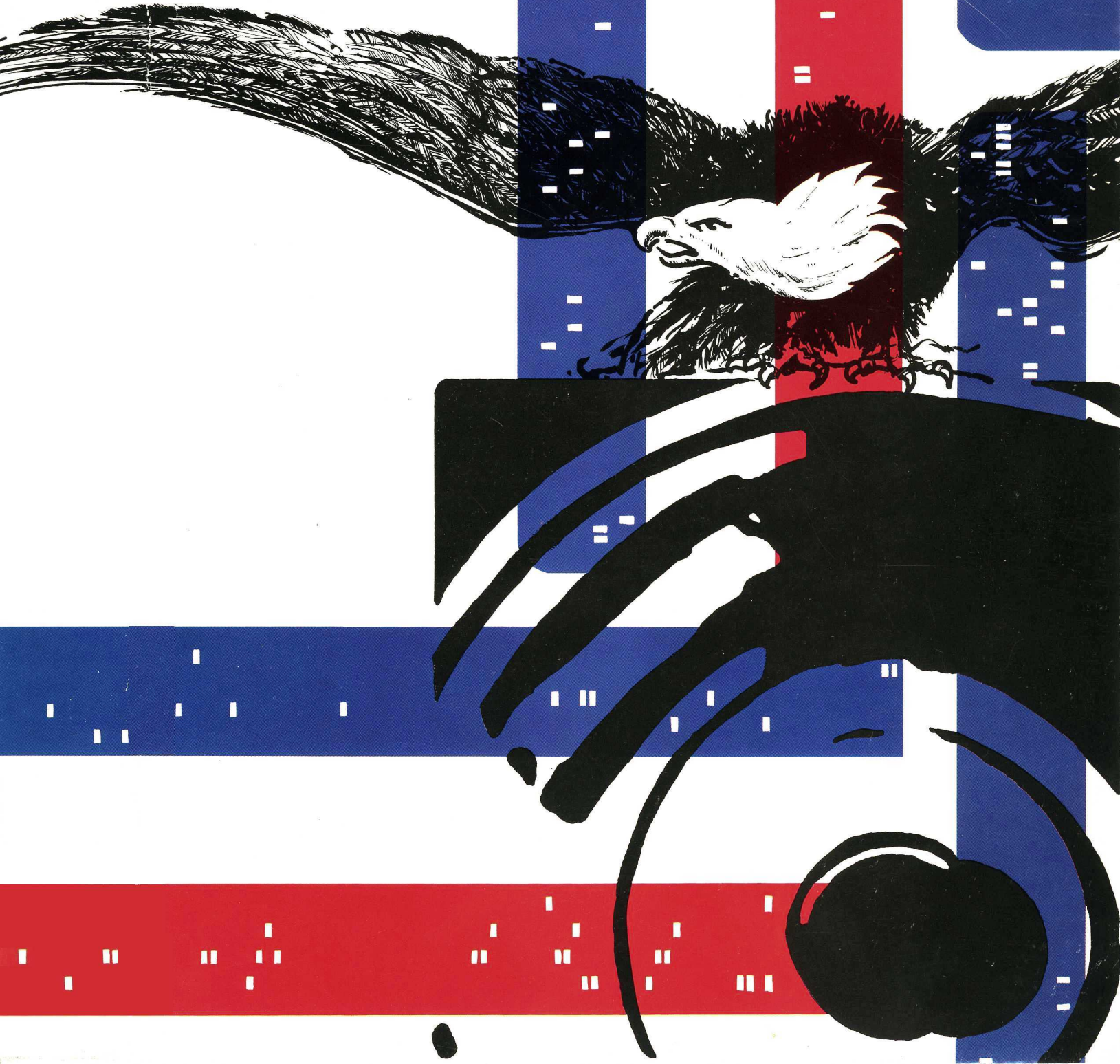


Honeywell

HONEYWELL INFORMATION SYSTEMS

Multics Processor Manual



SUBJECT:

Description and Use of the Multics Processor.

SOFTWARE SUPPORTED:

All Multics Software Releases

DATE:

October, 1975

ORDER NUMBER:

AL39, Rev. 0

PREFACE

This document describes the Processor used in the Multics system. It is assumed that the reader is familiar with the overall modular organization of the Multics system and with the philosophy of asynchronous operation. In addition, this manual presents a thorough discussion of virtual memory addressing concepts including segmentation and paging.

The manual is intended for use by system programmers responsible for writing software to interface with the special virtual memory hardware and with the fault and interrupt portions of the hardware. It should also prove valuable to programmers who must use machine instructions (particularly language translator implementors) and to those persons responsible for analyzing crash conditions in System Dumps.

