterisk When Zero)

o PM (Plus/Minus).

With this micro-op, the IF field represents a four bit binary mask where:

o Bit 0 of the IF field, IF(0), specifies the ES flag

o Bit 1 of the IF field ,IF(1), specifies the BZ flag

o Bit 2 of the IF field ,IF(2), specifies the AZ flag

o Bit 3 of the IF field ,IF(3), specifies the PM flag

As a function of each IF bit, the flow shown in Figure 7-6 occurs.

```
         *******
         * START *
         *******
            |
            v
          _____
         /     \     =1    _________
        < IF(0) >------->|  1 --> ES |-----
         \__?__/          |_________|     |
            |    =0                        |
            v                              |
       ___________                         |
      |           |                        |
      |  0 --> ES |                        |
      |___________|                        |
            |                              |
            |<------------------------------
            v
          _____
         /     \     =1    _________
        < IF(1) >------->|  1 --> BZ |-----
         \__?__/          |_________|     |
            |    =0                        |
            v                              |
          _____                            |
         /     \     =1    _________       |
        < IF(2) >------->|  1 --> AZ |--->|
         \__?__/          |_________|     |
            |    =0                        |
            |<------------------------------
            v
          _____
         /     \     =1    _________
        < IF(3) >------->|  1 --> PM |-----
         \__?__/          |_________|     |
            |    =0                        |
            |<------------------------------
            v
          *****
          * END *
          *****
```

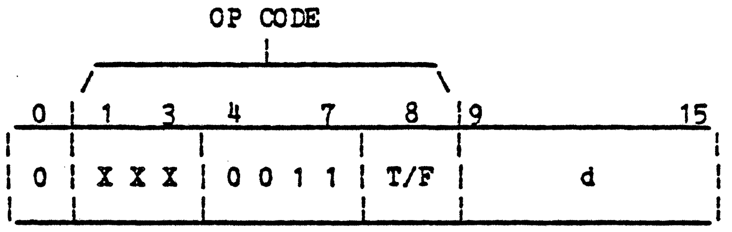
Figure 7-6 Set Edit Flags (SEF) Micro-Op Flow

The SEF micro-op has no effect on the sending and receiving field counts.

7.3.4 Branch Instructions

The Commercial branch instructions are a subset of the general branch on indicator instructions. The format of these instructions is shown in Figures 7-7.

These instructions enable branching on various indicator conditions. They are summarized in Table 7-3 and their numerical representation is given in Table 7-4.



where:

- OP = Opcode
- xxx = Specifies which CI bit is to be tested
- T/F = True or False
- d = Displacement - defines how to compute the EA. Refer to Figure 5-2.

Figure 7-7 Commercial Branch Instruction Format

Table 7-3 Commercial Branch Instruction Summary (Sheet 1 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
7.3.4.1	CBOV	Branch on Overflow	If [CI(OV)] = 1, then [P] <-- EA		
7.3.4.2	CBNOV	Branch on No Overflow	If [CI(OV)] = 0, then [P] <-- EA		See sheet 2
7.3.4.3	CBTR	Branch on Truncation	If [CI(TR)] = 1, then [P] <-- EA		

Table 7-3 Commercial Branch Instruction Summary (Sheet 2 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
7.3.4.4	CBNTR	Branch on No Truncation	If [CI(TR)] = 0, then [P] <-- EA		
7.3.4.5	CBSF	Branch on Sign Fault	If [CI(SF)] = 1, then [P] <-- EA		
7.3.4.6	CBNSF	Branch on No Sign Fault	If [CI(SF)] = 0, then [P] <-- EA		In all branch operations if the branch condition is true (i.e., branch is executed) and if M1(J) = 1 then trap to TV02.
7.3.4.7	CSYNC	Sync	See Text		
7.3.4.8	CSNBC	Sync and Branch	See Text		
7.3.4.9	CBE	Branch on Equal	If [CI(L)] \/ [CI(G)] = 0, then [P] <-- EA		
7.3.4.10	CBNE	Branch on Not Equal	If [CI(L)] \/ [CI(G)] = 1, then [P] <-- EA		
7.3.4.11	CBG	Branch on Greater than	If [CI(G)] = 1 then [P] <-- EA		

Table 7-3 Commercial Branch Instruction Summary (Sheet 3 of 3)

REFERENCE SUBSECTION	MNEMONIC	DESCRIPTION	OPERATION	INDICATORS AFFECTED	COMMENTS
7.3.4.12	CBLE	Branch on Less than or Equal	If [CI(G)] = 0 then [P] <— EA		
7.3.4.13	CBL	Branch on Less than	If [CI(L)] = 1 then [P] <— EA		See sheet 2
7.3.4.14	CBGE	Branch on Greater than or Equal	If [CI(L)] = 0 then [P] <— EA		

Table 7-4 Numerical Representation of Commercial Branch Instructions

SUBSECTION	H1	H2	H3 H4	MNEMONIC	ATOM SIZE
7.3.4.1	1	3	0 + d	CBOV	
7.3.4.2	1	3	8 + d	CBNOV	
7.3.4.3	2	3	0 + d	CBTR	NOT
7.3.4.4	2	3	8 + d	CBNTR	
7.3.4.5	3	3	0 + d	CBSF	
7.3.4.6	3	3	8 + d	CBNSF	APPLICABLE
7.3.4.7	4	3	0 + d	CSYNC	
7.3.4.8	4	3	8 + d	CSNBC	
7.3.4.9	5	3	0 + d	CBE	
7.3.4.10	5	3	8 + d	CNBE	
7.3.4.11	6	3	0 + d	CBG	
7.3.4.12	6	3	8 + d	CBLE	
7.3.4.13	7	3	0 + d	CBL	
7.3.4.14	7	3	8 + d	CBGE	

7.3.4.1 BRANCH ON OVERFLOW, CBOV

Type:

Branch

Description:

Branch to EA if CI(OV) is ON; i.e., indicating overflow.

Operation:

If [CI(OV)] = 1, then [P] <— EA

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBOV instruction is executed.

7.3.4.2 BRANCH ON NO OVERFLOW, CBNOV

Type:

Branch

Description:

Branch to EA if CI(OV) is OFF, that is, indicating no overflow.

Ooeration:

If [CI(OV)] = 0, then [P] <— EA

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBNOV instruction is executed.

7.3.4.3 BRANCH ON TRUNCATION, CBTR

Type:

Branch

Description:

Branch to EA if CI(TR) is ON; i.e., indicating truncation.

Operation:

If [CI(TR)] = 1, then [P] <— EA

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBTR instruction is executed.

7.3.4.4 BRANCH ON NO TRUNCATION, CBNTR

Type:

Branch

Description:

Branch to EA if CI(TR) is OFF, that is, indicating no truncation.

Operation:

If [CI(TR)] = 0, then [P] <— EA

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBNTR instruction is executed.

7.3.4.5 BRANCH ON SIGN FAULT, CBSF

Type:

Branch

Description:

Branch to EA if CI(SF) is ON; i.e., indicating a sign fault.

Operation:

If [CI(SF)] = 1, then [P] <— EA

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBSF instruction is executed.

7.3.4.6 BRANCH ON NO SIGN FAULT, CBNSF

Type:

Branch

Description:

Branch to EA if CI(SF) is OFF; i.e., indicating no sign fault.

Operation:

If [CI(SF)] = 0, then [P] <— EA

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBNSF instruction is executed.

7.3.4.7 COMMERCIAL SYNC, CSYNC

Type:

Branch

Description:

Wait for completion of the previous CIP instruction (if necessary) before going to the next instruction. No operation is performed.

Operation:

As above.

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

This instruction never causes a branch, so consequently the displacement d, given in bits 9 to 15 of the instruction, is used only to determine the size of the instruction (see subsection 7.3.4).

7.3.4.8 SYNC AND BRANCH, CSNCB

Type:

Branch

Description:

Wait for completion of the previous CIP instruction (if necessary), then unconditionally branch to EA.

Operation:

As above.

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CSNCB instruction is executed.

7.3.4.9 BRANCH ON EQUAL, CBE

Type:

Branch

Description:

Branch to EA if CI(L) = CI(G) = Zero; i.e., indicating that [DD1] = [DD2] in the previous compare or the result was Zero in the previous numeric instruction.

Operation:

If [CI(L)] \/ [CI(G)] = 0, then [P] <-- EA.

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBE instruction is executed.

7.3.4.10 BRANCH ON NOT EQUAL, CBNE

Type:

Branch

Description:

Branch to EA if either CI(L) = 1 or CI(G) = 1; i.e., indicating that [DD1] ≠ [DD2] in the previous compare or the result was not Zero in the previous numeric instruction.

Operation:

If [CI(L)] \/ [CI(G)] = 1, then [P] <-- EA.

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBNE instruction is executed.

7.3.4.11 BRANCH ON GREATER THAN, CBG

Type:

Branch

Description:

Branch to EA if CI(G) = 1; i.e., indicating that [DD1] > [DD2] in the previous compare or the result was greater than Zero in the previous numeric instruction.

Operation:

If [CI(G)] = 1, then [P] <-- EA.

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBG instruction is executed.

7.3.4.12 BRANCH ON LESS THAN OR EQUAL, CBLE

Type:

Branch

Description:

Branch to EA if CI(G) = 0; i.e., indicating that [DD1] $\leq$ [DD2] in the previous compare or the result was less than or equal to Zero in the previous numeric instruction.

Operation:

If [CI(G)] = 0, then [P] <-- EA.

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBLE instruction is executed.

7.3.4.13 BRANCH ON LESS THAN, CBL

Type:

Branch

Description:

Branch to EA if CI(L) = 1; i.e., indicating that [DD1] < [DD2] in the previous compare or the result was less than Zero in the previous numeric instruction.

Operation:

If [CI(L)] = 1, then [P] <-- EA.

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBL instruction is executed.

7.3.4.14 BRANCH ON GREATER THAN OR EQUAL, CBGE

Type:

Branch

Description:

Branch to EA if CI(L) = 0; i.e., indicating that [DD1] $\geq$ [DD2] in the previous compare or the result was greater than or equal to Zero in the previous numeric instruction.

Operation:

If [CI(L)] = 0, then [P] <-- EA.

Indicator Conditions:

OV Unchanged
TR Unchanged
SF Unchanged
G Unchanged
L Unchanged.

Special Conditions:

If the branch condition is true (that is, a branch is executed) and M1(J) = 1, then Trace trap (TV02) occurs when the CBGE instruction is executed.

SECTION 8 SCIENTIFIC INSTRUCTIONS

8.1 INTRODUCTION

This section describes the scientific instruction set. These instructions are part of the CSS instruction set and fall under the double and single operand and branch groupings.

Scientific instructions are classified as follows:

- o Double Operand (subsection 8.2)
- o Single Operand (subsection 8.3)
- o Scientific Branch (subsections 8.4 and 8.5)
- o Intrinsics (subsection 8.6). (These instructions are available only in the M6X and M6XE).

The double and single operand instructions use an Address Syllable (AS) field that is summarized in subsection 8.1.1 and described in detail in subsection 3.11.

Intrinsic instructions also use an Address Syllable (AS) but are restricted to a subset of Map 1 (reference subsection 8.6).

For convenience the instructions are summarized in Appendix H. They also appear in Appendix E as a function of their format.

8.1.1 Address Syllable

The single and double operand instructions generate operand references through a field called the Address Syllable (AS). The resolution of the AS usually results in the formulation of an Effective Address (EA) that points to an operand. For scientific instructions, the AS can take one of the following forms:

- o Register AS (RAS)
- o Immediate Operand AS (IMO)
- o Memory AS (MAS)

Note that:

- o Scientific Intrinsic instructions can only use AS entries Bn and registers (SA1, 2, 3) of AS Map 1 and cannot use AS Map 2 or 3.

- o All other scientific instructions can use AS Maps 1, 2, and 3 with the exception of IMO and AS3 entries in Map 2.

For more information about the AS, refer to subsection 3.11.

8.1.1.1 REGISTER AS (RAS)

The source or destination of the operand is a register (see subsection 3.11.1).

8.1.1.2 IMMEDIATE OPERAND AS (IMO)

The source or destination of the operand is a double or quadruple word location in main memory following the Address Syllable word. The memory length control bit corresponding to the selected SA (contained in M4, see 3.3.4.8) determines the length of an IMO for double-operand instructions.

8.1.1.3 MEMORY AS (MAS)

The source or destination of the operand is a double or quadruple word location in main memory. The AS specifies the EA. The memory length control bit corresponding to the selected SA (contained in M4, see 3.3.4.8) determines the length of a memory operand for double-operand instructions. The op code determines the length of a memory operand for single-operand instructions.

8.1.2 Normalization and Equilization

Normalization of a floating point number involves the shifting of the fraction (mantissa) to the left until its high-order hexadecimal digit is no longer Zero, and at the same time decrementing the exponent value for each digit shift. As the mantissa is shifted to the left, a Zero value is moved into the least significant digit position.

Floating point numbers are normalized when loaded into scientific accumulators by Scientific Load and Scientific Swap instructions. Any temporary copy of an operand, created for usage during a Scientific Compare instruction, will be converted into a floating point number and normalized if necessary; however, the original operand is left unchanged.

Integer numbers are converted to floating point numbers and are normalized when transferred to SA registers or prior to entering into scientific calculations.

Floating point results of scientific arithmetic operations are normalized prior to being stored in a destination SA. As the mantissa is shifted to the left, two guard digits and a Zero value are moved into the least significant digit position.

- If an exponent underflow condition is detected and the Exponent Underflow Mask M5 (EUF) is on, the normalized result with an exponent 128 too large is stored and a trap signal is posted.

- If an exponent underflow condition is detected and the M5 (EUF) is off, a clean Zero result is stored and no trap signal is posted.

Normalization of a fuzzy Zero produces a clean Zero (refer to subsection 3.1.3.1).

All other floating point numbers entering into arithmetic operation are not prechecked as to whether they are normalized.

Scaling right is the process of shifting floating point mantissa digits one hexadecimal position to the right, inserting a hex Zero or One digit into the most significant mantissa digit position, and increasing the exponent by One. This process can cause an exponent overflow condition; that is, the exponent is incremented to a number 128 less than the correct value. If the scaling right of the result of a scientific operation produces an exponent overflow, the normalized result with an exponent 128 too small is transferred to the designated register and an EOF trap posted.

Equalization is a scaling right process in which the exponent of the operand with the smaller exponent is incremented until it equals the exponent of the larger exponent of the two operands entering into a floating point addition or subtraction operation.

8.1.3 Mixed Mode Operands

Main memory operands used by scientific instructions are assumed to be in floating point format. Operands in registers (e.g., R, K) are assumed to be in integer format. Integer operands are converted within the CSS to floating point values prior to entering into a scientific calculation. See Figure 8-1.

The process for converting an integer to a floating point number includes:

1. If the integer value is plus or minus Zero, set the floating point exponent, sign, and mantissa fields to Zero (clean Zero).

2. If the integer is not Zero, convert the integer to a floating point sign and 14-digit mantissa value, right justified. Set the exponent field to +14. Normalize the floating point number and place it in the designated SA register. Low-order digit(s) are truncated/rounded if the accumulator length as determined by the mode register is less than the converted floating point number (i.e., loading a double integer operand into a single precision floating point register). No significance error or precision error conditions are sensed during an integer to floating conversion.

If the address syllable specifies Rn or Kn in a Scientific Store or Scientific Swap Instruction, the CSS converts the floating point value found in the source SA register prior to storing the integer in the designated register.

The process for converting a floating point number to an integer value includes:

1. An assumption is made that the floating quantity is a normalized number.

2. If the true value of the source SA register exponent is less than +1 (after mantissa rounding/truncating), set the integer to Zero.

3. If the integer fits in the destination register, then it is stored; that is, a value equal to or greater than Zero and not greater than four hex digits (including sign bit) if a single integer register, or not larger than eight hex digits (including sign bit) if a double integer register (after two's complementing if the floating number sign is negative).

4. If the integer does not fit in the destination register, set the SI(SE); that is, a value larger than four hex digits (including sign bit) if a single integer register, or larger than eight hex digits (including sign bit) if a double integer register (after two's complementing if the floating number sign is negative).

   o If the Significant Error Mask M5 (SEM) is Zero, store the largest two's complement integer; that is, positive value - 7FFF (hex), negative value - 8000 (hex) for a single integer, and positive value - 7FFF FFFF (hex), negative value - 8000 0000 (hex) for a double intger, with a sign bit and post no trap.

   o If the Signficant Error Mask M5 (SEM) is One, post a trap.

     - If M4(R/T) is Zero, truncate the low-order fractional portion of the converted integer and store the rightmost nonfractional four or eight digits (single or double integer).

       NOTE

       The high-order digit(s) and sign are not stored.

     - If M4(R/T) is One, do not round. Truncate the low-order fractional portion of the converted integer and store the right-most nonfractional four or eight digits (single or double integer).

       NOTE

       The high digit(s) and sign are not stored; SI(PE) is set if rounding of the converted integer would have incremented the truncated result by One.

   o If the command being executed is a Scientific Swap , the command is terminated; the registers are not altered if the state of M5(SEM) is One.

(Text continues after Figure 8-1)

```
                              *********
                             *  START  *
                              *********
                                  |
                                  |
                   ______________________________
                  / ARE SENDING OPERAND (S) AND \   NO
                 <  DESTINATION OPERAND (D) BOTH >--------------------
                  \_FLOATING POINT (FP) OPERANDS/                     |
                                  | YES                               |
                                  |                                   |
             ---------------------------------------------            |
             |                    |                      |            |
             v                    v                      v            |
     ________________      ________________      ________________     |
    | FOR S AND D ALIKE |  | FOR S = DFP AND  |  | FOR S = QFP AND  |   |
    |   SPECIFICALLY    |  |    D = QFP:      |  |    D = DFP:      |   |
    |  S = D = DFP OR   |  | EXPAND S TO FOUR |  | EXPAND D TO FOUR |   |
    |  S = D = QFP THEN |  | WORDS WITH RIGHT-|  | WORDS WITH RIGHT-|   |
    |EXECUTE INSTRUCTION|  | MOST TWO WORDS   |  | MOST TWO WORDS   |   |
    |_______(EI)________|  |_CLEARED TO ZERO__|  |_CLEARED TO ZERO__|   |
             |                    |                      |            |
             v                    v                      v            |
            ****                 ****                   ****          |
           * EI *               * EI *                 * EI *         |
            ****                 ****                   ****          |
                                                                      |
                                  ------------------------------------
                                  |
                                  |
                   ______________________________
                  /  IS SENDING OPERAND (S) AN  \   NO        o o
                 <  INTEGER (I) AND DESTINATION  >-------->o( A )o
                  \_ OPERAND (D) AN FP OPERAND _/             o o
                                  | YES
                                  |
             ---------------------------------------------
             |                    |                      |
             v                    v                      v
     ________________      ________________      ________________
    |  FOR S = SINGLE   |  |  FOR S = DOUBLE  |  |  FOR S = DOUBLE  |
    | WORD I, CONVERT S |  |WORD I AND D = DFP,|  |WORD I AND D = QFP,|
    | TO NORMALIZED FP  |  |   CONVERT S TO   |  |   CONVERT S TO   |
    |_OF LENGTH = TO D__|  |__NORMALIZED QFP__|  |__NORMALIZED QFP__|
             |                    |                      |
             v                    v                      v
            ****                 ****                   ****
           * EI *               * EI *                 * EI *
            ****                 ****                   ****

                           FOLLOWING INSTRUCTION
                           EXECUTION THE RESULT
                           IS NORMALIZED, ROUNDED
                           AND TRUNCATED TO DFP
```

Figure 8-1 Instruction Pre-Execution (Sheet 1 of 2)

```
                                        o o
                                       o( A )o
                                        o o
                                         |
                                         v
                              __________________________
                             ( SENDING OPERAND (S) IS A )
                             ( FP OPERAND AND DESTINA-  )
                             (  TION OPERAND (D) IS AN  )
                             (_______INTEGER (I)________)
                                         |
                                         |
                                         v
                             ____________________________
                             | NORMALIZE S AND TEST THE |
                             |_______EXPONENT OF S______|
                                         |
                                         |
                                         v
                                _____________________
                         YES  /         IS S          \
         -------------------< LARGER THAN CAPACITY >
         |                    \_______OF D ?_________/
         |                               |  NO
         |                               |
         v                               v
 _______________           ________________________________
 |  CONVERT TO  |          | CONVERT S TO INTEGER (2s     |
 | INTEGER (SEE |          | COMPLEMENT).  SET SI (PE)    |
 |SUBSECTION 8.1.3)|       |_IF FRACTION IS TRUNCATED_____|
         |                               |
         |                               |
         v                               v
        ****                            ****
       * EI *                          * EI *
   IF   ****                       IF   ****
 M5(SEM) |                       M5(PEM) |
   = 1   v                         = 1   v
       ******                          ******
      * TRAP *                        * TRAP *
       ******                          ******
```

Figure 8-1 Instruction Pre-Execution (Sheet 2 of 2)

(Text continued from sheet 8-4)

5. If the integer fits in the destination register and there are nonzero fractional digits, then:

   o If M4(R/T) is a Zero, truncate the fractional portion of the converted integer, store the result, set SI(PE), and if M5 (PEM) is on, post a trap.

   o If M4(R/T) is One, round the converted integer, store the result, and post no trap.

6. If the source SA register operand sign is negative, set the integer to the two's complement of the converted value.

8.1.4 Mixed Precision Operands

Scientific instructions operate on single precision and double precision operands (respectively, double word and quadruple word fields) with formats as described in subsection 3.1.3. Operands entering into scientific operation emanate from main memory, registers (following conversion to floating point quantities), and SA registers. Prior to command execution involving two operands, the CSS tests for unequal length floating point values. If the operand lengths are not equal, the destination scientific accumulator's length (refer to Mode Register, subsection 3.3.4.8) dominates. The adjustment proceeds as follows:

1. If the sending operand length and the receiving operand length are both two words and both are lengthened to four words (two lower-order words are set to Zero), normalize the four word result and round/truncate to two words following command execution (refer to subsection 8.1.5). If, however, both operands are four words, proceed to command execution.

2. If the sending operand length is two words and the receiving operand length is four words, the sending operand is left justified with the low-order (right-most) two words set to Zero prior to command execution. The contents of the sending operand location is not altered.

3. If the sending operand length is four words and the receiving operand length is two words, the receiving operand is lengthened to four words (two low-order words are set to Zero). Following command execution, normalize the four-word result and round/truncate to two words (refer to subsection 8.1.5).

8.1.5 Round/Truncate Mode

Scientific operations requiring right scaling may produce results in which nonzero data has been shifted off the end of the scientific accumulator into two guard digit locations. These guard digits will be truncated from the result after normalization if bit 0 of mode register 4 is Zero (refer to subsection 3.3.4.8). If bit 0 is One, the result after normalization is rounded using the most significant guard digit. If rounding overflows the mantissa, then scale right one digit, shifting in a hex One digit.

8.1.6 Notes On Scientific Operations

1. No scientific operation will produce an unnormalized value in a Scientific Accumulator (SA). The only way that an unnormalized value can occur in an SA is for software to modify the SIP context in the Interrupt Save Area (ISA) prior to a level change that loads the Scientific Accumalators from the ISA. This type of modification is not supported. Any scientific operation that uses an unnormalized value from an SA thus modified will produce unspecified results.

2. No scientific operation will result in both Scientific Indicators SI(g) (greater than) and SI(l) (less than) being set. The only way that both SI(g) and SI(l) can be set at the same time is for software to modify the SIP context in the ISA prior to a level change that loads the Scientific Indicators. This type of modification is not supported. Any scientific branch on indicators SI(g) and/or SI(l) when both are set will produce unspecified results.

3. Whenever the length of a Scientific Accumulator is altered by changing mode register M4, then the Scientific Accumulator must be reloaded by software prior to using it in any other scientific operation. Otherwise, the results of the scientific operation is unspecified.

8.1.7 Abbreviations Used for Indicator Conditions and Traps

Traps (refer to subsection 3.6.2):

PRE[1]	= Scientific Program Error (TV20)
DZ	= Scientific Divide by Zero (TV07)
EOF	= Scientific Exponent Overflow (TV08)
EUF[2]	= Scientific Exponent Underflow (TV19)
SE[2]	= Scientific Significance Error (TV21)
PE[2]	= Scientific Precision Error (TV22)

Where:

1 Note that the usage of IMO and AS3 entries in AS Map 2 by any scientific instruction will result in a Program Error Trap TV16 instead of a TV20.

2 This trap is controlled by a mask bit in the M5 register (refer to subsection 3.3.4.9).

Indicators (refer to subsection 3.3.4.4):

EUF	= Scientific Exponent Underflow
SE	= Scientific Significance Error
PE	= Scientific Precision Error
G	= Scientific Greater Than
L	= Scientific Less Than

8.2 SCIENTIFIC DOUBLE WORD OPERAND INSTRUCTIONS

The Scientific double operand instructions have the following format:

```
   0   1  2     3 4                8 9                     15
  _____________________________________________________________
 |    |    |       |                |                         |
 | 1  |OP  |  X X  |       O P      |           A S           |
 |____|____|_______|________________|_________________________|
 |                                                            |
 -                  Additional words as needed                -
 |____________________________________________________________|
```

or

```
   0   1  2     3 4                8 9                     15
  _____________________________________________________________
 |    |    |       |                |                         |
 | 1  |OP  |  X X  |       O P      |           A S           |
 |____|____|_______|________________|_________________________|
 |                 |          |     |                         |
 |      OFFSET     |   R F U  |MAP #|         AS2, 3          |
 |_________________|__________|_____|_________________________|
 | 0              3 4        6 7   8 9                      15|
 |                                                            |
 -                  Additional words as needed                -
 |____________________________________________________________|
```

where: OP = Opcode

XX = Selects one of the three scientific accumulators:
- 01 = SA1
- 10 = SA2
- 11 = SA3

AS = Address Syllable field.

Depending upon whether the AS specifies the RAS, IMO, or MAS form, the double operand instructions are defined to have the following format respectively:

RR - Register/Register
RI - Register/Immediate Memory
RM - Register/Memory.

These instructions are summarized in Table 8-1. Their numerical representation is given in Table 8-2.

Table 8-1 Scientific Instruction Summary - Double Operand Instructions

				INDICATORS					TRAPS						
REFERENCE	MNEMONIC	DESCRIPTION	OPERATION	EUF	PE	SE	G	L	PRE	DZ	EOF	EUF	SE	PE	COMMENTS
8.2.1	SLD	Scientific Load	[SA#] <-- [EA]	X							X	X			
8.2.2	SCM	Scientific Compare	SI(G), SI(L) <-- [SA#] :: [EA]				X	X							CZ = FZ
8.2.3	SAD	Scientific Add	[SA#] <-- [SA#] + [EA]	X							X	X			
8.2.4	SSB	Scientific Subtract	[SA#] <-- [SA#] - [EA]	X							X	X			
8.2.5	SML	Scientific Multiply	[SA#] <-- [SA#] * [EA]	X							X	X			
8.2.6	SDV	Scientific Divide	[SA#] <-- [SA#] / [EA]	X						X	X	X			
8.2.7	SST	Scientific Store	[EA#] <-- [SA#]		X	X			X		X		X	X	
8.2.8	SSW	Scientific Swap	[SA#] <---> [EA]	X	X	X			X		X	X	X	X	

Table 8-2 Numerical Representation of
Double Operand Scientific Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
8.2.1	8+r	8	8+m	n	SLD	Determined
8.2.2	C+r	8	8+m	n	SCM	by memory
8.2.3	8+r	9	8+m	n	SAD	control bit
8.2.4	C+r	9	8+m	n	SSB	correspon-
8.2.5	8+r	C	0+m	n	SML	ding to
8.2.6	C+r	C	0+m	n	SDV	the selec-
8.2.7	8+r	D	0+m	n	SST	ted SA, in
8.2.8	C+r	D	0+m	n	SSW	M4

where:	r	= Register number contained in bits 2 and 3
	m,n	= Coordinates of AS Map (subsection 3.11)

8.2.1 Scientific Load Instruction (SLD)

Format:

RR, RI, RM

Description:

The operand indentified by the address syllable is loaded into the scientific accumulator specified in bits 2 and 3.

Operations:

```
[SA] <-- [EA]
```

Indicator Conditions:

```
If (e) < -64, then EUF <-- 1
              else EUF <-- 0
                   SE  <-- Unchanged
                   PE  <-- Unchanged
                   G   <-- Unchanged
                   L   <-- Unchanged
```

Trap Conditions:

EOF, EUF.

Special Conditions:

EUF Trap (TV19) is posted only if EUF is set and [M5(0)] = 1.

8.2.2 Scientific Compare Instruction (SCM)

Format:

RR, RI, RM

Description:

The contents of the scientific accumulator, specified in bits 2 and 3, (first operand) is compared to the operand identified by the address syllable (second operand).

NOTE

A clean Zero operand is equal to a fuzzy Zero operand.

Operation:

```
SI(G), SI(L) <-- [SA] :: [EA]
```

Indicator Conditions:

```
                         EUF <-- Unchanged
                          SE <-- Unchanged
                          PE <-- Unchanged
    If [SA] > [EA] then G <-- 1
                   else G <-- 0
    If [SA] < [EA] then L <-- 1
                   else L <-- 0
```

Trap Conditions:

None.

Special Conditions:

None.

8.2.3 Scientific Add Instruction (SAD)

Format:

RR, RI, RM

Decription:

The operand identified by the address syllable is added to the contents of the scientific accumulator specified in bits 2 and 3.

Operation:

```
[SA] <-- [SA] + [EA]
```

Indicator Conditions:

```
    If (e) < -64, then EUF <-- 1
                  else EUF <-- 0
                        SE <-- Unchanged
                        PE <-- Unchanged
                         G <-- Unchanged
                         L <-- Unchanged
```

Trap Conditions:

EOF, EUF.

Special Conditions:

EUF Trap (TV19) is posted only if EUF is set and [M5(0)] = 1.

8.2.4 Scientific Subtract Instruction (SSB)

Format:

RR, RI, RM

Description:

The operand identified by the address syllable is subtracted from the contents of the scientific accumulator specified in bits 2 and 3.

Operation:

[SA] <-- [SA] - [EA]

Indicator Conditions:

```
If(e) < -64, then EUF <-- 1
             else EUF <-- 0
                   SE <-- Unchanged
                   PE <-- Unchanged
                    G <-- Unchanged
                    L <-- Unchanged
```

Trap Conditions:

EOF, EUF.

Special Conditions:

EUF Trap (TV19) is posted only if EUF is set and [M5(0)] = 1.

8.2.5 Scientific Multiply Instruction (SML)

Format:

RR, RI, RM

Description:

The operand indentified by the address syllable is multiplied by the contents of the scientific accumulator (specified in bits 2 and 3) and the product is put in SA (1, 2 or 3).

Operation:

[SA] <-- [SA] * [EA]

Indicator conditions:

```
    If(e) < -64, then EUF <-- 1
                 else EUF <-- 0
                       SE <-- Unchanged
                       PE <-- Unchanged
                        G <-- Unchanged
                        L <-- Unchanged
```

Trap Conditions:

EOF, EUF.

Special Conditions:

EUF Trap (TV19) is posted only if EUF is set and [M5(0)] = 1.

8.2.6 Scientific Divide Instruction (SDV)

Format:

RR, RI, RM

Description:

The content of the scientific accumulator specified in bits 2 and 3 is divided by the operand identified by the address syllable. The operands are normalized prior to the arithmetic operations, however, if the divisor is Zero, an unconditional Trap (TV07) is posted, the operands are left unmodified, and the divide operation is not executed.

Operation:

[SA] <-- [SA] / [EA]

Indicator Conditions:

```
    If (e) < -64, then EUF <-- 1
                  else EUF <-- 0
                        SE <-- Unchanged
                        PE <-- Unchanged
                         G <-- Unchanged
                         L <-- Unchanged
```

Trap Conditions:

EOF, EUF, DZ.

Special Conditions:

EUF Trap (TV19) is posted only if EUF is set and [M5(0)] = 1.

8.2.7 Scientific Store Instruction (SST)

Format:

RR, RM

Description:

The contents of the location specified by the address syllable is replaced by the contents of the scientific accumulator specified in bits 2 and 3. Conversion of the contents in SA to an integer prior to transfer to a register can cause one or more indicator changes (see below).

Operation:

[EA] <-- [SA]

Indicator Conditions:

EUF <-- Unchanged.

If scale left operation during conversion of a floating number to an integer number cause non-Zero bits to be lost (refer to subsection 8.1.3), then:

```
     SE <-- 1
else SE <-- 0.
```

If scale right operation during conversion of a floating number to an integer number cause non-Zero bits to be lost (refer to subsection 8.1.3),

```
then PE <-- 1
else PE <-- 0
     G  <-- Unchanged
     L  <-- Unchanged
```

Trap Conditions:

PRE, EOF, SE, PE.

Special Conditions:

1. SE Trap (TV21) is posted only if SE is set and [M5(2)] = 1.

2. PE Trap (TV22) is posted only if PE is set and [M5(3)] = 1.

3. Use of the IMO AS will cause TV20.

8.2.8 Scientific Swap Instruction (SSW)

Format:

RR, RI, RM

Description:

The content of a location identified by the address syllable is swapped with the content of the scientific accumulator specified in bits 2 and 3.

Operation:

```
[SA] <--> [EA]
```

Indicator Conditions:

```
If (e) < -64, then EUF <-- 1
              else EUF <-- 0.

If scale left operation during conversion of a floating number to an intger
number cause non-Zero bits to be lost (refer to subsection 8.1.3),
              then SE  <-- 1
              else SE  <-- 0.

If scale right operations during conversion of a floating number cause
non-Zero bits to be lost (refer to subsection 8.1.3),
              then PE  <-- 1
              else PE  <-- 0
                   G   <-- Unchanged
                   L   <-- Unchanged.
```

Trap Conditions:

PRE, EOF, EUF, SE, PE.

Special Conditions:

1. EUF Trap (TV19) is posted only if EUF is set and [M5(0)] = 1.

2. SE Trap (TV21) is posted only if SE is set and [M5(2)] = 1.

3. PE Trap (TV22) is posted only if PE is set and [M5(3)] = 1.

4. Use of the IMO AS will cause a TV20.

8.3 SCIENTIFIC SINGLE OPERAND INSTRUCTIONS

Scientific single operand instructions have the following format:

```
 0    1   3 4        8 9        15
 _________________________________
|    |     |          |           |
| 1  | 100 |    OP    |    AS     |
|____|_____|__________|___________|
|                                 |
-   Additional words as needed    -
|_________________________________|
```

or

```
 0    1   3 4        8 9        15
 _________________________________
|    |     |          |           |
| 1  | 100 |    OP    |    ASN    |
|____|_____|__________|___________|
|          |     |    |           |
| OFFSET   | RFU |MAP#|   AS2,3   |
|__________|_____|____|___________|
|0        3 4   6 7  8 9        15|
|                                 |
-   Additional words as needed    -
|_________________________________|
```

where bit 1 = 0 for all single operand instructions except
= 1 for scientific single operand instructions;

OP = Opcode field;

AS = Address Syllable field.

Depending upon whether the AS specifies an RAS, MAS, or IMO form, these instructions are defined to have the following format:

R = Register only
M = Memory only
I = Immediate only.

These instructions are summarized in Table 8-3. Their numerical representation is given in Table 8-4.

Table 8-3 Scientific Single Operand Instruction Summary

				INDICATORS					TRAPS						
REFERENCE	MNEMONIC	DESCRIPTION	OPERATION	EUF	PE	SE	G	L	PRE	DZ	EOF	EUF	SE	PE	COMMENTS
8.3.1	SCZD	Scientific Compare to Zero - 2 Words	SI(G), SI(L) <-- [EA] :: Zero				X	X	X						CZ = FZ
8.3.2	SNGD	Scientific Negate - 2 Words	[EA(s) <-- $\overline{\text{[EA(s)}}$						X						
8.3.3	SCZQ	Scientific Compare to Zero - 4 Words	SI(G), SI(L) <-- [EA] :: Zero				X	X	X						CZ = FZ
8.3.4	SNGQ	Scientific Negate - 4 Words	[EA(s) <-- $\overline{\text{[EA(s)}}$						X						

Table 8-4 Numerical representation of Single Operand Scientific Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
8.3.1	C	8	8+m	n	SCZD	DWord
8.3.2	C	9	8+m	n	SNGD	DWord
8.3.3	C	C	0+m	n	SCZQ	QWord
8.3.4	C	D	0+m	n	SNGQ	QWord

where: m,n = Coordinates of AS Map (subsection 3.11)

8.3.1 Scientific Compare to Zero - Two Words (SCZD)

Format:

I, M

Description:

The operand identified by the address syllable (first operand) is compared with a clean Zero (second operand).

A clean Zero operand is reported equal to a fuzzy Zero operand. An SCZD instruction using an RAS operand causes a scientific program error Trap (TV20) to be posted, and no change to the scientific indicator register.

Operation:

SI(G), SI(L) <-- [EA] :: Zero

Indicator conditions:

```
                        EUF <-- Unchanged
                         SE <-- Unchanged
                         PE <-- Unchanged
If [EA] > Zero,   then G  <-- 1
                  else G  <-- 0
If [EA] < Zero,   then L  <-- 1
                  else L  <-- 0
```

Trap Conditions:

PRE

Special Conditions:

PRE Trap (TV20) occurs if an RAS operand is selected (refer to subsection 3.11.1).