© 1974 Honeywell Information Systems, Inc.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

CONTENTS

		Page
Section I	Introduction To Processor	1-1
	Features of the Multics Processor	1-1
	Segmentation and Paging	1-1
	Address Modification and Address Appending	1-2
	Faults and Interrupts	1-2
	Summary of Processor Features	1-2
	Processor Modes of Operation	1-3
	Instruction Modes	1-3
	Normal Mode	1-3
	Privileged Mode	1-4
	Addressing Modes	1-4
	Absolute Mode	1-4
	Append Mode	1-4
	Bar Mode	1-4
	Processor Unit Functions	1-5
	Appending Unit	1-5
	Associative Memory Assemblies	1-5
	Control Unit	1-6
	Operation Unit	1-6
	Decimal Unit	1-6
Section II	Machine Instructions	2-1
	Instruction Repertoire	2-1
	Basic Operations	2-1
	Extended Instruction Set (EIS) Operations	2-1
	EIS Single-Word Operations	2-1
	EIS Multi-Word Operations	2-2
	Format of Instruction Description	2-2
	Definitions of Notation and Symbols	2-4
	Main Store Addresses	2-4
	Index Values	2-4
	Abbreviations and Symbols	2-4
	Register Positions and Contents	2-5
	Other Symbols	2-6
	Arrangement of Instructions	2-6
	Common Attributes of Instructions	2-6
	Illegal Modification	2-6
	Parity Indicator	2-6
	Instruction Word Formats	2-7
	Basic and EIS Single-Word Instructions	2-7
	Indirect Words	2-8
	EIS Multi-Word Instructions	2-9
	EIS Modification Fields (MF)	2-9
	EIS Operand Descriptors and Indirect Pointers	2-11
	Operand Descriptor Indirect Pointer Format	2-11
	Alphanumeric Operand Descriptor Format	2-12
	Numeric Operand Descriptor Format	2-14
	Bit String Operand Descriptor Format	2-15
	Fixed Point Arithmetic Instructions	2-16
	Data Movement Load Instructions	2-16
	Data Movement Store Instructions	2-25

CONTENTS (Cont)

	Page
Data Movement Shift Instructions	2-34
Addition Instructions	2-40
Subtraction Instructions	2-48
Multiplication Instructions	2-55
Division Instructions	2-57
Negate Instructions	2-60
Comparison Instructions	2-61
Miscellaneous Instructions	2-66
Boolean Operation Instructions	2-67
AND Instructions	2-67
OR Instructions	2-70
Exclusive OR Instructions	2-73
Comparative AND Instructions	2-76
Comparative NOT Instructions	2-78
Floating Point Arithmetic Instructions	2-80
Data Movement Load Instructions	2-80
Data Movement Store Instructions	2-81
Addition Instructions	2-84
Subtraction Instructions	2-87
Multiplication Instructions	2-90
Division Instructions	2-93
Negate Instructions	2-97
Normalize Instructions	2-98
Round Instructions	2-99
Compare Instructions	2-101
Miscellaneous Instructions	2-103
Transfer Instructions	2-105
Pointer Register Instructions	2-117
Data Movement Load Instructions	2-117
Data Movement Store Instructions	2-121
Address Arithmetic Instructions	2-124
Miscellaneous Instructions	2-125
Miscellaneous Instructions	2-126
Calendar Clock Instruction	2-126
Derail Instruction	2-127
Execute Instructions	2-128
Master Mode Entry Instructions	2-130
No Operation Instructions	2-133
Repeat Instructions	2-135
Ring Alarm Register Instruction	2-144
Store Base Address Register Instruction	2-145
Translation Instructions	2-146
Privileged Instructions	2-148
Register Load Instructions	2-148
Register Store Instructions	2-155
Clear Associative Memory Instructions	2-162
Configuration and Status Instructions	2-164
System Control Instructions	2-167
Miscellaneous Instructions	2-171
Extended Instruction Set (EIS)	2-172
Address Register Load Instructions	2-172
Address Register Store Instructions	2-175
Address Register Special Arithmetic Instructions	2-178
Alphanumeric Compare Instructions	2-186
Alphanumeric Move Instructions	2-197
Numeric Compare Instruction	2-205
Numeric Move Instructions	2-208
Bit String Combine Instructions	2-213
Bit String Compare Instructions	2-217

CONTENTS (Cont)

	Page
Bit String Set Indicators Instructions	2-219
Data Conversion Instructions	2-221
Decimal Addition Instructions	2-225
Decimal Subtraction Instructions	2-231
Decimal Multiplication Instructions	2-234
Decimal Division Instructions	2-237
Micro Operations for Edit Instructions	2-240
Micro Operation Sequence	2-240
Edit Insertion Table	2-241
Edit Flags	2-241
Terminating Micro Operations	2-242
MVNE and MVE Differences	2-242
Numeric Edit	2-242
Alphanumeric Edit	2-242
Micro Operators	2-243
Micro Operation Code Assignment Map	2-250
Section III	
Data Representation	3-1
Information Organization	3-1
Position Numbering	3-1
Number System	3-1
Information Formats	3-2
Data Parity	3-4
Representation of Data	3-5
Numeric Data	3-5
Fixed Point Binary Data	3-5
Fixed Point Binary Integers	3-5
Fixed Point Binary Fractions	3-6
Floating Point Binary Data	3-8
Overlength Registers	3-9
Normalized Numbers	3-9
Decimal Data	3-11
Decimal Data Values	3-13
Decimal Zero	3-14
Alphanumeric Data	3-14
Character String Data	3-14
Bit String Data	3-15
Section IV	
Program Accessible Registers	4-1
Accumulator Register (A)	4-2
Quotient Register (Q)	4-2
Accumulator-Quotient Register (AQ)	4-3
Exponent Register (E)	4-4
Exponent-Accumulator-Quotient Register (EAQ)	4-4
Index Registers (Xn)	4-5
Indicator Register (IR)	4-5
Base Address Register (BAR)	4-9
Timer Register (TR)	4-9
Ring Alarm Register (RALR)	4-10
Pointer Registers (PRn)	4-11
Procedure Pointer Register (PPR)	4-13
Temporary Pointer Register (TPR)	4-15
Descriptor Segment Base Register (DSBR,DBR)	4-16
Segment Descriptor Word Associative Memory (SDWAM)	4-18
Page Table Word Associative Memory (PTWAM)	4-21
Fault Register	4-23
Mode Register (MR)	4-25