8.3.2 Scientific Negate - Two Words (SNGD)

Format:

R, I, M (R only for SA1, 2, 3)

Description:

The operand identified by the address syllable is negated; that is , the sign bit of the floating point number (bit 7) is altered (if Zero, to One; if One, to Zero) except when the value of the floating point number is Zero. In this case the resultant sign bit is Zero. If the format is R, the length of the operand is determined by the mode register. Register addressing is restricted to the scientific accumulators. An SNGD instruction using any other RAS operand, causes a scientific program error Trap (TV20) to be posted and no change to the operand.

Operation:

$[EA(s)] \leftarrow \overline{[EA\ (s)]}$

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Trap Conditions:

PRE

Special Conditions:

1. PRE Trap (TV20) occurs if RAS (Kn, R4, R5, R6, R6/R7) AS is used.

2. Use of IMO AS may cause change of procedure.

8.3.3 Scientific Compare to Zero - Four Words (SCZQ)

Format:

I, M

Description:

The operand indentified by the address syllable (first operand) is compared with a clean Zero (second operand).

A clean Zero operand is reported equal to a fuzzy Zero operand. An SCZQ instruction using an RAS operand causes a program error Trap (TV20) to be posted and no change to the scientific indicator register.

Operation:

SI(G), SI(L) <-- [EA] :: Zero

Indicator Conditions:

```
                         EUF <-- Unchanged
                          SE <-- Unchanged
                          PE <-- Unchanged
If [EA] > Zero, then G <-- 1
                else G <-- 0
If [EA] < Zero, then L <-- 1
                else L <-- 0
```

Trap Conditions:

PRE

Special Conditions:

PRE Trap (TV20) occurs if an RAS operand is selected (refer to subsection 3.11.1).

8.3.4 Scientific Negate - Four Words (SNGQ)

Format:

R, I, M (R Only for SA1, 2, 3)

Description:

The operand identified by the address syllable is negated; that is , the sign bit of the floating point number (bit 7) is altered (if Zero, to One; if One, to Zero) except when the value of the floating point number is Zero. In this case, the resultant sign bit is Zero. If the format is R, the length of the operand is determined by the mode register. Register addressing is restricted to the scientific accumulators. An SNGQ instruction using any other RAS operand causes a scientific program error Trap (TV20) to be posted and no change to the operand.

Operation:

$[EA(s)] \leftarrow \overline{[EA(s)]}$

Indicator Conditions:

EUF <-- Unchanged
SE <-- Unchanged
PE <-- Unchanged
G <-- Unchanged
L <-- Unchanged

Trap Conditions:

PRE

Special Conditions:

1. PRE Trap (TV20) occurs if RAS (Kn, R4, R5, R6, R6/R7) AS is used.

2. Use of IMO AS may cause change of procedure.

8.4 BRANCH ON SCIENTIFIC ACCUMULATOR INSTRUCTIONS

Branch on Scientific Accumulator (BR) instructions have the format shown in Figure 8-2.

These instructions enable branching on selected Scientific Accumulators; e.g., equal to zero, less than zero, etc. These instructions are defined in summary form in Table 8-5. Their numerical representation is given in Table 8-6.

```
 0    1    2 3   4      8 9         15
 ____________________________________
|    |    |     |       |            |
| 0  | 0  | X X |  OP   |     d      |
|____|____|_____|_______|____________|
```

where: XX - Selects one of the three scientific accumulators:
01 = SA1
10 = SA2
11 = SA3

OP - Opcode field
d - Displacement (see Figure 5-1 for an explanation of the displacement field).

Figure 8-2 Branch on Scientific Accumulator Instruction Format

Table 8-5 Branch on Scientific Accumulators

REFERENCE	MNE-MONIC	DESCRIPTION	OPERATIONS	INDICATORS AFFECTED	COMMENTS
8.4.1	SBLZ	Branch if [SA] Less than Zero	If [SA(s)]=1 and if [SA(f)]≠0, then [P] <-- EA		In all branch operations, if the branch condition is true (i.e., branch will) be executed), if [M1(J)]=1, then Trap to TV02.
8.4.2	SBGEZ	Branch if [SA] Greater than or Equal to 0	If [SA(s)=0 or [SA(f)] ≠ 0, then [P] <-- EA		
8.4.3	SBEZ	Branch if [SA] Equal to 0	If [SA(f)]=0, then [P] <-- EA		
8.4.4	SBNEZ	Branch if [SA] Not Equal to 0	If [SA(f)]≠0, then [P] <-- EA		
8.4.5	SBGZ	Branch if [SA] Greater than 0	If [SA(f)]≠0 and if [SA(s)]=0, then [P] <-- EA		
8.4.6	SBLEZ	Branch if [SA] Less than or Equal to 0	If [SA(f)]=0, or if [SA(s)]=1, then [P] <-- EA		

where f = Fraction (mantissa) excluding sign

s = Sign

SA = Scientific Accumulator

Table 8-6 Numerical Representation of Branch on Scientific Accumulator Instructions

SUBSECTION	H1	H2	H3 H4	MNEMONIC	ATOM SIZE
8.4.1	0+r	4	0+d	SBLZ	
8.4.2	0+r	4	8+d	SBGEZ	NOT
8.4.3	0+r	5	0+d	SBEZ	APPLI-
8.4.4	0+r	5	8+d	SBNEZ	CABLE
8.4.5	0+r	6	0+d	SBGZ	
8.4.6	0+r	6	8+d	SBLEZ	

where r = Accumulator number contained in bits 2 & 3 of the instruction

d = 7-bit displacement.

8.4.1 Branch if [SA] Less Than Zero (SBLZ)

Format:

BR

Description:

Branch to the effective address if the content of the SA is negative.

Operation:

If [SA(s)] = 1 and if [SA(f)] $\neq$ 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trap TV02 (trace trap) occurs after the SBLZ instruction is executed.

8.4.2 Branch if [SA] Greater Than or Equal to Zero (SBGEZ)

Format:

BR

Description:

Branch to the effective address if the content of the SA is positive or Zero.

Operation:

If [SA(f)] = 0 or [SA(s)] = 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBGEZ instruction is executed.

8.4.3 Branch if [SA] Equal to Zero (SBEZ)

Format:

BR

Description:

Branch to the effective address if the content of the SA is Zero.

Operation:

If [SA(f)] = 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBNEZ instruction is executed.

8.4.4 Branch if [SA] Not Equal to Zero (SBNEZ)

Format:

BR

Description:

Branch to the effective address if the content of the SA is not Zero.

Operation:

If [SA(f)] $\neq$ 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged.
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBNEZ instruction is executed.

8.4.5 Branch if [SA] Greater Than Zero (SBGZ)

Format:

BR

Description:

Branch to the effective address if the content of the SA is greater than Zero.

Operation:

If [SA(f)] ≠ 0 and if [SA(s)] = 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch conditions is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBGZ instruction is executed.

8.4.6 Branch if [SA] Less Than or Equal to Zero (SBLEZ)

Format:

BR

Description:

Branch to the effective address if the content of the SA is less than or equal to Zero.

Operation:

If [SA(f)] = 0 or if [SA(s)] = 1 then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch conditions is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBLEZ instruction is executed.

8.5 BRANCH ON SCIENTIFIC INDICATOR INSTRUCTIONS

Branch on Scientific Indicator (BI) instructions have the format shown in Figure 8-3.

These instructions enable branching on various indicators conditions; e.g., equal, less than, greater than, etc. These instructions are summarized in Table 8-7. Their numerical representation is given in Table 8-8.

```
   0   1  2          8 9        15
  ________________________________
 |   |   |            |           |
 | 0 | 1 |     OP     |     d     |
 |___|___|____________|___________|
```

where: OP = Opcode field
d = Displacement (see Figure 5-2 for an explanation of the displacement field).

Figure 8-3 Branch on Scientific Indicator Instruction Format

8.5.1 Branch on Less Than (SBL)

Format:

BI

Description:

Branch to the effective address if the Scientific Less Than indicator is set.

Operation:

If [SI(L)] = 1, then [P] <-- EA

Indicator Conditions:

EUF <-- Unchanged
SE <-- Unchanged
PE <-- Unchanged
G <-- Unchanged
L <-- Unchanged

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then a trace Trap (TV02) occurs after the SBL instruction is executed.

Table 8-7 Branch on Scientific Indicators

REFERENCE	MNE-MONIC	DESCRIPTION	OPERATIONS	INDICATORS AFFECTED	COMMENTS
8.5.1	SBL	Branch on Less Than	If [SI(L)]=1 then [P] <-- EA		In all branch operations, if the branch condition is true (i.e., branch is executed) and if [M1(J)]=1, then trap to TV02.
8.5.2	SBPE	Branch on Precision Error	If [SI(PE)]=1 then [P] <-- EA		
8.5.3	SBSE	Branch on Significance Error	If [SI(SE)]=1, then [P] <-- EA		
8.5.4	SBGE	Branch on Greater Than or Equal to	If [SI(L)=0, then [P] <-- EA		
8.5.5	SBNPE	Branch on No Precision Error	If [SI(PE)]=0, then [P] <-- EA		
8.5.6	SBNSE	Branch on No Significance Error	If [SI(SE)]=0, then [P] <-- EA		
8.5.7	SBE	Branch on Equal	If [SI(L)/\SI(G)]=0 then, [P] <-- EA		
8.5.8	SBNE	Branch on Not Equal	If [SI(L)\/SI(G)]=1 then, [P] <-- EA		
8.5.9	SBG	Branch on Greater Than	If [SI(G)]=1, then [P] <-- EA		
8.5.10	SBEU	Branch on Exponent Underflow	If [SI(EUF)]=1,then then [P] <-- EA		
8.5.11	SBLE	Branch on Less Than or Equal	If [SI(G)]=0, then [P] <-- EA		
8.5.12	SBNEU	Branch on No Exponent Underflow	If [SI(EUF)]=0, then [P] <-- EA		

Table 8-8 Numerical Representation of Branch on Scientific Indicator Instructions

SUBSECTION	H1	H2	H3 H4	MNEMONIC	ATOM SIZE
8.5.1	4	4	0+d	SBL	
8.5.4	4	4	8+d	SBGE	
8.5.7	4	5	0+d	SBE	NOT
8.5.8	4	5	8+d	SBNE	
8.5.9	4	6	0+d	SBG	APPLI-
8.5.11	4	6	8+d	SBLE	
8.5.2	5	4	0+d	SBPE	CABLE
8.5.5	5	4	8+d	SBNPE	
8.5.3	6	4	0+d	SBSE	
8.5.6	6	4	8+d	SBNSE	
8.5.10	7	4	0+d	SBU	
8.5.12	7	4	0+d	SBNEU	

where d = 7-bit displacement

8.5.2 Branch on Precision Error (SBPE)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicator, Precision Error, is set.

Operation:

If [SI(PE)] = 1, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J)= 1, then trace Trap (TV02) occurs after the SBPE instruction is executed.

8.5.3 Branch on Significance Error (SBSE)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicator, Significance Error, is set.

Operation:

If [SI(SE)] = 1, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBSE instruction is executed.

8.5.4 Branch on Greater Than or Equal (SBGE)

Format:

BI

Description:

Branch to the effective address if the Scientific Less Than indicator is not set.

Operation:

If [SI(L)] = 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBGE instruction is executed.

8.5.5 Branch on No Precision Error (SBNPE)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicator, Precision Error, is not set.

Operation:

If [SI(PE)] = 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBNPE instruction is executed.

8.5.6 Branch on No Significance Error (SBNSE)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicator, Significance Error, is not set.

Operation:

If [SI(SE)] = 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBNSE instruction is executed.

8.5.7 Branch on Equal (SBE)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicators Greater Than and Less Than are not set.

Operation:

If [SI(L) /\ SI(G)] = 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBE instruction is executed.

8.5.8 Branch on Not Equal (SBNE)

Format:

BI

Description:

Branch to the effective address if either of the Scientific Indicators, Greater Than or Less Than are set.

Operation:

If [SI(L) \/ SI(G)] = 1, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If a branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBNE instruction is executed.

8.5.9 Branch on Greater Than (SBG)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicator, Greater Than is set.

Operation:

If [SI(G)] = 1, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBG instruction is executed.

8.5.10 Branch on Exponent Underflow (SBEU)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicator, Exponent Underflow, is set.

Operation:

If [SI(EUF)] = 1, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBEU instruction is executed.

8.5.11 Branch on Less Than or Equal (SBLE)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicator, Less Than is not set.

Operation:

If [SI(G)] = 0, then [P] <-- EA

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBLE instruction is executed.

8.5.12 Branch If No Exponent Underflow (SBNEU)

Format:

BI

Description:

Branch to the effective address if the Scientific Indicator, Precision Error, is not set.

Operation:

If [SI(EUF)] = 0, then [P] <-- EA

Indicator Conditions:

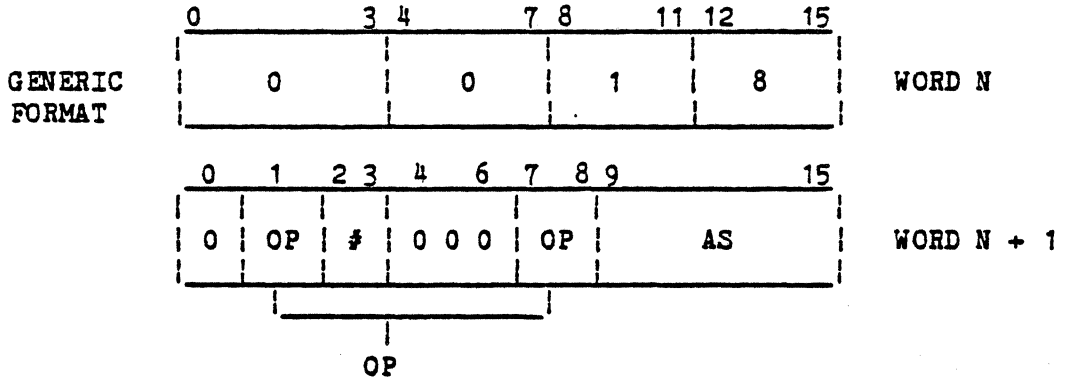
```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Special Conditions:

If the branch condition is true (i.e., a branch is executed) and M1(J) = 1, then trace Trap (TV02) occurs after the SBNEU instruction is executed.

8.6 Scientific Intrinsics

These instructions are available only on the M6X and M6XE. The format for intrinsic instructions is a generic instruction followed by a single word that specifies AS, SA, and function. The format is:



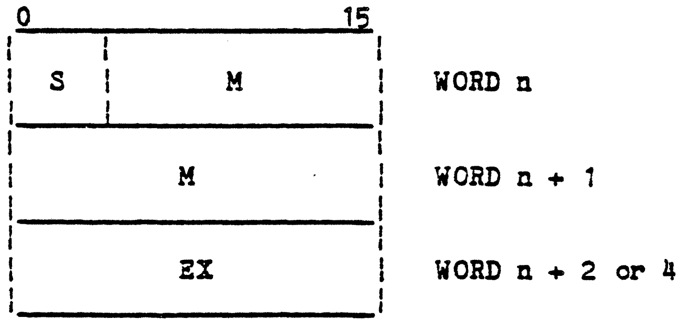
where: # - Selects one of the three scientific accumulators (SA1, SA2, SA3):
01 = SA1
10 = SA2
11 = SA3

OP - Function select:
000 = Sine (SSIN)
001 = Arc Tangent (SART)
010 = Exponentiate (SEXP)
011 = Convert Decimal to Hex (SDTH)
100 = Cosine (SCOS)
101 = Square Root (SSRT)
110 = Log (SLOG)
111 = Convert Hex to Decimal (SHTD)

AS - Address Syllable:

Bn and SA are the only legal AS for intrinsics except for the two convert instructions which may only use the Bn AS else TV05.

The convert instructions are used to convert between a pseudo decimal floating point number and a floating point hexadecimal number. The source or destination of the pseudo decimal floating point number is defined by the instruction AS which points to a three- or five-word area in memory. The memory format is:



where EX, S and M are defined in subsection 3.1.3.3.

Intrinsic instructions are used to compute a square root, power of e, logarithm, or trigonometric function. Intrinsic instructions are summarized in Table 8-9 and described in subsections 8.6.1 through 8.6.8. Their numerical representation is given in Table 8-10.

The intrinsic instructions provide accuracy equal to or greater than the accuracy provided by Advanced FORTRAN, Release 1 under the MODE 400, Release 200 Operating System (using the appropriate precision SI mode or SI simulator).

The conversion functions try to preserve as much precision as possible. The effective weight of the least significant bit of the converted value is no more than the weight of the least significant bit of the input argument.

Although the precision criteria is maintained during conversion, the instructions SHTD and SDTH are not exact inverses. A value (V1) converted to V2 by SDTH, and V2 converted by SHTD will result in V3 which although equivalent to V1 may not be the same representation.

Table 8-9 Scientific Instruction Summary - Intrinsic Instructions

				INDICATORS					TRAPS						
REFERENCE	MNEMONIC	DESCRIPTION	OPERATION	EUF	PE	SE	G	L	PRE	DZ	EOF	EUF	SE	PE	COMMENTS
8.6.1	SHTD	Convert Hex to Decimal	[EA] <-- [SA#]												Note 1
8.6.2	SDTH	Convert Decimal to Hex	[SA#] <-- [EA]	X							X	X			Note 2
8.6.3	SSIN	Trigonometric Sine	[SA#] <-- SINE [EA]												
8.6.4	SCOS	Trigonometric Cosine	[SA#] <-- COSINE [EA]												
8.6.5	SLOG	Natural Logarithm	[SA#] <-- LOG_e [EA]												
8.6.6	SEXP	Exponentiation	[SA#] <-- $e^{[EA]}$												
8.6.7	SART	Trigonometric Arc Tangent	[EA#] <-- TAN^{-1}[EA]												
8.6.8	SSRT	Square Root	[SA#] <--- $[EA]^{1/2}$												

NOTES

1. Floating hex to floating decimal.
2. Floating decimal to floating hex.
3. REG = SA1, 2 or 3.

Table 8-10 Numerical Representation of Intrinsic Scientific Instructions

SUBSECTION	H1	H2	H3	H4	MNEMONIC	ATOM SIZE
	0	0	1	8	Intrinsic	
8.6.3	r	0	m	n	SSIN #	Determined
8.6.7	r	0	8+m	n	SART #	by memory
8.6.6	r	1	m	n	SEXP #	contol bit
8.6.2	r	1	8+m	n	SDTH #	correspon-
8.6.4	4+r	0	m	n	SCOS #	ding to
8.6.8	4+r	0	8+m	n	SSRT #	the selec-
8.6.5	4+r	1	m	n	SLOG #	ted SA, in
8.6.1	4+r	1	8+m	n	SHTD #	M4

where: r = Scientific Accumulator bits 2 and 3;
m,n = Coordinates of AS Map (see subsection 3.11).
= The numerical representation given is for the second word of the instruction which determines the instruction type.

8.6.1 CONVERT HEXADECIMAL TO DECIMAL (SHTD)

Format:

Ge

Description:

The hexadecimal number in the selected SA is converted to two binary numbers, EX, M and S (representing the decimal exponent, mantissa and sign), which are stored in the memory area designated by the AS.

Operation:

As above.

Indicator Conditions:

EUF <-- Unchanged
SE <-- Unchanged
PE <-- Unchanged
G <-- Unchanged
L <-- Unchanged

Trap Conditions:

None

Special Conditions:

Trap (TV05) occurs if the AS used was not Bn from Map 1.

8.6.2 CONVERT DECIMAL TO HEXADECIMAL (SDTH)

Format:

Ge

Description:

The two binary numbers (representing the decimal sign, mantissa, and exponent) in the memory field designated by the AS are converted to floating point hexadecimal representation and stored in the selected SA. The AS field of the mode register (M4) determines the mantissa size assumed during the conversion. If the magnitude of the converted value is too small or too large to fit the standard hexadecimal floating point format, an EOF trap or exponent underflow (EUF) occurs, respectively.

Operation:

As above.

Indicator Conditions:

```
If EX < -77, then EUF <-- 1
            else EUF <-- 0
                  SE <-- Unchanged
                  PE <-- Unchanged
                   G <-- Unchanged
                   L <-- Unchanged
```

Trap Conditions:

EOF, EUF

Special Conditions:

Trap (TV05) occurs if the AS used was not Bn from Map 1.

8.6.3 SINE (SSIN)

Format:

Ge

Description:

The trigonometric sine of the operand identified by the address syllable is computed and the result is stored in the specified scientific accumulator. The operand is in radians. The precision of the operation is determined by the operand and accumulator lengths specified by M4. Refer to subsections 8.1.2 through 8.1.5.

Operation:

[SA] <-- SINE [EA]

Indicator Conditions:

EUF <-- Unchanged
SE <-- Unchanged
PE <-- Unchanged
G <-- Unchanged
L <-- Unchanged

Trap Conditions:

None

Special Conditions:

1. TV33 occurs if the absolute value of the input operand is greater than:

   o Single Precision - 8E6487EC hexadecimal floating point (.10541436E9 decimal floating point)

   o Double Precision - 9C1921FB54EB96BF hexadecimal floating point.

2. Trap TV05 occurs if the AS used was not Bn or REG (SA1, 2, 3) from Map 1.

8.6.4 COSINE (SCOS)

Format:

Ge

Description:

The trigonometric cosine of the operand identified by the address syllable is computed and the result is stored in the specified scientific accumulator. The operand is in radians. The precision of the operation is determined by the operand and accumulator lengths specified in M4. Refer to subsections 8.1.2 through 8.1.5.

Operation:

[SA] <-- CONSINE [EA]

Indicator Conditions:

EUF <-- Unchanged
SE <-- Unchanged
PE <-- Unchanged
G <-- Unchanged
L <-- Unchanged

Trap Conditions:

None

Special Conditions:

Same as for SSIN.

8.6.5 NATURAL LOGARITHM (SLOG)

Format:

Ge

Description:

The natural logarithm of the operand identified by the address syllable is computed and the result is stored in the specified scientific accumulator. The precision of the operation is determined by the operand and accumulator lengths specified in M4. Refer to subsections 8.1.2 through 8.1.5.

Operation:

[SA] <-- LOG_e [EA]

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Trap Conditions:

None

Special Conditions:

1. TV33 occurs if the operand is negative or zero.

2. Trap TV05 occurs if the AS used was not Bn or REG (SA1, 2, 3) from Map 1.

8.6.6 EXPONENTIATE (SEXP)

Format:

Ge

Description:

The number e (natural logarithm base) is raised to the power specified in the operand identified by the address syllable. The result is stored in the specified scientific accumulator. The precision of the operation is determined by the operand and accumulator lengths specified in M4. Refer to subsections 8.1.2 through 8.1.5.

Operation:

$[SA] \leftarrow e^{[EA]}$

Indicator Conditions:

EUF <-- Unchanged
SE <-- Unchanged
PE <-- Unchanged
G <-- Unchanged
L <-- Unchanged

Trap Conditions:

None

Special Conditions:

1. TV33 occurs if the value of the operand is greater than:

   o Single Precision - 84AEAC4E hexadecimal floating point

   o Double Precision - 84AEAC4EF88B9777 hexadecimal floating point.

2. Trap TV05 occurs if the AS used was not Bn or REG (SA1, 2, 3) from Map 1.

8.6.7 ARC TANGENT (SART)

Format:

Ge

Description:

The angle whose tangent is the operand specified by the address syllable is computed. The result, in radians, is stored in the specified scientific accumulator. The precision of the operation is determined by the operand and accumulator lengths specified in M4. Refer to subsections 8.1.2 through 8.1.5.

The result of the SART instruction is within the range of $\pm$ pi/2 radians.

Operation:

$[SA] \leftarrow TAN^{-1} [EA]$

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Trap Conditions:

None

Special Conditions:

Trap TV05 occurs if the AS used was not Bn or REG (SA1, 2, 3) from Map 1.

8.6.8 SQUARE ROOT (SSRT)

Format:

Ge

Description:

The square root of the operand specified by the address syllable is computed and the result is stored in the specified scientific accumulator. The pre- cision of the operation is determined by the operand and accumulator lengths specified in M4. Refer to subsections 8.1.2 through 8.1.5.

Operation:

$[SA] \leftarrow [EA]^{1/2}$

Indicator Conditions:

```
EUF <-- Unchanged
 SE <-- Unchanged
 PE <-- Unchanged
  G <-- Unchanged
  L <-- Unchanged
```

Trap Conditions:

None

Special Conditions:

1. TV33 occurs if the value of the operand is negative.

2. Trap TV05 occurs if the AS used was not Bn or REG (SA1, 2, 3) from Map 1.

This page is intentionally blank.

SECTION 9 NEW INSTRUCTION SET DEFINITION

This section does not apply to this EPS-1.

This page is intentionally blank.

SECTION 10 MULTIPROCESSOR CONSIDERATIONS/INTERFACES

10.1 MULTIPROCESSOR CONSIDERATIONS

DPS6 Stage 3 Systems will offer both the traditional DPS6 monoprocessor configurations (i.e., CR41E-based models) as well as tightly coupled multiprocessor configurations (e.g., Dual 6X models).

Note that a tightly coupled multiprocessor system, as the term is used in this document, is defined as a group of processor resources under the control of a single operating system and sharing a common address space.

The following information is presented to convey some of the finer points to be considered in the use of the L6 in multiprocessor configurations.

10.1.1 Basic Configurations

L6 Architecture is designed to accomodate from one to four central subsystems attached to the system Bus (Megabus or Extended Megabus). Table 10-1 defines Bus Slot requirements for those DPS6-CSSs allowing for multiprocessor support.

10.1.2 Channel Numbering

Each CSS in a multi-CSS configuration is assigned a unique channel number. This channel number is used by memory and configured I/O controllers to communicate (interrupts, status, and Memory Read responses) with the individual CSS. Channel numbers are also used to relocate individual CSS' "Dedicated Memory" areas (see subsection 10.1.3).

Table 10-1 CSS Bus Slot Requirements

GENERIC NAME	CSS FUNCTIONALITY	CSS BUS SLOTS REQ'D	MEMORY SIZE (BYTES)			MEMORY BUS SLOTS PER INCREMENT
			Min.	Max.	Modularity	
M5X	M5X CSS & SCF (Data-Net only)	5 for Dual	512K (N/A)	2M	512K Increments	2 total
M5XE	M5XE CSS & SCF (Data-Net only)	5 for Dual	2M	8M	2M Increments	1 total
M6X	M6X (CSS, SIP, CIP, SCF)	3*	1M (4M)	16M	1M, 4M, 8M, 16M	2 total
M6XE	M6X (CSS, SIP, CIP, SCF, EMMU)	3*	1M (4M)	16M	1M, 4M, 8M, 16M	2 total

*Two additional 10-slot 6X chassis' must be configured in addition to the Megabus chassis.

Channel number assignments are fixed depending on the position of the CSS in the cabinet. CSS0 is assigned channel number 0000 and CSS1 is assigned channel number 0040. Note that in some documentation CSS0 is referred to as Central System No. 1 and CSS1 is referred to as Central System No. 2.

10.1.3 CPU Specialization

Several functions are performed by the CSS to guarantee integrity of itself, as well as that of the other components (processors) attached to the system bus. Specifically, the CPU performs a Self Test (QLT) on both itself and the system main memory, provides a system bus deadman timeout facility to prevent bus hangs, and provides System Bootstrap capability for "Cold Start" of the system. The "Deadman Timer" is approximately a 5-microsececond timer started with the generation of each system bus cycle, regardless of the originator, and causes a "no acknowledge" response (NACK) to be generated by the CSS if, upon expiration of the timer, no external response is received. The NACK response frees the system bus for further operations. Note that other forms of timeout functions have been provided in certain models (i.e., M6X and M6XE CSS 1 ms operations timeouts) and are implementation specific.

In a multiprocessor system, each processor attached cannot assume the responsibility for simultaneously performing external services (i.e. Memory QLT, Bootload, System Bus Timers) without catastrophic results. To alleviate this problem, CSS0 (channel number 0000) is delegated the task of performing these services while all other processors only perform their internal self-testing.. In this sense a master/slave relationship is established where CSS0 is the master and all others are slaves, by virtue of the configurable channel number assignment.

10.1.4 Dedicated Memory

Each L6 CSS has a 256-word dedicated control area in memory that contains IVs, TVs, etc. These areas are kept separate by the CSSs via relocation based upon the CSS channel number, and are specified in Table 10-2.

Table 10-2 Dedicated Memory Areas

CPU Number	Channel Number	Dedicated Memory Region (Hex Address)
0	0000	000000 - 0000FF
1	0040	000100 - 0001FF

CSSs use their (relocated) dedicated memory area whenever addressing any of the entries (except for the memory error count, 00001F, which is not relocated). The reallocation is automatic and performed by the CSS. Direct access to any dedicated memory location by software must take this into account.

10.1.5 Control Panels

The system's control panel functionality is provided by the VCP (Virtual Control Panel) capability of the SCF and a separate full control panel. The SCF is connected to the master CSS. The full control panel is connected to the slave CSS.

The depression of the clear button on the panel is seen system wide. Individual clearing of CPUs is not possible (see 10.1.6)

10.1.6 Initialization

Upon sensing the presence of the "Master Clear" signal, all processors react by clearing specified registers, modes, trivializing MMU images, and executing internal self-test/verification firmware. In completing these functions, the processors determine what condition invoked the clear signal (Power-On, Panel Clear, etc.) and react to the stimulus as a result of what processor number is configured.

The initialize operations performed by a CSS depend on whether the CSS is the master or the slave. In dual 6X and 6XE systems, CSS0 is always configured as the master while CSS1 is always configured as the slave.

The master CSS (CSS0) reacts to an initialize by performing a self test and the following functions as a function of panel state and the initialize situation:

- o If the panel is unlocked, CSS0 halts at Level 0 and waits for manual intervention.
- o If the panel is locked and performing a Start, CSS0 performs an automatic bootload (autoboot) from channel 0400. Execution starts at memory location 000100 with the CSS at Level 0.
- o If the panel is locked and performing a Restart, CSS0 performs an autorestart. Execution starts at memory location 000000 with the CSS at Level 0.

The slave CSS (CSS1) reacts to an initialize by performing a self test and the following functions as a function of panel state:

- o If the panel is unlocked, CSS1 halts at Level 0 and waits for manual intervention.
- o If the panel is locked (typical setting) CSS1 goes to Level 63 and waits for an interrupt from the master CSS before it starts execution.

10.1.7 Interprocessor Communications

There is no explicit hardware communication between CPUs. Each CPU runs on its own, using its own dedicated memory, Level flags, etc.

A CPU can interrupt another CPU if the code in execution builds an I/O instruction that appears as an interrupt and directs it to the other CPU. The format of the I/O instruction is shown in Figure 10-1.

Although CPUs can interrupt each other, the CPUs do not have any interrupt stacking capability or visibility to the "Resume Interrupts" bus signal (as in I/O controllers). Thus software using the interrupts for communications between CPUs must check that the interrupts are acknowledged and be prepared to retry the interrupt until it is accepted.

To avoid having to retry the interrupt, the interrupting CSS can set the appropriate Activity Flag (AF) of the CSS to be interrupted by using a read-modify-write operation. The interrupting CSS then sends an interrupt to the CSS to be interrupted. If the interrupt is accepted, the interrupted CSS has honored the interrupt. If the interrupt is not accepted, the CSS to be interrupted is currently processing another interrupt or is executing at a higher interrupt level. Since the appropriate AF has been set, the interrupting CSS can assume that the CSS to be interrupted will honor the interrupt as soon as its execution level decreases to the appropriate level set by the interrupting CSS.

NOTE

The M5X does not synchronize with Read Modify Write when accessing the IAF area.

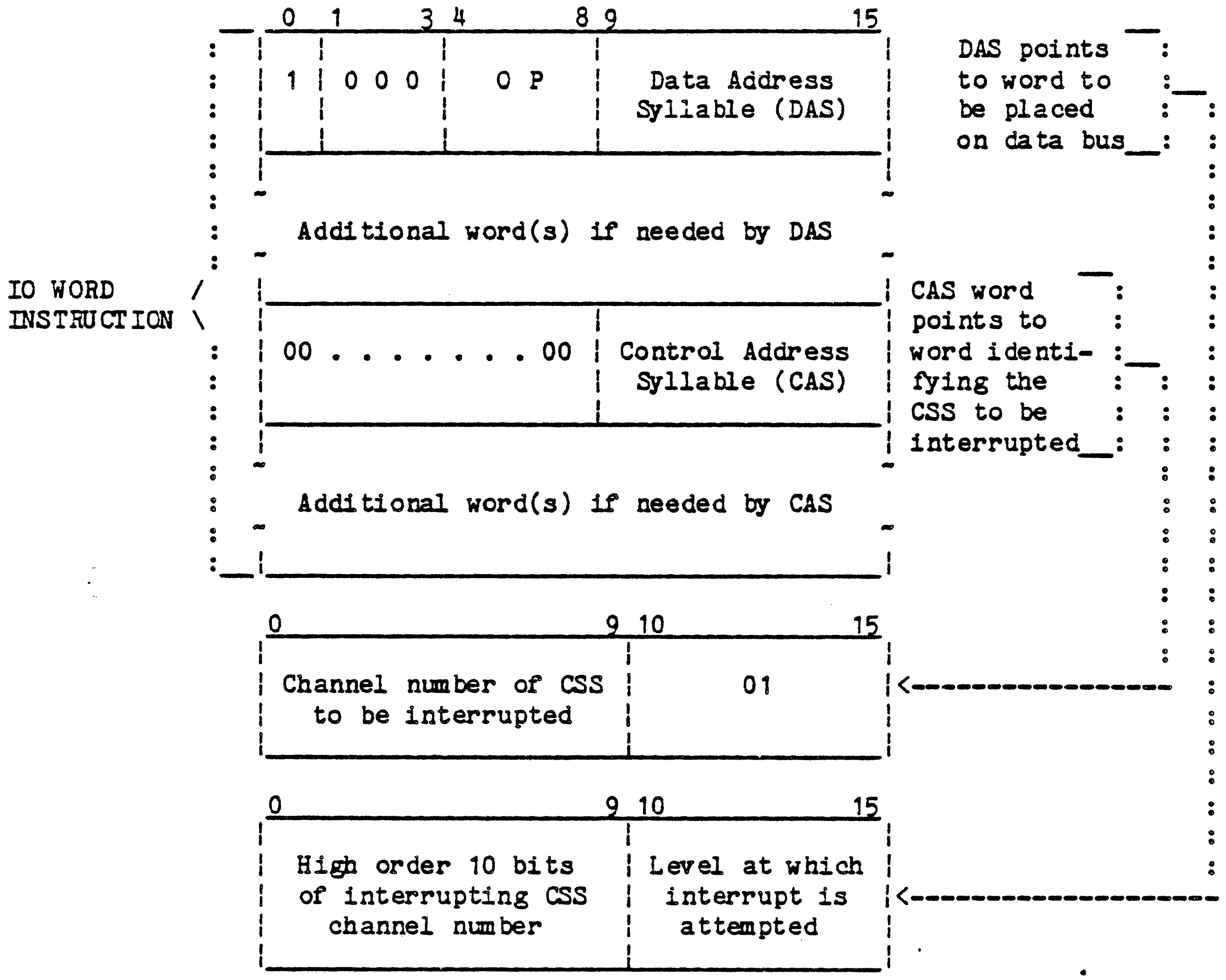

0 1 3 4 8 9 15
1
0 0 0
O P
Data Address Syllable (DAS)
DAS points to word to be placed on data bus
Additional word(s) if needed by DAS
IO WORD INSTRUCTION
00 00
Control Address Syllable (CAS)
CAS word points to word identifying the CSS to be interrupted
Additional word(s) if needed by CAS
0 9 10 15
Channel number of CSS to be interrupted
01
0 9 10 15
High order 10 bits of interrupting CSS channel number
Level at which interrupt is attempted

Figure 10-1 Format of I/O Instruction Used to Generate a CSS-to-CSS Interrupt

10.1.8 Read-Modify-Write (RMW) Operations

All CSSs have instructions that perform Read-Modify-Write (RMW) operations on memory locations. Cooperating procedures in different CSSs can use these instructions to "lock" and "unlock" shared data structures by accessing a predefined "LOCK" memory location. The read-modify-write instructions are:

- o Increment (INC)
- o Decrement (DEC)
- o Load Bit and Set True (LBT)
- o Load Bit and Set False (LBF)
- o Load Bit and Complement (LBC)
- o Load Bit and Swap (LBS).

Instructions that use an RMW operation to access a Lock in main memory are:

- o Queue on Head (QOH)
- o Queue on Tail (QOT)
- o Dequeue from Head (DQH)
- o Dequeue on Address (DQA)
- o Search Queue from Head (SQH)
- o Search Queue by Address (SQA).

The double "Lock" provided by these instructions, that is, RMW operation plus Lock in main memory, ensures access to the Lock by only one CSS at a time.

The M6X and M6XE use a RMW operation when addressing Activity Flags in dedicated memory.

10.1.9 Double Word (32 Bit) Operand Coherency

Within the M6X and M6XE CSSs, the caches are organized on a single-word basis (16 bits), and the possibility exists that half or all of a requested double word operand is present in cache memory, coincident with a current memory write operation to the same physical location by a second CSS. Due to the cache update pipeline delays and parallelism inherent in the memories/Megabus, the operand retrieved, for the case of the entire double word already encached, will be the 'old contents' of main memory ('old' by less than a few hundred nanoseconds). To prevent a situation where the double word operand retrieved contains half 'old' data and half 'new' data, the CSS/cache treats all double word (DBWD) requests as either a 'Full Hit' or 'Full Miss' during cache look-up.

To guarantee data coherency in double word operand instructions, the following programming rules must be obeyed:

- o Even word or odd word starting addresses are allowed, but double word operands must not end on a Modulo 16 word boundary due to memory interleaving (access to each Mod-16 word block of memory addresses different memory controllers).

- o Overlapped double words (word pairs) are NOT allowed, as shown in Figure 10-2.

NOTE

Use of even word starting addresses is the best practical way to avoid problems (i.e., memory interleaving and overlapped operands.

Double word coherency functions independently from read-modify-write-type operations.

Note that the M5X and M5XE do not support this functionality in the cache.