CONTENTS (Cont)

	Page
Cache Mode Register (CMR)	4-28
Control Unit (CU) History Registers	4-29
Operations Unit (OU) History Registers	4-32
Decimal Unit (DU) History Registers	4-34
Appending Unit (AU) History Registers	4-37
Configuration Switch Data	4-39
Control Unit Data	4-40
Decimal Unit Data	4-45
Section V	
Addressing -- Segmentation and Paging.	5-1
Addressing Modes.	5-1
Absolute Mode.	5-1
Append Mode.	5-2
Segmentation.	5-2
Paging.	5-3
Changing Addressing Modes	5-6
Address Appending	5-7
Address Appending Sequences.	5-7
Appending Unit Data Word Formats.	5-10
Segment Descriptor Word (SDW) Format	5-10
Page Table Word (PTW) Format	5-11
Section VI	
Effective Address Formation.	6-1
Definition of Effective Address	6-1
Types of Effective Address Formation.	6-1
Effective Address Formation Description	6-2
Effective Address Formation Involving Offset Only	6-2
The Address Modifier (TAG) Field	6-2
General Types of Offset Modification	6-2
Effective Address Formation Flowcharts	6-4
Register (R) Modification.	6-4
Register Then Indirect (RI) Modification	6-6
Indirect Then Register (IR) Modification	6-8
Indirect Then Tally (IT) Modification.	6-9
Effective Address Formation Involving Both	
Segment Number and Offset.	6-15
The Use of Bit 29 of the Instruction Word.	6-16
Special Modifiers.	6-16
Indirect to Pointer (itp) Modification.	6-17
Indirect to Segment (its) Modification.	6-18
Effective Segment Number Generation.	6-19
Effective Address Formation for Extended	
Instruction Set.	6-21
Character- and Bit-String Addressing	6-23
Character- and Bit-String Address Arithmetic	
Algorithms.	6-23
9-Bit Character String Address Arithmetic	6-24
6-Bit Character String Address Arithmetic	6-24
4-Bit Character String Address Arithmetic	6-24
Bit String Address Arithmetic	6-24
Section VII	
Faults and Interrupts.	7-1
Fault Cycle Sequence.	7-1
Fault Priority.	7-2
Fault Recognition	7-3
Fault Descriptions.	7-4

CONTENTS (Cont)

	Page
Group 1 Faults	7-4
Group 2 Faults	7-4
Group 3 Faults	7-5
Group 4 Faults	7-5
Group 5 Faults	7-6
Group 6 Faults	7-7
Group 7 Faults	7-7
Program Interrupts and External Faults.	7-8
Execute Interrupt Sampling	7-8
Execute Interrupt Cycle Sequence	7-9
Section VIII Hardware Ring Implementation	8-1
Ring Protection Philosophy.	8-1
Ring Protection in Multics.	8-1
Ring Protection in the Multics Processor.	8-2
Appending Unit Operation with Ring Mechanism.	8-4
Section IX Cache Store Operation.	9-1
Philosophy of Cache Store	9-1
Cache Store Organization.	9-1
Cache Store/Main Store Mapping	9-2
Cache Store Addressing	9-4
Cache Store Control	9-4
Enabling and Disabling Cache Store	9-4
Cache Store Control in Segment Descriptor Words.	9-5
Loading the Cache Store.	9-6
General Clear	9-6
Selective Clear	9-7
Dumping the Cache Store.	9-7

CONTENTS (Cont)

	Page
Appendix A	Operation Code Map A-1
Appendix B	Alphabetic Operation Code List B-1
Appendix C	Address Modifiers. C-1