```
                |<--DWD Operand # 1-->|
    ILLEGAL:    | 1st Word | 2nd Word |
                |          |          |
                |__________|__________|__________
      ...       |  Word A  | Word A+1 | Word A+2 |...
                |          |          |          |
                |__________|__________|__________|
                           |          |          |
                           | 1st Word | 2nd Word |
                           |<--DWD Operand # 2-->|

                |<--DWD Operand # 1-->|
    LEGAL:      | 1st Word | 2nd Word |
                |          |          |
                |__________|__________|_____________________
      ..        |  Word A  | Word A+1 | Word A+2 | Word A+3 |...
                |          |          |          |          |
                |__________|__________|__________|__________|
                                      |          |          |
                                      | 1st Word | 2nd Word |
                                      |<--DWD Operand # 2-->|
```

Figure 10-2 Overlapped Double Words

10.1.10 I/O Considerations

I/O instructions can be issued by any CSS to any I/O channel (or other CSS). Interrupts, however, are always returned to the CSS specified in the I/O channel's control word (containing the channel number of the CSS to be interrupted and the interrupt level). A CSS does not apply its channel number to the data transmited to the controller during an Output Interrupt Control command, and, if this is desired, it must be performed by software. Therefore, I/O operations can be initiated from one CSS and terminated (interrupt handling) on the same or different) CSS in a multi CSS system, according to the software convention used.

Because the I/O instructions can cause multiple bus sequences on the Megabus, it is the software's responsibility to insure via software means that I/O channels are addressed by only one CSS at a time. If more than one CSS is allowed to address the same I/O channel without a software interlock mechanism, unspecified results can occur.

In addition, note that the MDC and MLC16 controllers require that all their channels be addressed by the same CSS.

10.1.11 Coexistance of Different CSS Models

Configurations using mixtures of different CSS Models are not recommended. Due to the differing physical characteristics in areas such as physical address size and other limitations from one model to the next, anomolous behavior may result. Table 10-3 describes the potential for configuring such L6 systems.

Table 10-3 Multiprocessor Coexistance on the System Bus

Generic Model Name	M5X	M6X	M6XE	CR41E	CR41	M5XE
M5XE	NO	NO	NO	NO	NO	YES* (up to 4)
CR41	NO	NO	NO	NO	NO	
CR41E	NO	NO	NO	NO		
M6XE	NO	NO	YES (up to 2)			
M6X	NO	YES (up to 2))				
M5X	YES* (up to 4)					

* Datanet 8 Applications only.

NOTE

These rules apply only to tightly coupled multiprocessor environments.

10.2 INTERFACES

10.2.1 System Bus

The Stage 3 CSSs attach to the system bus in fixed positions and use the bus to communicate with the I/O controllers and other CSSs as follows:

o Standard Megabus (CR41, CR41E, M5X, and M5XE models)

o Extended Megabus (M6X and M6XE models).

The CSSs adhere to the Megabus functionally as described in the Extended Megabus EPS-1 and the MRX Megabus EPS-1 (document numbers 60126298 and 60149832, respectively).

10.2.2 Control Panel

The control panel interface provides control panel visibility. It is a private interface and is described in the SCF EPS-1, document number 60139142. The master CSS uses this interface to communicate with the SCF. The slave CSS uses this interface to communicate with the full control panel.

10.2.3 QLT

The QLT interface provides QLT information in the M6X and M6XE CSSs. The master CSS uses the interface to communicate with the SCF. The slave CSS uses this interface to communicate with the QLT display that is mounted inside the cabinet.

10.3. RESILIENCY

Dual M6X and M6XE systems provide no specific features for higher resiliency. Recovery from a CSS failure is possible only if the following conditions are met:

- o The failure is not in the master CSS
- o The system is rebooted, and
- o A T&V program verifies that the faulted slave CSS is not interfering with the system operation.

This page is intentionally blank.

SECTION 11 ENVIRONMENTAL AND PHYSICAL STRUCTURE

11.1 PHYSICAL STRUCTURE

The Central Subsystem is contained on one or more mother/daughter board combinations. Processor functionality is configured for the various systems as follows.

11.1.1 CR41 and CR41E Physical Structure

The CR41 and CR41E CSSs each occupy one standard Megabus slot, offering 5XCPU, MMU, CIP, and SCF with on-board memory options of 512K bytes, 1 Megabyte (using 64K bit RAMs), 2 Megabytes, and CR41E, 4 Megabytes (using 256K RAMs).

11.1.2 M5X and M5XE Physical Structure

The M5X and M5XE CSSs require a Megabus slot each for cache, CPU, CIP, SCF, and optional SIP, respectively. Main memory is configured separately in the bus and occupies one slot. Memory sizes are 512KB, 1MB, and 2MB; and for M5XE, expansions of 4MB and 8MB are possible.

11.1.3 Monoprocessor M6X and M6XE Physical Structure

The M6X and M6XE CSSs are configured in a 10-slot 'Local Bus' chassis and connect to the system bus via one bus slot, and a second bus slot is reserved for the SCF. The Local Bus chassis provides for 6X CSS functionality of the CPU, cache, CIP, SIP, and SMMU, and for M6XE, the EMMU is included. Memory requires two bus slots and offers capacities of 2MB, 4MB, 8MB, 12MB and 16MB.

11.1.4 Dual Processor M6X and M6XE Physical Structure

The Dual processor configurations of the M6X and M6XE require two 10-slot local bus chassis's and two system bus slots (SCF and memory requirements are identical to the monoprocessor versions). Dual tightly-controlled multiprocessor functions are provided. See Section 10 for usage.

11.1.5 Generic Stage 3 I/O Physical Structure

Refer to Section 14 for System Configurations of the various Stage 3 CSSs.

11.2 ENVIRONMENTAL CONDITIONS

11.2.1 Operating

11.2.1.1 SUBSYSTEM LEVEL OPERATING ENVIRONMENT

All non-Mechanical Assemblies (i.e. Electronic/Logic PWAs) are designed to comply with HIS Standard B01.08 Class 3, with the following exceptions:

- o Contamination requirement waived
- o Non condensation conditions.

11.2.1.2 SYSTEM LEVEL OPERATING ENVIRONMENT

Per HIS Standard B01.08 Class 2 (modified) as follows:

- o Temperature - 50-100.4^{0}F
- o Relative Humidity - 20-80%, non-condensing
- o Contamination requirement waived.

11.2.1.3 NON-OPERATING

Per HIS Standard B01.10

11.2.2 Electrical Characteristics

The Stage 3 systems are capable of operating with either 60- or 50-Hz power systems, as specified below:

- o 60-Hz Power System, Voltage Group 2:
  - Voltage: 120/208 Vac, +10%, -15%, Single/Multiple phase as required by the specific Configuration.
  - Frequency: 60 Hz $\pm$ 0.6 Hz
  - Other Power Requirements: As specified in HIS Standard B01.48.

- 50-Hz Power System, Voltage Group 1:
  - Voltage: 220/380 Vac, +10%, -15%, or240/415 Vac, +6%, -15%, Single/Multiple phase as required by the specific Configuration
  - Frequency: 50 Hz $\pm$ 0.5 Hz
  - Other Power Requirements: As specified in HIS Standard B01.48.
  - For the European CII-HB market a UPS or MG set is required to meet B01.48. section 6.2, compliance 2.

11.2.3 Electromagnectic Interference (EMI)

The EMI levels must comply with U.S. FCC Class A standards for data processing equipment. EMI tests will be done in accordance with HIS Operating Standard B01.08 (Environment).

11.2.4 Electrostatic Discharge (ESD)

The ESD levels comply with HIS Standard B01.08, paragraph 9, "Electrostatic Discharge."

11.2.5 Safety

The systems will comply with HIS Standard B01.09 requirements for safety.

11.2.6 Audible Noise

All sources of audible noise should be minimized to conform to office environment standards per B01.08. The possible sources of audible noise are as follows:

- Printing
- Motor (disk drives)
- Cooling fans

11.2.7 Power Distribution and Pakaging Requirements

All Stage 3 DPS6 systems are configured using the standard Level 6/DPS6 30" and 60" Cabinetry and Power/Cooling assemblies.

A Power Distribution Unit (PDU), at the bottom of the cabinet, provides master circuit breaker protection, ac branch protection, ac filtering, cabinet on/off control and ac power distribution.

This page is intentionally blank.

SECTION 12 PERFORMANCE

12.1 PERFORMANCE GOALS

Table 12-1 gives the performance goals for the Stage 3 DPS6 CSSs.

Table 12-1 Processor Relative Performance Goals

SYSTEM	CPU	COMMERCIAL	SCIENTIFIC
CR41	1.0	1.0	1.0
CR41E	1.0	1.0	1.0
M5X	1.7	1.8	2.4
M5XE	1.7	1.8	2.4
M6XS	3.0	4.0	6.5
M6XES	3.0	4.0	6.5
M6X	4.4	4.2	7.5
M6XE	4.4	4.2	7.5
Dual M6X	8.1	7.6	13.4
Dual M6XE	8.1	7.6	13.4

This page is intentionally blank.

SECTION 13 AVAILABILITY/MAINTAINABILITY STRATEGY

13.1 MAINTAINABILITY

The Stage 3 CSSs are partitioned into a set of mother/daughter boards and are composed of the following units:

- o Conventional logic found in present DPS 6 packaged systems
- o Proprietary and Non-proprietary VLSI microprocessor components.

13.2 MAINTENANCE STRATEGY

The maintenance strategy for the Stage 3 CSS consists of partitioning it into one- or two-board ORUs that can be effectively diagnosed for a faulty condition via a combination of diagnostic techniques. These diagnostic aids are executable on site by the customer or customer service engineer, or remotely by a customer service engineer at the Honeywell Field Repair Center.

Simple repairs, such as the replacement of a defective ORU with an operational one, can be carried out by trained customer personnel or a service engineer.

13.3 SYSTEM CONTROL FACILITY

The System Control Facility (SCF) is an integral part of the DPS 6 Stage 3 systems, serving as a control panel and console and providing remote maintenance capability for these systems.

For the CR41 and the CR41E, the SCF consists of the following:

- o Control Panel
- o VCP
- o Modem
- o Local terminal as console.

For the remaining Stage 3 CSSs, the SCF consists of the following:

- o Control Panel
- o MPU
- o Modem
- o Local terminal as console.

The VIP7300 terminal or equivalent, with properly labeled function keys on the keyboard, provides local or remote control and display of status and/or maintenance information.

The SCF becomes a prime tool of the Technical Assistance Center (TAC) to facilitate the resolution of software, operational, and hardware faults in a more rapid manner than by telephone conversation followed by on-site calls.

The multifaceted characteristics of the SCF support the following functions:

- o Control panel functionality
- o Systems console functionality
- o Local and remote maintenance functionality.

13.4 MAINTAINABILTY FEATURES

The primary test method is the use of traditional Quality Logic Tests (QLTS). The secondary level of testing is via T&V software.

13.5 VIRTUAL CONTROL PANEL

System support in the Stage 3 systems is provided by the VCP located in the CSS in conjunction with the locally connected and remotely connected VIP7300 display terminals. This combination of elements performs System Control Facility (SCF) functionality.

SCF Functionality Provided in all Systems

- o L6 basic control panel and display terminal
- o Status information
- o Local and remote maintenance information
- o EIA RS-422 direct connect interface to locally connected terminals
- o EIA RS-232C interface to modem for remote terminal connections.
- o Auxiliary printer interface
- o Battery backup support.

Additional Functionality Not Provided in CR41 and CR41E

- o Power marginal checking

In multiprocessor systems, the SCF connects to the master CSS (CSS0) only. A full control panel is provided to gain access to the slave CSS.

Twelve special function keys in the top row of the display terminal (VIP7300) keyboard, labeled to correspond to the full control panel, generate codes that are interpreted along with certain standard keys by the VCP. The VCP responds to these keycodes by performing the equivalent functions that are performed when the corresponding keys are activated on a Level 6 full control panel.

The control panel display, visible on line 25 of the VIP7300, is updated under VCP control after each keystroke, as appropriate. The control panel display provides the following:

- o Status information
- o Maintenance information such as register and memory contents, etc.
- o Commands and messages to the operator.

NOTE

In an M5XE with EMMU selected, all address register displays will be six digits (24 address bits), as in the M6XE. However the number of registers supported are as in a M5X.

For a more in depth description of the SCF functionality refer to EPS-1 60139142.

This page is intentionally blank.

SECTION 14 CONFIGURATIONS

Information relative to the suggested DPS6 Stage 3 configuration is presented below. Note that the bus allocation diagrams in subsection 14.2 are not intended to depict all possible variations, but are the recommended set based on the DPS6 Packaged System strategy. However, any variances must be configured according to the prevailing Megabus rules and restrictions.

14.1 I/O CONFIGURABILITY

14.1.1 Glossary of Terms

CR41	- One-slot CSS with integrated CPU, SCF, CIP, and up to 2MB memory
CR41E	- CR41 with up to 4MB memory
HPDC	- High Performance Disk Controller. Host for SMD/CMD or FSD drives on optional adapters. Mother board has two versions: Reversed Prinet (current 6X disk controller) or Standard Prinet (new MRX-HPDC), and are not interchangeable.
HSDC	- 16-bit disk controller hosting FSD drives; includes optional floppy and 1/4" Sentinal Streamer attachments
LAN	- Local Area Network controller featuring ONMINET and Ethernet/IEEE CSMA/CD connections initially
LARK	- Larkette Controller.

M5X	-	Current CSS with separate CPU, SCF, CIP, SIP, cache, and 2MB memory
M5XE	-	M57 with up to 8MB memory
M6X	-	Corrent CSS with separate CPU, SCF, CIP, SIP, cache and 16MB memory
M6XS	-	Slow version of M6X
M6XE	-	M6X with optional 5X MMU or EMMU and up to 16MB memory (both M6 and M6E will be supported in mono and dual processor configurations by M400 R4)
M6XES	-	Slow version of M6XE
MDC	-	Multiple Device Controller. Hosts standard unit record devices via version III firmware and reader/sorter devices via BDC5 and BDC7 firmware. Note that new 1985 firmware releases of all supported versions of MDC-based devices is required in dual processor M6X and M6XE systems.
MLC16	-	Current 16-port serial controller (supports both FLAPs and RS422, 232 FLAP-less connections)
MPDC	-	Medium Performance Disk Controller. Hosts SMD/CMD disk (16-bit Megabus).
MSC	-	Current SMD/CMD disk controllers
MTC	-	Current NRZI, PE and GCR tape controllers.

14.1.2 I/O Controller Configurability

Listed below are the controllers that are allowed to be configured in dual 6X systems; all others are excluded.

o Disk I/O:
- Disk Controllers: Larkette, MPDC ('DC1'), HPDC, HSDC, and WREN (4Q86).
- Disk Devices: See Table 14-1.

Table 14-1 Disk Devices Supported

DISK INTERFACE	DRIVES SUPPORTED	REMARKS
LMD	Lark 2	10 MHz, 10' Cable Max.
SMD1H	SMD, CMD	Honeywell 'Special' 10MHz, 50' Cable Max.
SMDO	FSD I, FSD II	Industry standard, 15 MHz, 50' Cable Max.
SMDO-E	EMD I	Industry standard, 15 MHz, 50' Cable Max.
ESDI	Wren II, III	Industry standard, 10 MHz, 10' Cable Max.

- Disk Device Interfaces: See Table 14-2.

o Communications Controllers Supported - MLC16, LACS, L66 FEP Coupler.

o Unit Record Controllers Supported - MDC-III, BDC-5 (234-XYZ RDR-Sorter), MDC-7 (234-0 RDR/Sorter), BDC-3 (NRZI/PE Tape), MDC-II (8" Floppy).

Table 14-2 Disk Device Interfaces Supported

CONTROLLER	LMD	SMD1H	SMD0	SMD0-E	ESDI
Larkette	YES	NO	NO	NO	NO
WREN	NO	NO	NO	NO	YES
MPDC	NO	YES	NO	NO	NO
HPDC	NO	YES	YES	YES	NO
HSDC	NO	NO	YES	YES	NO

14.2 MEGABUS SLOT ASSIGNMENTS

Figures 14-1 through 14-6 give the Megabus slot assignments for the various Stage 3 systems.

14.2.1 DPS6/40E

- o Available 1Q86.
- o Not Field Upgradable.
- o CSS Performance = 1.0.
- o Standard MMU, only.
- o Memory Size Options of 2MB or 4MB.
- o Configuration: 30" System Cabinet with 5 Slots, and 2 X 8" Winchester Disk Drives.

```
 ______                                                          _
|    05|--- STANDARD* HSDC DISK or SIP (OPTION SLOT)              |
|      |--- STANDARD* HSDC or LARKETTE or SMD/CMD or MDC (OPTION SLOT)|
|      |--- MDC or NMLC (OPTION SLOT)                             > 5  CARD
|      |--- MLC16 4-RS422                                         |  CHASSIS
|____01|--- CR41E CSS (SCF, MEMORY, CIP)                         _|
```

* Standard HSDC disk includes 4 SMD0 disk ports, standard floppy and optional streamer tape.

Figure 14-1 DPS6/40E Megabus Assignments

14.2.2 DPS6/45E

- o Available 1Q86.
- o Field Upgradable to Models 75E, 85E.
- o CSS Performance = 1.0.
- o Standard MMU, only.
- o Memory Size Options of 2MB or 4MB.
- o Configuration: 60" System Cabinet with 20 Slots and additional Peripheral Cabinets accomodating FSDs or 8" Winchester Disk Drives.

```
 ___________                                                                -
|         20|--- LACS (OPTION SLOT)                                          |
|           |--- SIP or HSDC or LARKETTE or SMD/CMD (OPTION SLOT)            |
|           |--- HSDC or LARKETTE or SMD/CMD (OPTION SLOT)                   |
|           |--- HSDC DISK, FLOPPY, (OPTIONAL STREAMER)                      |
|           |--- MDC (OPTION SLOT)                                           |
|         15|--- GCR/PE (OPTION SLOT)                                        |
|           |--- MLC16 (OPTION SLOT)                                         |
|           |--- MLC16 4-RS422                                               |
|           |--- CR41E CSS (SCF, MEMORY, CIP)                                |
|           |--- TERMINATOR*                                                  > 20 CARD
|         10|--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|   CHASSIS
|           |--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
|           |--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
|           |--- +++++++++++++++                          +++++++++++++++++++|
|           |--- +++++++++++++++    REGION NOT POWERED    +++++++++++++++++++|
|         05|--- +++++++++++++++                          +++++++++++++++++++|
|           |--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
|           |--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
|           |--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
|_________01|--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
```

* The terminator board takes up a functional slot due to the lack of power in the normal "terminator-only" slot, below.

Figure 14-2 DPS6/45E Megabus Assignments

14.2.3 DPS6/70E

o Available 1Q86.

o Not Field Upgradable.

o CSS Performance = 1.7.

o Memory Management**: Standard MMU or Extended MMU.

o Memory Size Options of 2MB, 4MB, 6MB or 8MB.

o Configuration: 30" System Cabinet with 10 Slots, with integrated 8" Winchester Disk Drives and additional Peripheral Cabinets accomodating FSD or 8" Winchester Disk Drives.

```
 ________                                                                _
|      10|--- SCF                                                         |
|        |--- M5XE-CACHE                                                  |
|        |--- M5XE-CSS                                                    |
|        |--- M5X-CIP                                                     |
|        |--- M5X-SIP (OPTION SLOT)                                       > 10 CARD
|      05|--- LACS or STANDARD HSDC* (OPTION SLOT)                        |  CHASSIS
|        |--- STANDARD HSDC* or LARKETTE or SMD/CMD or MDC (OPTION SLOT)|
|        |--- MDC or MLC16 (OPTION SLOT)                                  |
|        |--- MLC16 4-RS422                                               |
|______01|--- MZG MEMORY                                                 _|
```

** If M400 R4.0 is configured then only EMMU mode will be active.

* Standard HSDC disk includes 4 SMDO disk ports, standard floppy and optional streamer tape.

Figure 14-3 DPS6/70E Megabus Assignments

14.2.4 DPS6/75E

- Available 1Q86.
- Field Upgradable to Models 85E, 95E.
- CSS Performance = 1.7.
- Memory Management*: Standard MMU or Extended MMU.
- Memory Size Options of 2MB, 4MB, 6MB or 8MB.
- Configuration: 60" System Cabinet with 20 Slots and additional Peripheral Cabinets accomodating FSDs or 8" Winchester Disk Drives.

```
 ______                                                              _
|    20|--- SCF                                                       |
|      |--- M5XE-CACHE                                                |
|      |--- M5XE-CSS                                                  |
|      |--- M5X-CIP                                                   |
|      |--- M5X-SIP (OPTION SLOT)                                     |
|    15|--- LACS (OPTION SLOT)                                        |
|      |--- DISK CACHE or LACS (OPTION SLOT)                          |
|      |--- HSDC DISK or LARKETTE or SMD/CMD (OPTION SLOT)            |
|      |--- HSDC DISK (OPTION SLOT)                                   |
|      |--- HSDC DISK, FLOPPY, (OPTIONAL STREAMER)                    > 20 CARD
|    10|--- MDC or GCR/PE TAPE (OPTION SLOT)                          | CHASSIS
|      |--- MDC or DOC HANDLER (OPTION SLOT)                          |
|      |--- MLC16 (OPTION SLOT)                                       |
|      |--- MLC16 (OPTION SLOT)                                       |
|      |--- MLC16 (OPTION SLOT)                                       |
|    05|--- MLC16 (OPTION SLOT)                                       |
|      |--- MLC16 (OPTION SLOT)                                       |
|      |--- MLC16 4-RS422                                             |
|      |--- DISK CACHE BUFFER (OPTION SLOT)                           |
|____01|--- MZG MEMORY                                               _|
```

* If M400 R4.0 is configured then only EMMU mode will be active.

Figure 14-4 DPS6/75E Megabus Assignments

14.2.5 DPS6/85E

- o Available 1Q86.
- o Field Upgradable to Model 95E.
- o CSS Performance = 3.0.
- o Memory Management*: Standard MMU or Extended MMU.
- o Memory Size Options of 2MB, 4MB, 6MB or 8MB.
- o Configuration: 60" System Cabinet with 20 Slots (15 powered) and additional Peripheral Cabinets accomodating FSDs or 8" Winchester Disk Drives.

```
 _______                                                                  _
|     20|--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
|       |--- +++++++++++++++                          +++++++++++++++++++|
|       |--- +++++++++++++++   REGION NOT POWERED    +++++++++++++++++++|
|       |--- +++++++++++++++                          +++++++++++++++++++|
|       |--- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
|     15|--- TERMINATOR**                                                 |
|       | -- LACS (OPTION SLOT)                                           |
|       |--- DISK CACHE or HSDC DISK (OPTION SLOT)                        |
|       |--- HSDC DISK, FLOPPY, (OPTIONAL STREAMER)                       |
|       |--- HSDC DISK or LARKETTE or SMD/CMD (OPTION SLOT)               |
|     10|--- MDC or GCR/PE TAPE or DOC HANDLER (OPTION SLOT)              > 20 CARD
|       |--- MDC or DOC HANDLER (OPTION SLOT)                             | CHASSIS
|       |--- MLC16 (OPTION SLOT)                                          |
|       |--- MLC16 (OPTION SLOT)                                          |
|       |--- MLC16 (OPTION SLOT)                                          |
|     05|--- MLC16 4-RS422                                                |
|       |--- MXG MEMORY (OPTION SLOT)                                     |
|       |--- MXG MEMORY-BANKED                                            |
|       |--- SCF                                                          |
|_____01|--- CSS MBA                                                     _|

 _______                                                                  _
|       |--- ///////////////////////////////////////////////////////////////|
|       |--- ///////////////////      SLOW M6XE     ///////////////////////|
|       |--- ///////////////////         CPU        ///////////////////////|
|     05|--- ///////////////////         CIP        ///////////////////////|
|       |--- ///////////////////         SIP        ///////////////////////|
|       |--- ///////////////////        CACHE       ///////////////////////|
|       |--- ///////////////////    & 6X ASSEMBLY   ///////////////////////|
|_____01|--- ///////////////////////////////////////////////////////////////|
```

* If M400 R4.0 is configured then only EMMU mode will be active.

** The terminator board takes up a functional slot due to the lack of power in the normal "terminator-only" slot, below.

Figure 14-5 DPS6/85E Megabus Assignments

14.2.6 DPS6/95E

- o Available 1Q86.
- o Not Field Upgradable.
- o CSS Performance = 4.5.
- o Memory Management*: Standard MMU or Extended MMU.
- o Memory Size Options of 4MB, 8MB, 12MB or 16MB.
- o Configuration: 60" System Cabinet with 30 Slots and additional Peripheral Cabinets accomodating FSDs or 8" Winchester Disk Drives.

```
 ______                                                   _
|    30|--- CSS1 MBA (OPTION SLOT)                         |
|      |--- DISK CACHE (OPTION SLOT)                       |
|      | -- (OPTION SLOT)                                  |
|      |--- LARKETTE or SMD/CMD (OPTION SLOT)              |
|      |--- GCR/PE TAPE (OPTION SLOT)                      |
|    25|--- GCR/PE TAPE (OPTION SLOT)                      |
|      | -- MDC/FLOPPY                                     |
|      |--- MDC or DOC HANDLER (OPTION SLOT)               |
|      |--- MDC or DOC HANDLER (OPTION SLOT)               |
|      |--- MLC16 (OPTION SLOT)                            |
|    20|--- MLC16 (OPTION SLOT)                            |
|      |--- MLC16 (OPTION SLOT)                            |
|      |--- MLC16 (OPTION SLOT)                            |
|      |--- MLC16 (OPTION SLOT)                            |
|      |--- MLC16 (OPTION SLOT)                            |
|    15|--- MLC16 (OPTION SLOT)                            > 30 CARD
|      |--- MLC16 4-RS422                                  | CHASSIS
|      |--- SCF                                            |
|      |--- MXG MEMORY (OPTION SLOT)                       |
|      |--- MXG MEMORY (OPTION SLOT)                       |
|    10| -- BUS CONNECTION                                 |
|      |--- MXG MEMORY-INTERLEAVED                         |
|      |--- MXG MEMORY INTERLEAVED                         |
|      | -- LACS (OPTION SLOT)                             |
|      | -- LACS (OPTION SLOT)                             |
|    05|--- HPDCX DISK                                     |
|      |--- HPDCX DISK (OPTION SLOT)                       |
|      |--- HPDCX DISK (OPTION SLOT)                       |
|      |--- HPDCX DISK (OPTION SLOT)                       |
|____01|--- CSS0 MBA                                      _|

 ______                                                   _
|      |--- ////////////////////////////////////////////////////|
|      |--- //////////////////        M6XE       ////////////////////////|
|      |--- //////////////////        CPU        ////////////////////////|
|    05|--- //////////////////        CIP        ////////////////////////|
|      |--- //////////////////        SIP        ////////////////////////|
|      |--- //////////////////       CACHE       ////////////////////////|
|      |--- //////////////////   & 6X ASSEMBLY   ////////////////////////|
|____01|--- ////////////////////////////////////////////////////_|
```

* If M400 R4.0 is configured then only EMMU mode will be active.

Figure 14-6 DPS6/95E Megabus Assignments

SECTION 15 WORLDWIDE MARKETING REQUIREMENTS

15.1 AC INPUT POWER

All DPS6 Stage 3 systems operate in accordance with the requirements set forth in this specification when connected to ac power sources of either 50 or 60 Hertz, Voltage Groups 1 & 2, per subsection 11.2.2. Systems are factory-configured for either 50- or 60-Hertz operation as a function of their final destination.

Peripherals such as a disk drive or tape drive whose motor rotation speed is sensitive to line frequency are accommodated by pulleys/gears to maintain proper operation.

Real-time clocks and interval timers whose timing is derived from line frequency will exhibit different outputs when driven from 60 or 50 Hertz. The operating system accounts for this difference at initialization time.

15.2 INTERNATIONAL KEYBOARDS

The VIP7300 (or equivalent) terminal, used as a console with the SCF, is planned to have international keyboards available for overseas markets.

15.3 INTERNATIONAL LANGUAGES

The VIP7300 (or equivalent) terminal is planned to have available international character generators. Software and firmware packages are planned for displaying status, maintenance, and operator messages on the 25th line in the language of the host country.

This page is intentionally blank.

APPENDIX A MAIN MEMORY DIAGNOSTIC FUNCTIONALITY

A.1 SCOPE

This appendix describes the Main Memory diagnostic functionality supported by the M6X and M6XE. Main Memory diagnostic functionality supported by the CR41E and M5XE is TBD. The following features are supported:

- o Read Memory ID - Allows identification of memory type, size address mode and other information.
- o Read Memory Status - Allows reporting of memory error information for isolation of RAM failures.
- o Automatic testing of EDAC Logic - Allows testing of EDAC logic without requiring operator intervention.
- o Address reconfiguration - Allows the following types of memory reconfiguration functions to be performed:
  - Half board and quarter board swap
  - Memory controller module number can be dynamically changed to any value within the addressing space
  - A failed memory controller can be placed offline, communicated with, and put back online.

Additionally, half and quarter controllers can be placed offline.

- o "Here I Am" Light - Allows the T&V to physically identify a failing memory controller for field personnel by turning on that memory controller's "Here I Am" light.

- o Soft Error Rewrite Control- Allows the activation or deactivation and cycle speed selection of the rewrite logic.

A.2 MAIN MEMORY DIAGNOSTIC COMMAND CODES

When a memory controller is to perform a diagostic operation, the following conditions must be satisfied:

- o The proper module ID must be transmitted:
  - For banked mode (64K chip) - Address bits 0 through 3
  - For interleaved moe (64K chip) - Address bits 0 through 2 and bit 18
- o Memory reference must be True
- o A Read command for <u>one word</u> must be used
- o Yellow must be true.

If these conditions are satisfied, then address bits 19, 20 and 21 will be decoded to determine the type of diagnostic operation being requested. The decoded diagnostic commands are as follows:

ADDRESS BITS			TYPE OF OPERATION
19	20	21	
0	0	0	Read ID Word
0	0	1	Read Status Word
0	1	0	Set EDAC Mode
0	1	1	Clear EDAC Mode
1	0	0	Address Reconfiguration
1	0	1	"HERE I AM" Light Control
1	1	0	Soft Error Rewrite Control
1	1	1	RFU

Note that the above text pertains to the bus dialog and is not independent of the software and firmware used to initiate the action (see subsection TBD).

A.2.1 Read ID Word

When this diagnostic code is received, the memory reads the location being addressed. However, the contents of the ID register are substituted for the data and transmitted to the register. The ID word is sent on the data bus (bits 0:15). Bits 0-3 of the ID, define the memory module type and bits 4-15 are memory module type specific. The following applies:

```
o  ID( 0- 3) = 0000, Memory Module Type is BFMMXE

   - ( 4- 5) = RFU = 00
   - (    6) = Chip type: 0 = 64K chip; 1 = 16K chip
   - (    7) = Array pacs swapped (upper <-> lower) if set
   - (    8) = EDAC in test mode if set
   - (    9) = Soft error rewrite in test mode if set
   - (   10) = Bus error if set
   - (11-14) = Array pac 0 through array pac 3 present if set
   - (   15) = Interleaved if set


o  ID( 0- 3) = 0001, Memory Module Type is BF6MXF

   - ( 4- 5) = Hardware revision
   - (    6) = Chip type: 0 = 64K chip; 1 = 256K chip
   - (    7) = Double density array pacs if set
   - (    8) = EDAC in test mode if set
   - (    9) = Soft error rewrite in test mode if set
   - (   10) = Controller reconfigured if set
   - (   11) = Eighth  board swapped if set
   - (   12) = Quarter board swapped if set
   - (   13) = Half    board swapped if set
   - (   14) = Fully populated controller if set else half populated
   - (   15) = Interleaved if set


o  ID( 0- 3) = 0010, Memory Module Type is BF8MXG

   - ( 4- 5) = 00 - no retry attempted
             = 10 - retry was successful
             = 11 - retry failed
   - (    6) = Chip type: 0 = 64K chip; 1 = 256K chip
   - (    7) = Double density array pacs if set
   - (    8) = EDAC in test mode if set
   - (    9) = Soft error rewrite in test mode if set
   - (   10) = Controller reconfigured if set
   - (   11) = Mixed array pacs if set
   - (   12) = Quarter board swapped if set
   - (   13) = Half    board swapped if set
   - (   14) = Fully populated controller if set else half populated
   - (   15) = Interleaved if set
```

A.2.2 Read Status Word

When this diagnostic code is received, the memory reads the location being addressed. However, the contents of the status register are substituted for the data and transmitted to the register. The status word is sent on the data bus (0:15) with the following format:

```
 0                          7         8                          15
 ______________________________       ______________________________
|   |   |   |   |   |   |   |   |     |   |   |   |   |   |   |   |   |
|A0 |A1 |S0 |S1 |S2 |S3 |S4 |S5 |     |A0 |A1 |S0 |S1 |S2 |S3 |S4 |S5 |
|___|___|___|___|___|___|___|___|     |___|___|___|___|___|___|___|___|
       EVEN WORD STATUS                      ODD WORD STATUS
```

where: A0 - Identifies lower/upper addressed daughter board

A1 - Identifies lower/upper addressed row of chips within a daughter board.

S0-S5 - EDAC syndrome bits (identifies failing bit within a word).

COMMENTS

- o The combination of address bits (A0 & A1) plus syndrome bits (S0-S5) allows isolation to the RAM failure.
- o The status register contains information related to the most recent single/double bit error.
- o The status register is cleared to Zero at the end of the read status operation.

A.2.3 Set EDAC Mode

When this diagnostic code is received, the memory will put itself into the EDAC test mode. It will then read the location being addressed and ship the data to the register. While in this mode, all 'BSRED' signals are inhibited during read cycles and the check bit field is forced to Zero during write cycles.

A.2.4 Clear EDAC Mode

When this diagnostic code is received, the memory turns off the EDAC test mode. It then reads the location being addressed and ships the data to the register.

A.2.5 Address Reconfiguration

When this diagnostic code is received, the memory reads the location being addressed. During the second half bus cycle, the data is shipped to the register and the reconfiguration information contained in the present address field is loaded to a special register. Upon receipt of subsequent bus cycles, the memory uses the register contents to determine its module identification. The reconfiguration register powers up with all Zeros (unused state), does not change with bus Master Clear, and retains its contents on power outages if the memory has a power save unit (BBU). When memory receives the address reconfiguration code (4 in address bits 19, 20 and 21), it interprets the remaining bits of the address field as follows:

```
   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23
 |R00|R01|R02|R03|OL1|OL0|RFU|HBS|QBS| I | X | 1 | 0 | 0 | X | X |
 |_______________|                       |___________|
         |                                     |
 Reconfigured Module Address            Configuration Code
```

where R00 = Will be the address bit value compared with BSAD00

R01 = Will be the address bit value compared with BSAD01

R02 = Will be the address bit value compared with BSAD02

R03 = Will be the address bit value compared with BSAD03 or BSAD18

OL0/OL1 = Offline Command Bits:
00 - Module on line
01 - Quarter module offline
10 - Half module offline
11 - Full module offline

HBS - Half Board Swap Bit:
0 = Normal
1 = Swap daughter board pair

QBS - Quarter Board Swap:
0 = Normal
1 = Swap half daughter board pairs

I - Interleaved Bit:
0 = Banked mode
1 = Interleaved mode

X = Don't care.

A.2.5.1 COMMUNICATION WITH MEMORY

Communication with memory controllers can be accomplished in either of two modes. First, in the normal or online mode, the memory receives its module address from the address bus, BSAD00, BSAD01, BSAD02 and BSAD03 for banked or BSAD18 for interleaved. Second, when the controller is in the maintenance (offline) mode, it responds to the proper module address value plus the values of address bus bits 6, 7 and BSYEL0.

A.2.5.1.1 Normal Mode (BSYELO = 0)

BSAD00	01	02	03	
A00	A01	A02	A03	Module Address for Banked Memories

BSAD00	01	02	18	
A00	A01	A02	A03	Module Address for Interleaved Memories

A.2.5.1.2 Maintenance Mode (BSYELO = 1)

BSAD00	01	02	03 or 18	04	05	06	07	
A00	A01	A02	A03	RFU	RFU	C	R	Module Address For Banked/Inter-leaved Memories in the On-Line/ Off-Line Mode

where: C - "Off-Line" command module ID Bit:
0 = Communication with on-line memories
1 = Communication with off-line memories

R - Default command bit:
0 = Normal mode
1 = Resets all controllers to switch settings.

A.2.5.2 MEMORY RECONFIGURATION

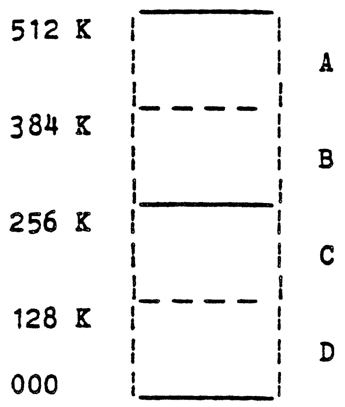
A.2.5.2.1 Half Board and Quarter Board Swap

All examples assume a 512-KW modlue fully populated with four memory PACs and 64 K RAMs. In this example, assume a double bit error occurs in location 000000, rendering the module and the system unusable. With the use of the Reconfiguration command, the memory PACs can be rearranged to put the defective location at the top of the memory controller.