ILLUSTRATIONS

Figure 2-1	Basic and EIS Single-Word Instruction Format	2-7
Figure 2-2	Indirect Word Format	2-8
Figure 2-3	EIS Multi-Word Instruction Format.	2-9
Figure 2-4	EIS Modification Field (MF) Format	2-10
Figure 2-5	Operand Descriptor Indirect Pointer Format	2-12
Figure 2-6	Alphanumeric Operand Descriptor Format	2-12
Figure 2-7	Numeric Operand Descriptor Format.	2-14
Figure 2-8	Bit String Operand Descriptor Format	2-15
Figure 2-9	Repeat Double (RPD) Instruction Word Format.	2-135
Figure 2-10	Repeat Link (RPL) Instruction Word Format.	2-138
Figure 2-11	Repeat (RPT) Instruction Word Format	2-141
Figure 2-12	EIS Address Register Special Arithmetic Instruction Format.	2-178
Figure 2-13	Compare Alphanumeric Strings (CMPC) EIS Multi-Word Instruction Format.	2-186
Figure 2-14	Scan Characters Double (SCD) EIS Multi-Word Instruction Format.	2-188
Figure 2-15	Scan with Mask (SCM) EIS Multi-Word Instruction Format .	2-191
Figure 2-16	Test Character and Translate (TCT) EIS Multi-Word Instruction Format.	2-194
Figure 2-17	Move Alphanumeric Left to Right (MLR) EIS Multi-Word Instruction Format.	2-197
Figure 2-18	Move Alphanumeric Edited (MVE) EIS Multi-Word Instruction Format.	2-200
Figure 2-19	Move Alphanumeric with Translation (MVT) EIS Multi-Word Instruction Format.	2-202
Figure 2-20	Compare Numeric (CMPN) EIS Multi-Word Instruction Format	2-205
Figure 2-21	Move Numeric (MVN) EIS Multi-Word Instruction Format . .	2-208
Figure 2-22	Move Numeric Edited (MVNE) EIS Multi-Word Instruction Format.	2-211
Figure 2-23	Combine Bit Strings Left (CSL) EIS Multi-Word Instruction Format.	2-213
Figure 2-24	Compare Bit Strings (CMPB) EIS Multi-Word Instruction Format.	2-217
Figure 2-25	Binary to Decimal Convert (BTD) EIS Multi-Word Instruction Format.	2-221
Figure 2-26	Decimal to Binary Convert (DTB) EIS Multi-Word Instruction Format.	2-223
Figure 2-27	Add Using 2 Decimal Operands (AD2D) EIS Multi-Word Instruction Format.	2-225
Figure 2-28	Add Using 3 Decimal Operands (AD3D) EIS Multi-Word Instruction Format.	2-228
Figure 2-29	Micro Operation (MOP) Character Format	2-240
Figure 3-1	Unstructured Machine Word Format	3-2
Figure 3-2	Unstructured Word Pair Format.	3-3
Figure 3-3	Unstructured 4-bit Character Format.	3-3
Figure 3-4	Unstructured 6-bit Character Format.	3-3

CONTENTS (Cont)

		Page
Figure 3-5	Unstructured 9-bit Character Format	3-4
Figure 3-6	Unstructured 18-bit Half Word Format	3-4
Figure 3-7	Upper 18-bit Half Word Floating Point Binary Operand Format	3-8
Figure 3-8	Lower 18-bit Half Word Floating Point Binary Operand Format	3-8
Figure 3-9	Single Precision Floating Point Binary Operand Format . .	3-9
Figure 3-10	Double Precision Floating Point Binary Operand Format . .	3-9
Figure 4-1	Accumulator Register (A) Format	4-2
Figure 4-2	Quotient Register (Q) Format	4-2
Figure 4-3	Accumulator-Quotient Register (AQ) Format	4-3
Figure 4-4	Exponent Register (E) Format	4-4
Figure 4-5	Exponent-Accumulator-Quotient Register (EAQ) Format . . .	4-4
Figure 4-6	Index Register (Xn) Format	4-5
Figure 4-7	Indicator Register (IR) Format	4-5
Figure 4-8	Base Address Register (BAR) Format	4-9
Figure 4-9	Timer Register (TR) Format	4-9
Figure 4-10	Ring Alarm Register (RALR) Format	4-10
Figure 4-11	Pointer Register (PRn) Format	4-11
Figure 4-12	Procedure Pointer Register (PPR) Format	4-13
Figure 4-13	Temporary Pointer Register (TPR) Format	4-15
Figure 4-14	Descriptor Segment Base Register (DSBR,DBR) Format . . .	4-16
Figure 4-15	Segment Descriptor Word Associative Memory (SDWAM) Format	4-18
Figure 4-16	Page Table Word Associative Memory (PTWAM) Format	4-21
Figure 4-17	Fault Register Format	4-23
Figure 4-18	Mode Register (MR) Format	4-25
Figure 4-19	Cache Mode Register (CMR) Format	4-28
Figure 4-20	Control Unit (CU) History Register Format	4-29
Figure 4-21	Operations Unit (OU) History Register Format	4-32
Figure 4-22	Appending Unit (AU) History Register Format	4-37
Figure 4-23	Configuration Switch Data Formats	4-39
Figure 4-24	Control Unit Data Format	4-41
Figure 4-25	Decimal Unit Data Format	4-46
Figure 5-1	Final Address Generation for an Unpaged Segment	5-3
Figure 5-2	Examples of Page Number Formation	5-4
Figure 5-3	Final Address Generation for an Paged Segment	5-6
Figure 5-4	Appending Unit Operation Flowchart	5-9
Figure 5-5	Segment Descriptor Word (SDW) Format	5-10
Figure 5-6	Page Table Word (PTW) Format	5-11
Figure 6-1	Address Modifier (TAG) Field Format	6-2
Figure 6-2	Common Effective Address Formation Flowchart	6-4
Figure 6-3	Register Modification Flowchart	6-6
Figure 6-4	Register Then Indirect Modification Flowchart	6-7
Figure 6-5	Indirect Then Register Modification Flowchart	6-9
Figure 6-6	Indirect Then Tally Modification Flowchart	6-15
Figure 6-7	Format of Instruction Word ADDRESS When Bit 29 = 1	6-16
Figure 6-8	IIP Pointer Pair Format	6-18
Figure 6-9	ITS Pointer Pair Format	6-19
Figure 6-10	Effective Segment Number Generation Flowchart	6-20
Figure 6-11	EIS Effective Address Formation Flowchart	6-22
Figure 8-1	Complete Appending Unit Operation Flowchart	8-4
Figure 9-1	Cache Store/Main Store Mapping	9-3