```
                 ____________
512 K           |            |
                |            |  D   Memory Controller
                | _ _ _ _ _  |      A,B Lower Board Pair
384 K           |            |      C,D Upper Board Pair
                |            |  C
                |____________|
256 K           |            |
                |            |  B
                | _ _ _ _    |
128 K           |            |
                |            |  A
000             |____________|
```

When memory is sent a diagnostic code with the correct combination of half board and quarter board swap, it takes on of the following logical identity and moves the double bit error to the top of memory. The code sent is as follows:

1. BSYELO True
2. BSMREF True
3. Read One Word
4. Address (0000E8)

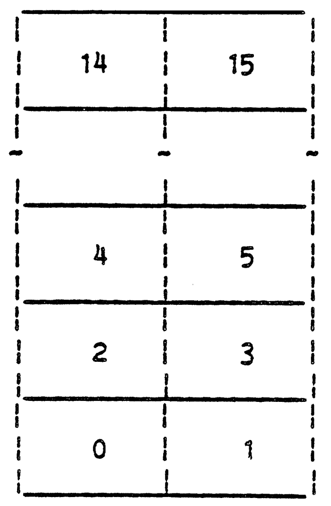


The result as shown is a rearranged module with defective memorypac at the top.

A.2.5.2.2 Address Reconfiguration

The following example of address reconfiguration is intended to Bhow a possible way in which a defective memory controller can be placed off-line and a fully functional module can be readdressed to fill the vacated module space. A system of 16 MBytes is given so that all available memory slots are used up.

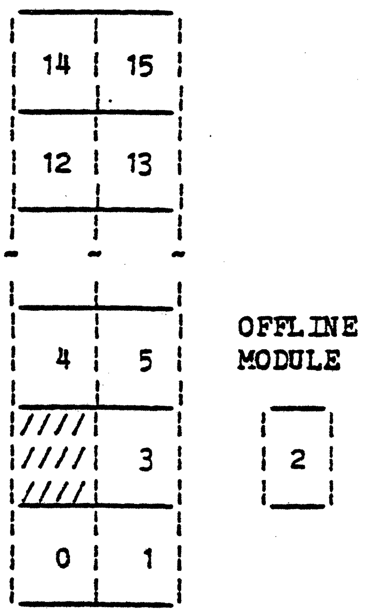
Assume the memory system to the right comprised of eight pairs of interleaved modules, each pair being two megabytes. Upon power-up, the T&V detects a problem in Module #2 which renders the entire module inoperative. If left alone, some means would be necessary to map around both modules #2 and #3, since they are configured interleaved. What can be done with the use of the Reconfiguration command is to put module #2 off-line, place module #14 in the space of module #2, and put module #15 in banked mode with a starting address of one location above module #13. The following steps illustrate this:



STEP 1

Move module #2 to off-line state:
a. BSYELO is true
b. Memory Reference true, BSMREF
c. Read one word
d. Address = 101628

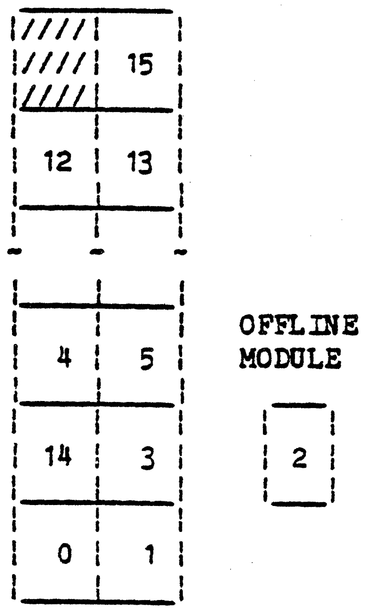
This command is interpreted by module #2 as a Reconfiguration command placing it off-line with the same ID.



STEP 2

Move module #14 to module #2 SPACE:
a. BSYELO is true
b. Memory Reference true, BSMREF
c. Read one word
d. Address = 701028

These instructions are interpreted by module #14 as a Reconfiguration command placing a module number of 2 in the reconfiguration register. The controller's ID now is determined by that value rather than the switches. The reconfigured modele #14 also is placed in interleaved mode so that it works with module #3.



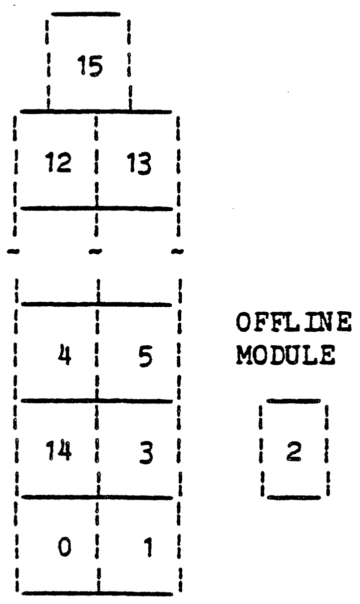
STEP 3

Give module #15 a starting address one location above module #13 and place in banked mode:

a. BSYELO is true
b. Memory Reference true, BSMRE F
c. Read one word
d. Address = 707038

This command is interpreted by module #15 as a Reconfiguration command. The module is put in banked mode and the module number is under control of the reconfiguration register whose value is contiguous from module #13.

From these steps, a defective module can be placed in "off-line" status, and other functional modules can be put in the defective slot so that memory space is contiguous and operational.



B In order to address a module in the "off-line" status condition (module #2 in our example), signals BSYELO and BSAD06 in addition to the module address must be invoked. Since the "off-line" module was previously reconfigured, its module # is determined by the reconfiguration register contents. By way of illustration, assume in Step 2 it was necessary to move the offline module #2 to the vacated space of module #14, and further assume the fault in module #2 was such that only the lower board pair was defective. It will be the intent to place the off-line module #2 from off-line status to on-line module #14 with a swap in board positions. The following would occur:

STEP 4

a. BSYELO is true
b. BSAD06 is true
c. Memory Reference true, BSMREF
d. Read one word
e. Address = 1170A8

Communicating with the off-line module #2 is accomplished with BSYELO and BSAD06 signals and the module address value contained in the reconfiguration register. The module issues a MYACKR corresponding to a single word read operation and on the MYDCNN signal the reconfiguration information is clocked into its register. This information places the module in the interleaved mode at the physical space of the original module #14, swaps the daughter boards, and clears the off-line command bit. Eventually the module will reside at the top pair along with module #15. The bottom half of the memory pair will be functional.

Once the memory controller has been reconfigured, the information may be updated or changed by issuing subsequent Reconfiguration commands. However, returning to the original state under control of the module switches can only be accomplished by issuing a default command consisting of BSYELO, BSMREF and BSAD07. These fnctions are interpreted by all memory controllers clearing any Reconfiguration command and reverting control to the module's switches.

- 2 -	15
12	13
6	7
4	5
14	3
0	1

A.2.6 "Here I Am" Light

When this diagnostic code is received, the memory reads the location being addressed. During the second half bus cycle, the data is shipped to the register and the information controlling the "Here I Am" light (contained in current address field) is loaded into a flop. The "Here I Am" light powers-up in the off state and clears with bus master clear.

When the memory receives the "Here I Am" code (5 in address bits 19, 20 & 21) it interprets the remaining bits of the address as follows:

8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	V

16	17	18	19	10	21	22	23
0	0	X	1	0	1	X	X

where: V = Value of "Here I Am" light:
0 = Light Off
1 = Light On
X = Do not care.

A.2.7. Soft Error Rewrite Control

When this diagnostic code is received, the memory reads the location being addressed and sends data to the register during the second half bus cycle. Memory then modifies the operation of its soft error rewrite logic based upon the two bi in the address field. For proper operation of automatic controls, the manual override switches must be in their normal position.

When the memory receives the soft error rewrite control code (6 in address bits 19, 20, and 21), it interprets the remaining bits of the address field as follows:

8	9	10	11	12	13	14	15
0	0	0	0	0	0	S	B

16	17	18	19	10	21	22	23
0	0	X	1	1	0	X	X

where: B = Bypass Soft Error Rewrite:
- 0 = Soft Error Rewrite On
- 1 = Soft Error Rewrite Off

S = Soft Error Rewrite Cycle:
- 0 = Normal Cycle
- 1 = High-Speed T&V Cycle (5 sec).

A.3 MAIN MEMORY DIAGNOSTIC PROGRAMMING

The following describes the programming required to perform main memory diagnostics.

Subsection A.2 describes the diagnostic funcitonality from a hardware point of view. It describes the diagnostic functions available and how they are invoked. Refer to Table A-1 for a summary.

Table A-1 Hardware Command Summary

DIAGNOSTIC COMMAND	ADDRESS BITS																							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Read ID	M	M	M	M	M	M	X	X	X	X	X	X	X	X	X	X	X	X	M	0	0	0	0	X
Read Status	M	M	M	M	M	M	X	X	X	X	X	X	X	X	X	X	X	X	M	0	0	1	0	X
Set EDAC Mode	M	M	M	M	M	M	X	X	X	X	X	X	X	X	X	X	X	X	M	0	1	0	0	X
Reset EDAC Mode	M	M	M	M	M	M	X	X	X	X	X	X	X	X	X	X	X	X	M	0	1	1	0	X
Set "Here I Am" Indicator	M	M	M	M	M	M	X	X	0	0	0	0	0	0	0	V	0	0	M	1	0	1	0	X
Soft Error Rewrite	M	M	M	M	M	M	X	X	0	0	0	0	0	0	S	B	0	0	M	1	1	0	0	X

NOTES

1. Address bits refer to the address bits on the Megabus or Extended Megabus. Thus a main memory command is really a physical address.
2. M (bits 0 through 5 and bit 18) stands for Module select.
3. X = Don't care. It may be either 0 or 1.
4. Bits 19 through 21 contain the diagnostic command code.
5. Bits 8 through 17 contain the supplementary diagnostic command code.

Subsection 5.9.27 describes the new CPU instruction which has been defined to perform main memory diagnostics. What follows describes how B5 and the memory management unit (MMU) are used to generate the bit patterns (in SMMU mode) defined in Table A-1. Note that a base register contains a logical address (LA) which will be converted into a physical address to be sent to memory. Note also that the bit patterns defined in Table A-1 reflect physical addresses (PA).

To understand the LA-PA translation process, refer to Figure A-1.

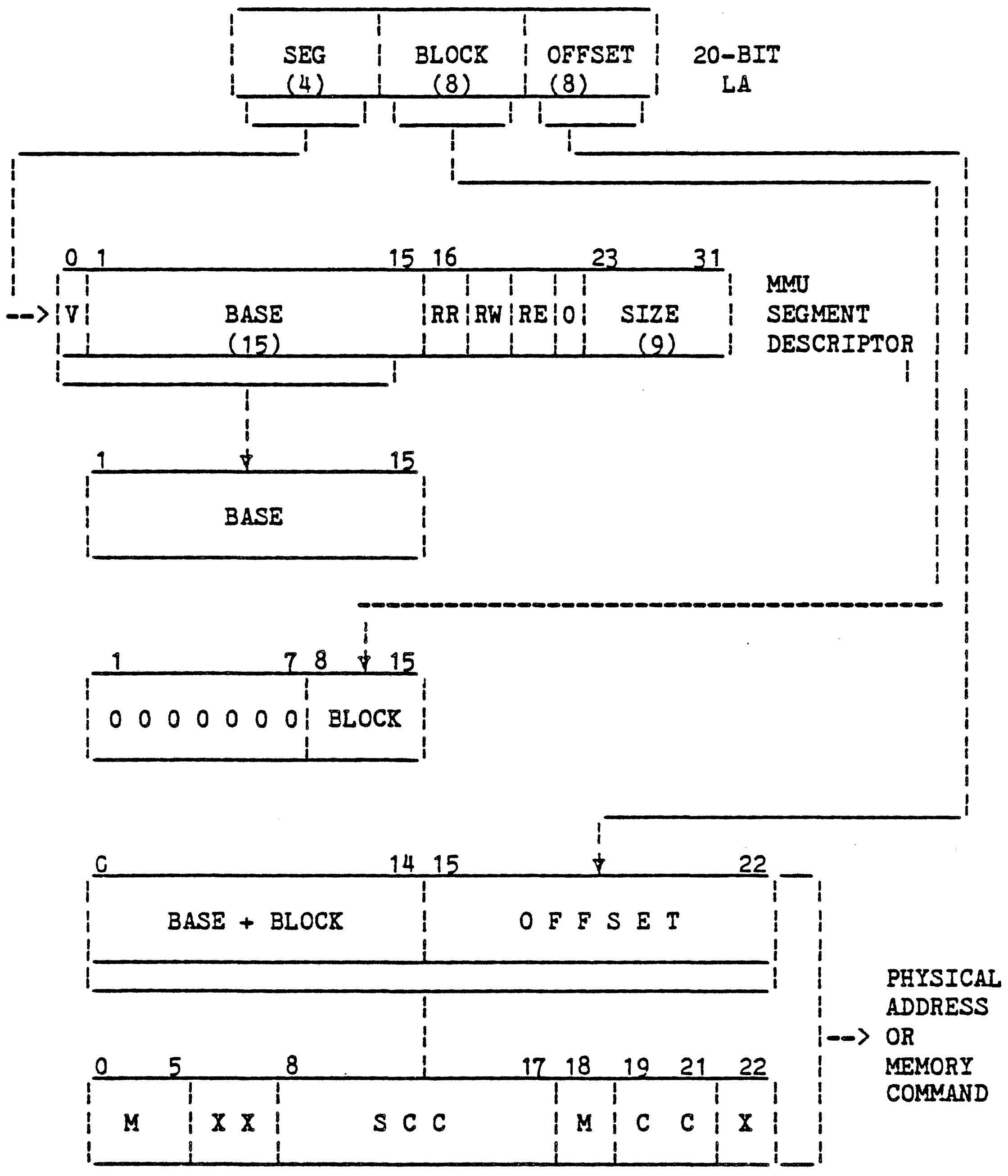
Basically the LAs consist of three fields: segment number, block number and offset. The segment field is used to reference a segement descriptor (SD) in the MMU. The base in the SD is then added to the block number field and the result is concatenated to the offset field to produce a 23-bit PA.

With this background, it is now possible to describe one way of generating a main memory command (PA).

- Let the LA in a base register have:
  - Segment = X
  - Block = 0
  - Offset = 001
- Set the desired memory command code from Table 5-19 in K6 (reference subsection 5.9.27)
- Set the base field of SD# X to the module address
- Activate SD# X
- Execute the generic main memory command.

Other ways of generating a main memory command can be defined by following the flow given in Figure A-1.

Note that the above description operates in the same manner for EMMU mode, with the exception that the LA segment number field (SEG) extends to 8 bits.



Example shows M5X Mode Operation only

Figure A-1 Logical to Physical Address Translation

This page is intentionally blank.

APPENDIX B MMU DIAGNOSTIC FUNCTIONALITY

B.1 SCOPE

This appendix describes the Memory Management Unit (MMU) diagnostic functionality supported by the M5XE, M6X and M6XE.

B.2 MMUD FUNCTIONS

R5 supplies a four-bit function code that specifies which diagnostic action is to be performed. On an M6X or M6XE, R5 is returned with the disaster syndrome. On an M5XE, R5 is not changed. Depending on the value of the sign bit and the contents of the three least significant bits of R5, the following functions can be invoked:

- o <u>Sign bit R5(0) = One</u> - Display Current Task Segment Table parameters: The contents of the CPU hidden registers, containing the most recent Task Segment Table Base (TSTB) and Task Segment Table Limit (TSTL), are loaded into R6 and R7 as shown below.