CONTENTS (Cont)

Page

TABLES

Table 2-1	R-type Modifiers for REG Fields.	2-10
Table 2-2	Alphanumeric Character Number (CN) Codes	2-13
Table 2-3	Alphanumeric Data Type (TA) Codes.	2-13
Table 2-4	Sign and Decimal Type (S) Codes.	2-14
Table 2-5	Relation Between Data Bits and Indicators.	2-21
Table 2-6	Control Relations for Store Character Instructions (Nine Bit).	2-28
Table 2-7	Control Relations for Store Character Instructions (Six Bit)	2-30
Table 2-8	Default Edit Insertion Table Characters.	2-241
Table 2-9	Micro Operation Code Assignment Map.	2-251
Table 3-1	Fixed Point Binary Integer Values.	3-6
Table 3-2	Fixed Point Binary Fraction Values	3-7
Table 3-3	Floating Point Binary Operand Values	3-10
Table 3-4	Decimal Sign Character Interpretation.	3-12
Table 3-5	Decimal Data Values.	3-13
Table 3-6	Character String Data Length Limits.	3-14
Table 4-1	Processor Registers.	4-1
Table 4-2	System Controller Illegal Action Codes	4-25
Table 5-1	Appending Unit Cycle Definitions	5-8
Table 6-1	General Offset Modification Types	6-3
Table 6-2	Register Modification Decode	6-5
Table 6-3	Variations of Indirect Then Tally Modification	6-10
Table 6-4	Special Append Mode Address Modifiers.	6-17
Table 7-1	List of Faults	7-3

SECTION I

INTRODUCTION TO PROCESSOR

The Processor described in this reference manual is a hardware module designed for use with the MULTIplexed Information and Computing Service (Multics). The many distinctive features and functions of Multics are enhanced by the powerful hardware features of the Processor. The addressing features, in particular, are designed to permit the Multics software to compute relative and absolute addresses, locate data and programs in different devices, and retrieve such data and program as necessary.

MULTICS PROCESSOR FEATURES

The Multics Processor contains the following general features:

1. Storage protection to place access restrictions on specified segments.
2. Capability to interrupt a process in execution in response to an external signal (e.g., I/O termination) at the end of any even/odd instruction pair (mid-instruction interrupts are permitted for some instructions), to save Processor status, and to restore the status at a later time without loss of continuity of the process.
3. Capability to fetch instruction pairs and to buffer two instructions (up to four instructions, depending on certain main store overlap conditions) including the one currently in execution.
4. Overlapping instruction-execution, address preparation, and instruction fetch. While an instruction is being executed, address preparation for the next operand (or even the operand following it) or the next instruction pair is taking place. The operations unit can be executing instruction N; instruction N+1 can be buffered in the operations unit (with its operand buffered in a main store port); and the control unit can be executing instructions N+2 or N+3 (if such execution does not involve the main store port or registers of instructions N or N+1), or preparing the address to fetch instructions N+4 and N+5.
5. Capability to detect main store instructions that alter the contents of buffered instructions. Ability to delay preprocessing of an address using register modification if the instruction currently in execution changes the register to be used in that modification.
6. Interlacing capability to direct main store accesses to the proper system controller module.
7. Intermediate storage of address and control information in high-speed registers addressable by content (Associative Memory).

8. Intermediate storage of base address and control information in pointer registers which are loaded by the executing program.
9. Absolute address computation at execution time.

SPECIAL CAPABILITIES

The Processor also includes several unique capabilities, such as hardware implemented segmentation and paging, address modification, address appending, and detection of faults and external interrupts. These features are summarized in this section and described in detail respectively in Sections V, VI, and VII.

Segmentation and Paging

A segment is a collection of data or instructions that is assigned a symbolic name and addressed symbolically by the user. Paging is at the discretion of the software; the user need not be aware of the existence of pages. User visible address preparation is concerned with the calculation of a segment effective address relative to the origin of the segment; the Processor hardware completes address preparation by translating the final segment effective address into an absolute main store address. The user may view each of his segments as residing in an independent main store unit. Each segment has its own origin which can be addressed as location zero. The size of each segment varies without affecting the addressing of the other segments. Each segment can be addressed like a conventional main store image starting at location zero. Maximum "segment-size" is 262,144 words.

When viewed from the Processor, main store consists of blocks or pages, each of which is defined as "page-size" words in length. (The page size used by Multics is 1024 words.) Each page begins at an absolute address which is zero modulo the page size. Any page of a segment can be placed in any available main store block. These pages may be addressed as if they were contiguous even though they are in widely scattered absolute locations. Only currently referenced pages need be in main store. If a segment is not paged, the complete segment is located in contiguous blocks of main store. In the current Multics implementation, all user segments are paged.

Address Modification and Address Appending

Prior to each main store access, two major phases of address preparation take place:

1. "Address-modification" by Register or Indirect Word content, if

specified by the Instruction Word or Indirect Word.

2. "Address-appending", in which a segment effective address is translated into an absolute address to access main store.

Although the above two types of modification are combined in most operations, they are described separately in Sections V and VI. The address modification procedure can go on indefinitely, with one type of modification leading to repetitions of the same type or to other types of modification prior

to a main store access for an operand. However, to simplify the descriptions in this manual, each type of address modification is described as if it were the first (and usually the only) modification prior to a main store access.

Faults and Interrupts

The Processor detects certain illegal procedures; faulty communication with the main store; programmed faults; certain external events; and arithmetic faults. Many of the Processor fault conditions are deliberately or inadvertently caused by the software and do not necessarily involve error conditions.

Similarly, the Processor communicates with the other system modules by setting and answering external interrupts. When a fault or interrupt is recognized, a trap results. This causes the forced execution of a pair of instructions in a main store location, unique to the fault or interrupt, known as the fault or interrupt vector. The first of the forced instructions may cause safe storage of the Processor status. The second instruction in a fault vector should be a transfer, or the faulting program will be resumed without the fault having been processed. "Faults" and "interrupts" are described in Section VII.

Interrupts and certain low priority faults are recognized only at specific times during the execution of an instruction pair. If, at these times, the Processor detects the presence of bit 28 in the Instruction Word, the trap is inhibited and program execution continues. The interrupt or fault signal is saved for future recognition and is reset only when the trap occurs.

PROCESSOR MODES OF OPERATION

There are three "modes of main store addressing" (Absolute Mode, Append Mode, and BAR Mode), and two modes of instruction execution (Normal Mode and Privileged Mode). These modes of operation and the functions performed are summarized in Table 1-1.

Instruction Modes

NORMAL MODE

Most instructions can be executed in the "Normal Mode" of operation. Certain instructions, classed as privileged, cannot be executed in Normal Mode. These are identified in the individual instruction descriptions. An attempt to execute privileged instructions while in the Normal Mode results in an Illegal Procedure Fault. In the Normal Mode, various restrictions are indicated in Segment Descriptor Words and Page Table Words, which are explained in Section V. Address Preparation uses the appending phase. The Processor executes in Normal Mode when the access bits of the Segment Descriptor Word specify a nonprivileged procedure.

PRIVILEGED MODE

In Privileged Mode, all instructions can be executed. Address Preparation uses the appending phase. The Processor executes in "Privileged Mode" when the access bits of the Segment Descriptor Word specify a privileged procedure and the execution ring is equal to zero. Refer to Sections V and VIII for more detailed information.

Addressing Modes

ABSOLUTE MODE

All instructions can be executed in the "Absolute Mode" and unrestricted access is permitted to privileged hardware features. Address Preparation for instruction fetches does not use the appending phase. During instruction fetches, the Procedure Pointer Register is ignored.

The Processor enters Absolute Mode immediately after a fault or interrupt and remains in Absolute Mode until it executes a transfer instruction whose operand is obtained via explicit use of the appending mechanism, that is, via explicit reference to one of the Pointer Register by the use of bit 29 of the Instruction Word (See Append Mode below).

APPEND MODE

The "Append Mode" is the most commonly used main store addressing mode. In this mode the final effective segment address is either added to the Procedure Pointer Register, or it is added to one of the eight Pointer Registers. If bit 29 of the Instruction Word contains a 0, then the Procedure Pointer Register is selected; otherwise, the Pointer Register given by bits 0-2 of the instruction word is selected.

BAR MODE

In "BAR"(Base-Address-Register) Mode, the 18-bit BAR is used. The BAR contains a 0 modulo 512 address bound in bit positions 9-17 and a 0 modulo 512 base address in bit positions 0-8. All addresses are relocated by adding the effective segment address to the base address in bits 0-8. The relocated address then becomes the final segment effective address as in Append Mode and is added to the Procedure Pointer Register. A process is kept within certain main store limits by subtracting the unrelocated effective address from the

address bound in bits 9-17. If the result is zero or negative, the relocated address would be out of range, and a Store Fault occurs.

Table 1-1. Modes of Operation

<u>FUNCTIONS</u>	<u>NORMAL</u>	<u>PRIVILEGED</u>	<u>ABSOLUTE</u>	<u>BAR</u>
Execute privileged instructions.	No	Yes	Yes	No
Main store address for instruction fetch.	Append	Append	Absolute	Procedure Pointer Register plus BAR base address.
Main store address for for operand fetch.	Append	Append	Append if bit 29 = 1, else Absolute.	Procedure Pointer Register plus BAR base address.
Restriction of access to other segments.	Some	Some	None	Total

PROCESSOR UNIT FUNCTIONS

Major functions of each principal logic element are listed below and are described in subsequent sections of this manual.