```
|          R6           |            R7          |
|0          7  8  9   15|0                     15|
|           |     |     |                        |
|   (TSTL)  |  0  | <-----------(TSTB)---------> |
|           |     |   23 bit Physical Address    |
|___________|_____|_____|________________________|
```

Note that the TSTB is the converted PA derived form the ASV logical address.

If the sign bit R5(0) = ONE, then the three least significant bits of R5 (Function Code bits - FC) are ignored.

o <u>Sign bit R5(0) = ZERO</u> - Depending upon the content of the three least significant bits of R5, [R5(13:15)], the following eight function codes can be invoked:

- <u>FC = 0</u> - Read Segment Descriptor (SD): The 32-bit SD, corresponding to the segment containing the virtual address specified in B5, is fetched from the MMU SD storage facility and delivered to R6 and R7.

  In an M6X and M6XE the following applies:

  The format of the 32-bit result is rotated eight bit positions from the standard SD format:

R6			R7		
0 7	8	9 15 0 7	8 13	14	15
8 LSB OF SZ	V	B A S E	ACCESS RIGHTS	RFU	SZ MBZ

  (Executing DCL R7, 8 will restore a more familiar format.)

  In an M5XE the following applies:

  The format of the 32-bit result is identical to that of the standard SD format:

	R6	R7	
0	1 15	0 6	7 15
V	BASE	ACCESS RIGHTS	SIZE

- <u>FC = 1</u> - Set Task Segment Table Parameters (for M6X and M6XE): The 8-bit TSTL supplied in B5(24:31) is transferred to the EMMU TSTL register. The pointer to the TST supplied in R6(9:15) and R7(0:15) is transferred to the EMMU TSTB register.

- <u>FC = 1</u> - Trap TV16 (for M5XE).

- <u>FC = 2</u> - Translate Address: The virtual address from B5 is converted via the appropriate segment descriptor, and the resulting physical address delivered (right justified) to R6, R7. The eight most significant bits of R6 receive the right-most byte of the segment descriptor (part of size field) in an M6X and M6XE and are set to zero in an M5XE.

- FC = 3 - Set Segment Descriptor: The SD, corresponding to the segment containing the logical address specified in B5, is loaded from the 32-bit content of R6, R7.

- FC = 4 - Read Interrupt Register: The content of the 16-bit external interrupt register (10-bit interrupting channel number, 6-bit level number) is placed in R6. R7 is cleared.

- FC = 5 - Set Interrupt Register (for M6X and M6XE): The 2-bit channel number and the 6-bit level number supplied in R7, bits 6, 7, 9-14, are transferred to the external interrupt register. Bit 8 defines CSS on line/off line as follows:

  - Bit 8 = 0 ---> CSS is placed off line
  - Bit 8 = 1 ---> CSS is placed on line.

- FC = 5 - Trap TV16 (for M5XE).

- FC = 6 - Read Mode (for M6X and M6XE): The 16-bit mode register in the MMU/Cache is interrogated and its content placed in both R6 and R7.

- FC = 6 - Read Mode (for M5XE): The 8-bit mode register is interrogated and its content placed in R7(0:7). R7(8:15) and R6 are set to zero.

- FC = 7 - Set Mode (for M6X and M6XE): A single bit of the MMU/Cache mode register is set or cleared as specified by R7(11:15). R7(0:10) and R6 are reserved for future use and should be Zeros:

  $00 \leq [R7] \leq 07$ - Clear selected bit 8-15

  $08 \leq [R7] \leq 0F$ - Set selected bit 8-15

  $10 \leq [R7] \leq 17$ - Clear selected bit 0-7

  $18 \leq [R7] \leq 1F$ - Set selected bit 0-7

- FC = 7 - Set Mode (for M5XE): A single bit of the MMU mode register is set or cleared as specified by R7(12) (which specifies a reset if set to 0 or specifies a set if set to 1) and R7(13:15) (which specifies the bit, one of eight, to be updated). .

B.3 MMUD FAULT CONDITIONS

a. Any MMUD executed with less than maximum privilege (Ring 0) causes a trap (TV13) regardless of function code, etc.

b. An MMUD executed on a Level 6 model which does not implement this instruction causes a trap (TV05).

c. Note that the MMUD instruction (particularly with FC = 0 or 2) does not test the validity nor size field of the referenced segment descriptor, or the legality of the logical address supplied from B5 (e.g., above 1 MW). Software which uses these function codes is responsible for prechecking the appropriate descriptor(s), etc.

B.4 MMUD MODES

The MMU/Cache mode register contains 16 (for M6X and M6XE) and 8 (for M5XE) mode control bits which can be individually set or cleared by firmware or by the MMUD instruction. All are cleared by any Master Clear signal (including power-up and the maintenance panel CLEAR button). Changing multiple bits in the mode register requires the multiple MMUD executions.

NOTE

Usage of the mode control bits requires full understanding of the hardware operation.

The names and functions of the 16 mode bits (for M6X and M6XE) are:

BIT #	NAME	FUNCTION (WHEN ON)
0 *	SSTATV	Enter EMMU mode.
1 *	SPRSNT	Set Task SD present bit in EMMU.
2	L2FHIT	Meaningful only if FRCHIT (bit 7) is set. If L2FHIT = 0, then force a hit on cache level 1. If L2FHIT = 1, then force a hit on cache level 2.
3 *	PRCBYP	Bypass cache update for procedure reads.
4 *	PRTYER	Force a parity error in the directory and SD.
5	HMRINH	Inhibit memory reads/writes on tick (e.g., to the memory locations; RTC current, WDT current and memory yellow error count).
6	BYPMRY	Inhibit all Megabus accesses. Results of misses and non-memory references are undefined.
7	FRCHIT	For Cache to act as if a hit occurred on Level #1 if physical address bit 11 is Zero, on Level #2 if physical address bit 11 is One. If request is a read, return data from indicated Cache level; if request is a write, "Update" indicated level.
8	INITLZ	Suppress normal alternation between the two Cache levels for "Replace" operations.
9	IHRGCK	Inhibit access rights checking for memory references initiated by the CSS.
10	INHARL	Inhibit address relocation.

* M6XE only

BIT #	NAME	FUNCTION (WHEN ON) (continued)
11	CAHBYP	Bypass Cache (Cache continues to "Update" on write hits, but does not "Replace" on read misses, nor supply data on read hits).
12	FRCMIS	Force Miss (Cache does not supply data, does "Replace" on all reads, does not "Update" on writes).
13	FNHMDE	Generate a UAR Trap (TV15) if any memory reference results in a cache miss.
14	MRWRAP	Memory writes do not "Update" Cache directly, but are "wrapped" via Megabus adapter to simulate writes originating in other subsystems.
15	WRDRTY	UAR does not block "Replace" in Cache directory.

Certain mode combinations deserve further comment:

a. When CAHBYP and MRWRAP are both on, CSS-initiated memory writes are queued in the Megabus adapter FIFO buffer, to be selectively "Updated" into the Cache data store (depending on hit/miss) when CAHBYP is cleared. It is assumed software assures the absence of other Megabus activity during such a test.

b. When CAHBYP and FRCMIS are both on, the Cache performs neither "Updates" nor "Replaces"; i.e., the Cache contents are unchanged by either reads or writes. Subsequent to such a test, software is responsible for restoring the correspondence of Cache and main memory contents.

c. When BYPMRY is One, FRCHIT should also be on to avoid the undefined results of a miss. Likewise, I/O operations should not be attempted during such a test (they won't work).

The names and functions of the 8 mode bits (for M5XE) are:

BIT #	NAME	FUNCTION (WHEN ON)
0	EMMU	Enter EMMU mode.
1	SPRSNT	Set Task SD present bit in EMMU.
2	ADDINH	Inhibit address translation.
3	RFU	
4	NOCHECK	Disable ring checking.
5 - 7	RFU	

<u>B.5 M6X AND M6XE CACHE ACTIONS AND TERMINOLOGY (with all mode bits cleared)</u>

The Cache acts on every memory read or memory write initiated by the CSS. The Cache also monitors all memory writes originating from other subsystems on the megabus (e.g., I/O controllers, other CSS).

For each of the above cases, the Cache checks its "directory" to determine whether the referenced memory location (physical address) is represented in either level of the Cache. If the location is so represented, the reference is termed a "hit", otherwise it is a "miss".

When a CSS-initiated read results in a hit, the Cache supplies the desired data to the requesting processor, with no Megabus involvement. (Exception: if the read request includes a command to "Lock" the memory, as in Read-Modify-Write instructions, the Lock command is transmitted to the main memory module involved, and the Cache waits for the command to be acknowledged before transmitting the data to the processor.)

When a CSS-initiated read results in a miss, the Cache relays the request via the Megabus adapter to main memory and waits for the data to be returned. It then passes the data to the processor and also "Replaces" a suitable location in the Cache directory and data store to represent this most recent reference. Successive "Replaces" are directed alternately to Level 1 and Level 2.

When a memory write is initiated by the CSS, the Cache relays the request via the Megabus adapter to main memory.

When any memory write (initiated by CSS or Megabus subsystem) which results in a hit is acknowledged by the addressed memory module, the Cache "Updates" the appropriate location in its data store to reflect the new content.

<u>B.6 M6X AND M6XE SYNDROME</u>

Whenever a "Disaster" stimulus (master clear, unavailable resource (UAR), protection violation (PROV), uncorrectable memory error (RED) or bus parity error (PAR)) is detected by CPU hardware, a syndrome pattern describing the nature of the stimulus is stored in a hardware register. The syndrome is analyzed by the CPU firmware to determine the appropriate action to be taken (e.g., trap). The syndrome remains available for software interrogation by the MMUD instruction (for example) until the next disaster occurs. This syndrome pattern is loaded into R5 by the CPU firmware during the execution of the MMUD instruction.

The syndrome pattern (for CPU release 2.1) is:

BIT #	MEANING IF TRUE
0	UAR on procedural read
1	PROV on procedural read
2	RED on procedural read
3	PAR on procedural read
4	UAR on non-procedural read or write
5	PROV on non-procedural read or write
6	RED on non-procedural read
7	PAR on non-procedural read
8	WRAP (overflow in address calculation)
9	Master Clear (ignore other syndrome bits)
10,11	Effective ring number used in reference
12	IO (reference was not to memory)
13	Error was associated with left word (i.e., was not just in right half of doubleword reference)
14,15	Amount by which non-procedural address, Y, was incremented subsequent to reference and before disaster was detected.

APPENDIX C SEARCH INSTRUCTION EXAMPLES

The Search instruction can be classified into four distinct cases:

- o Search String Single (i.e., search a string to determine whether it contains the single search argument given in the search list);
- o Search String Multiple (i.e., search a string to determine whether it contains any one of the multiple search arguments given in the search list);
- A o Search Array Single (i.e., search an array to determine whether it contains the single search argument given in the search list); and
- o Search Array Multiple (i.e., search an array to determine whether it contains any one of the multiple search arguments given in the search list).

Following are examples for each case.

C.1 SEARCH STRING SINGLE EXAMPLE

[DD1] - The search list which contains one search argument (SA). The SA can consist of one or more characters.

[DD2] - If a match is found, then store in [DD2] the search argument number (= 0) and the displacement.

[DD3] - The string.

B Given the string:

```
0 1 2 3 4 5 6 7 8 9 A B C
a b c d e f g h i j d e k
```

```
fBr SA = d      --> Post = and set [DD2] = 0, 3

for SA = fg     --> Post = and set [DD2[ = 0, 5

for SA = fh     --> Post ≠ and set [DD2] = unchanged

for SA = dek    --> Post = and set [DD2] = 0, A

for SA = lm     --> Post ≠ and set [DD2] = unchanged

for SA = a      --> Post = and set [DD2] = 0, 0.
```

C.2 SEARCH STRING MULTIPLE EXAMPLE

[DD1] - The search list which contains multiple search arguments. The length of an SA* can be one or more characters but all SAs must have the same length. This case requires the use of R4 which contains the SAL* and the SLL*.

[DD2] - If a match is found, then store in [DD2] the search argument number and the displacement.

[DD3] - The string.

Given the string:

```
0 1 2 3 4 5 6 7 8 9 A B C
a b c d e f g h i j d e k
```

```
for SAL* = 1, SLL* = 3 --> f, e, j  --> Post = and set [DD2] = 1, 4

for SAL = 1, SLL = 2 --> r, s       --> Post ≠ and set [DD2] = unchanged

for SAL = 2, SLL = 6 --> de, hi, er --> Post = and set [DD2] = 0, 3

for SAL = 3, SLL = 6 --> cdf, hij   --> Post = and set [DD2] = 1, 7

for SAL = 2, SLL = 4 --> cb, ka     --> Post ≠ and set [DD2] = unchanged
```

*Defined in subsection 7.3.2.4 (NOTE).

C.3 SEARCH ARRAY SINGLE EXAMPLE

[DD1] - The search list which contains one search argument. The SA* can consist of one or more characters.

[DD2] - If a match is found, then store in [DD2] a Zero for search argument number and the displacement.

[DD3] - The array. This case requires the use of R6 which contains the OEL* and the OL*.

Given the array:

```
00  a b d f

04  a c b e

08  c a d e

0C  d e f g

10  m j o p

14  e a c b
```

where R6 specifies OEL* = 4 and OL* = 24,

```
for SA* = ca      --> Post = and set [DD2] = 0, 08

for SA = a        --> Post = and set [DD2] = 0, 00

for SA = mjo      --> Post = and set [DD2] = 0, 10

for SA = mjpo     --> Post ≠ and set [DD2] = unchanged

for SA = acbec    --> Post = and set [DD2] = 0, 04

for SA = eacba    --> Post ≠ and set [DD2] = unchanged

for SA = bac      --> Post ≠ and set [DD2] = unchanged

for SA = cade     --> Post = and set [DD2] = 0, 08
```

C.4 SEARCH ARRAY MULTIPLE EXAMPLE

[DD1] - The search list which contains multiple search arguments. The length of an SA can be one or more characters but all SAs must have the same length. This case requires the use of R4 which contains the SAL* and the SLL*.

[DD2] - If a match is found, then store in [DD2] the search argument number and the displacement.

*Defined in subsection 7.3.2.4 (NOTE).

[DD3] - The array. This case requires the use of R6 which contains the OEL* and the OL*.

Given the array:

```
00  a b d f

04  a c b e

08  c a d e

0C  d e f g

10  m j o p

14  e a c b
```

where R6 specifies OEL* = 4 and OL* = 24,

```
for SAL* = 3, SLL* = 6 --> acb, acd         --> Post = and set [DD2] = 0, 04
for SAL = 1, SLL = 3 --> c, a, d            --> Post = and set [DD2] = 1, 00
for SAL = 4, SLL = 8 --> defg, abcd         --> Post = and set [DD2] = 0, 0C
for SAL = 2, SLL = 6 --> ad, ea, mj         --> Post = and set [DD2] = 2, 10
for SAL = 3, SLL = 9 --> aab, abb, eac      --> Post = and set [DD2] = 2, 14
for SAL = 5, SLL = 10 --> abdfb, mjope      --> Post = and set [DD2] = 1, 10
```

*Defined in subsection 7.3.2.4 (NOTE).

APPENDIX D VERIFY INSTRUCTION EXAMPLES

The Verify instruction can be classified into four distinct cases:

- o Verify String Single (i.e., verify a string to determine whether its characters are all equal to the single verify argument given in the verify list;
- o Verify String multiple (i.e., verify a string to determine whether its characters are each equal to one of the multiple verify arguments given in the verify list;
- o Verify Array Single (i.e., verify an array to determine whether its elements are all equal to the single verify argument given in the verify list); and
- o Verify Array multiple (i.e., verify an array to determine whether its elements are each equal to one of the multiple verify arguments given in the verify list).

Following are examples for each case.

D.1 VERIFY STRING SINGLE EXAMPLE

[DD1] - The verify list which contains one verify argument.

[DD2] - If a noncompare is found, then store in [DD2] the displacement.

[DD3] - The string.

Given the string:

```
0 1 2 3 4 5 6

a a a b a a c
```

for VA = a --> Post ≠ and set [DD2] = 3

for VA = b --> Post ≠ and set [DD2] = 0

Given the string:

```
0 1 2 3 4 5 6

b b b b b b b
```

for VA = b --> Post = and set [DD2] = unchanged.

D.2 VERIFY STRING MULTIPLE EXAMPLE

[DD1] - The verify list which contains multiple verify arguments (VAs). The length of a VA* should be one character. This case requires the use of R4 which contains VAL* and VLL*.

[DD2] - If a noncompare is found, then store in [DD2] the displacement.

[DD3] - The string.

Given the string:

```
0 1 2 3 4 5 6 7 8 9

a b c b b a d b c c
```

for VA = a, b --> Post = and set [DD2] = 2

for VA = a, b, c, e --> Post ≠ and set [DD2] = 6

for VA = a, b, c, d --> Post = and set [DD2] = unchanged.

D.3 VERIFY ARRAY SINGLE EXAMPLE

[DD1] - The verify list which contains one verify argument. The VA can consist of one or more characters.

[DD2] - If a noncompare is found, then store in [DD2] the displacement.

[DD3] - The array. This case requires the use of R6 which contains the OEL and the OL.

*Defined in subsection 7.3.2.5 (Special Conditions)

Given the array:

```
0 a b a

3 a b c

6 a b d

9 a c b
```

where R6 specifies OEL = 3 and OL = 12,

for VA = ab --> Post ≠ and set [DD2] = 9

for VA = abc --> Post ≠ and set [DD2] = 0

for VA = a --> Post = and set [DD2] = unchanged
for VA = aba --> Post ≠ and set [DD2] = 3.

D.4 VERIFY ARRAY MULTIPLE EXAMPLE

[DD1] - The verify list which contains multiple verify arguments. The length of a VA* can be one or more characters but all VAs must have the same length. This case requires the use of R4 which contains the VAL* and the VLL*.

[DD2] - If a noncompare is found, then store in [DD2] the displacement.

[DD3] - The array. This case requires the use of R6 which contains the OEL* and the OL*.

Given the array:

```
0  a b c d

4  a c d b

8  b c a d

C  a c b d
```

where R6 specifies OEL = 4 and OL = 16,

for VA = ab, ac --> Post ≠ and set [DD2] = 8

for VA = ab, ac, bc --> Post = and set [DD2[= unchanged

for VA = abc, acd, acb --> Post ≠ and set [DD2[= 8

for VA = abcd, acbd --> Post ≠ and set [DD2] = 4

*Defined in subsection 7.3.2.5 (Special Conditions).

This page is intentionally blank.

APPENDIX E GENERAL INSTRUCTION SUMMARY

This Appendix contains summaries of the formats, instructions, and numerical representations of the General instruction set.

It will be supplied at a later date.

This page is intentionally blank.

APPENDIX F EXTENDED INTEGER INSTRUCTION (EII) SUMMARY

This Appendix contains summaries of the formats, instructions, and numerical representations of the Extended Integer Instruction (EII) set.

It will be supplied at a later date.

This page is intentionally blank.

APPENDIX G COMMERCIAL INSTRUCTION SUMMARY

This Appendix contains summaries of the formats, instructions, and numerical representations of the Commercial Instruction set.

It will be supplied at a later date.

This page is intentionally blank.

APPENDIX H SCIENTIFIC INSTRUCTION SUMMARY

This Appendix contains summaries of the formats, instructions, and numerical representations of the Scientific Instruction set.

It will be supplied at a later date.

This page is intentionally blank.

REVIEWED BY: [illegible] DATE: [illegible]
J.R. HANNIGAN
DIRECTOR, HRX PROJECTS

REVIEWED BY: Virendra S. Negi DATE: 10/14/85
V.S. NEGI
DIRECTOR, CENTRAL SYSTEMS DEVELOPMENT

REVIEWED BY: [illegible] DATE: 10-24-85
G.J. BEKAMPIS
DIRECTOR, HARDWARE AND PERIPHERALS DEVELOPMENT

REVIEWED BY: [illegible] DATE: 10/25/85
T. HATCH
DIRECTOR, SYSTEMS SOFTWARE DEVELOPMENT

REVIEWED BY: [illegible] DATE: 10/28/85
J. BEHYMER
DIRECTOR, APPLIED SOFTWARE DEVELOPMENT

APPROVED BY: Dana Moore DATE: 11/6/85
D.W. MOORE
DIRECTOR, DPS-6 HARDWARE, SOFTWARE AND PERIPHERALS DEVELOPMENT

APPROVED BY: W C Gould DATE: 11/6/85
W.C. GOULD
VICE PRESIDENT, SCP ENGINEERING