Appending Unit

- Controls data input/output to main store.
- Performs main store selection and interlace.
- Does address appending.
- Controls fault recognition.

Associative Memory Assembly

This assembly consists of sixteen 72-bit Page Table Word Associative Memory (PTWAM) registers and sixteen 108-bit Segment Descriptor Word Associative Memory (SDWAM) registers. These registers are used to hold pointers to most

recently used segments (SDWs) and pages (PTWs). This unit obviates the need for possible multiple main store accesses before obtaining an absolute main store address of an operand.

Control Unit

- Performs all Processor control functions.
- Performs address modification.
- Controls mode of operation (Privileged, Normal, etc.).
- Performs interrupt recognition.
- Decodes Instruction Words and Indirect Words.
- Performs Timer Register loading and decrementing.

Operation Unit

- Does fixed and floating binary arithmetic.
- Does shifting and Boolean operations.

Decimal Unit

- Does decimal arithmetic.
- Does character- and bit-string operations.

SECTION II

MACHINE INSTRUCTIONS

This section describes the comprehensive set of "machine" instructions for the Multics Processor. The presentation assumes that the reader is familiar with the general structure of the Processor, the representation of information, the data formats, and the method of address preparation. Additional information on these subjects appears near the beginning of this section and in Sections III through VI.

INSTRUCTION REPERIOIRE

The Processor interprets a 10 bit field of the Instruction Word as the Operation Code. This field size yields an instruction universe of 1024 of which 547 are implemented. The instruction population is divided into 456 Basic Operations and 91 Extended Instruction Set (EIS) Operations.

Arrangement of Instructions

Instructions in this section are presented alphabetically by their mnemonic codes within functional categories. However, an overall alphabetic listing of instruction codes and their names appears in Appendix 3 to aid the user in locating specific instructions via that code.

Basic Operations

The 456 "basic" operations in the Processor all require exactly one 36-bit machine word and are further subdivided into the following types:

181	Fixed Point Binary Arithmetic	75	Pointer Register
85	Boolean Operations	17	Miscellaneous
34	Floating Point Binary Arithmetic	28	Privileged
36	Transfer of Control		

Extended Instruction Set (EIS) Operations

The 91 "Extended" Instruction Set (EIS) Operations are further subdivided into 62 EIS Single-Word Instructions and 29 EIS Multi-Word Instructions.

EIS SINGLE-WORD OPERATIONS

The 62 "EIS" Single-Word Instructions load, store, and perform special arithmetic on the Address Registers (ARn) used to access bit- and character-string operands, and save/store Decimal Unit (DU) control information required to service a Processor fault. Like the Basic Operations, EIS Single-Word Instructions require exactly one 36 bit Machine Word.

EIS MULTI-WORD OPERATIONS

The 29 "EIS" Multi-Word Instructions perform Decimal Arithmetic and bit- and character-string operations. They require 3 or 4 36-bit Machine Words depending on individual Operand Descriptor requirements.

FORMAT OF INSTRUCTION DESCRIPTION

Each instruction in the repertoire is described in the following pages of this section. The descriptions are presented in the format shown below.

MNEMONIC	INSTRUCTION NAME	OP CODE (OCTAL)
FORMAT:	Figure or Figure reference	
SUMMARY:	Text and/or bit transfer equations	
MODIFICATIONS:	Text	
INDICATORS:	Text and/or logic statements	
NOTES:	Text	

Line 1: MNEMONIC, INSTRUCTION NAME, OP CODE (OCTAL)

This line has three parts that contain the following:

1. Mnemonic -- The "mnemonic" code for the Operation field of the assembler statement. The Multics assembler, ALM, recognizes this value and maps it into the appropriate binary pattern when generating the actual object code.
2. Instruction Name -- The name of the machine instruction from which the Mnemonic was derived.
3. Op Code (Octal) -- The octal value of the operation code for the instruction. A zero or a one in parentheses following an octal code indicates whether bit 27 (Op Code extension bit) of the instruction

word is OFF or ON.

Line 2: FORMAI

The layout and definition of the subfields of the instruction word or words is given here either as a Figure or as a reference to a Figure.

Line 3: SUMMARY

The change in the state of the processor affected by the execution of the instruction is described in a short and generally symbolic form. If reference is made to the state of an indicator in the SUMMARY, it is the state of the indicator before the instruction is executed.

Line 4: MODIFICATIONS

Those modifiers that cannot be used with the instruction are listed explicitly as exceptions either because they are not permitted or because their effect cannot be predicted from the general address modification procedure. (See "Effective Address Formation" in Section VI.)

Line 5: INDICATORS

Only those indicators are listed whose state can be changed by the execution of the instruction. In most cases, a condition for setting ON as well as one for setting OFF is stated. If only one of the two is stated, then the indicator remains unchanged if the condition is not met. Unless stated otherwise, the conditions refer to the contents of registers existing after instruction execution. Refer also to "Common Attributes of Instructions", later in this section.

Line 6: NOTES

This part of the description exists only in those cases where the SUMMARY is not sufficient for in depth understanding of the operation.

DEFINITIONS OF NOTATION AND SYMBOLS

Main Store Addresses

- Y** = the 18 low order bits of the final 24 bit main store address of the instruction operand after all address preparation is complete.
- Y-pair** = a symbol denoting that Y designates a pair of main store locations with successive addresses, the smaller address being even. When the main store address is even, it designates the pair Y(even), Y+1; and when it is odd, the pair Y-1, Y(odd). The main store location with the smaller (even) address contains the most significant part of a double-word operand or the first of a pair of instructions.
- Y-block_p** = a symbol denoting that Y designates a block of main store locations of 4-, 8-, or 16-word extent. For a block of p-word extent, the Processor assumes that Y-block_p is a 0 modulo p address and performs address incrementing through the block accordingly, stopping when the address next reaches a value 0 modulo p. Note the difference between Y-block addressing and Y-pair addressing that forces the address to be 0 modulo 2.
- Y-char_{pk}** = a symbol denoting that Y designates a character or string of characters in main store of character size p bits as described by the kth Operand Descriptor. p is specified by the data type field of Operand Descriptor k and may have values 4, 6, or 9. See Section VI, Effective Address Formation, for details of Operand Descriptors.
- Y-bit_k** = a symbol denoting that Y designates a bit or string of bits in main store as described by the kth Operand Descriptor. See Section VI, Effective Address Formation, for details of Operand Descriptors.

Index Values

When reference is made to the elements of a string of characters or bits in main store, the notation shown in Register Position and Contents below is used. The index used to show traversing a string of extent p may take any of the values in the interval (1,p) unless noted otherwise. The elements of a main store block are traversed explicitly by using the index as an addend to the given block address, e.g., Y-block8+m and Y-block4+2m+1.

Abbreviations and Symbols

- | | |
|-----------------------|--------------------------------------------------------------------------------------------|
| A | Accumulator Register |
| AR_n | Address Register n (n = 0, 1, 2, ..., 7)
(consists of: PRn.WORDNO; PRn.CHAR; PRn.BITNO) |
| AQ | Combined Accumulator-Quotient Register |
| BAR | Base Address Register |

C()	"Contents of"
CA	Computed Address
DSBR	Descriptor Segment Base Register
DSBR.ADDR	Descriptor Segment Base Address Register of DSBR
DSBR.BND	Descriptor Segment Bound Register of DSBR
DSBR.STACK	Stack Base Register of DSBR
DSBR.U	Unpaged Flag of DSBR
E	Exponent Register
EA	Combined Exponent and Accumulator Registers
EAQ	Combined Exponent-Accumulator-Quotient Register
ERN	Effective Ring Number
ESN	Effective Segment Number
IC	Instruction Counter
IR	Indicator Register
PPR	Procedure Pointer Register
PPR.PRR	Procedure Ring Register of PPR
PPR.PSR	Procedure Segment Register of PPR
PPR.IC	Instruction Counter Register of PPR
PPR.P	Privilege Flag of PPR
PRn	Pointer Register n (n = 0, 1, 2, ..., 7)
PRn.RNR	Ring Number Register of PRn
PRn.SNR	Segment Number Register of PRn
PRn.WORDNO	Word Address Register of PRn
PRn.CHAR	Character Address Register of PRn
PRn.BITNO	Bit Offset Register of PRn
Q	Quotient Register
PTWAM	Page Table Word Associative Memory
SDWAM	Segment Descriptor Word Associative Memory
RALR	Ring Alarm Register
TPR	Temporary Pointer Register
TPR.CA	Computed Address Register of TPR
TPR.TRR	Temporary Ring Register of TPR
TPR.TSR	Temporary Segment Register of TPR
TPR.TBR	Temporary Bit Register of TPR
TR	Timer Register
Xn	Index Register n (n = 0, 1, 2, ..., 7)
Z	Temporary pseudo-result of a nonstore comparative operation

Register Positions and Contents

("R" standing for any of the registers listed above as well as for main store words, word-pairs, word-blocks, and character strings.

R _i	the i th bit position of R
R(i)	the i th register of a set of n registers, R
R _{i,j}	the bit positions i through j of R
C(R)	the contents of the full register R
C(R) _i	the contents of the i th bit or character of R
C(R) _{i,j}	the contents of the bits or characters i through j of R
xx...x	a string of binary bits (0's or 1's) of any necessary length

When the description of an instruction specifies a change for a part of a register or main store location, it is understood that the part of the register or main store location not mentioned remains unchanged.

Other Symbols

->	replaces
::	compare with
&	the Boolean connective AND
	the Boolean connective OR
⊕	the Boolean connective NON-EQUIVALENCE (or EXCLUSIVE OR)
~	the Boolean unary NOT operator
≠	not equal
n^m	indicates exponentiation (n and m are integers); for example, the fifth power of 2 is represented as 2^5 .
x	multiplication; for example, C(Y) times C(Q) is represented as $C(Y) \times C(Q)$.
/	division; for example, C(Y) divided by C(A) is represented as $C(Y) / C(A)$.
	concatenation; for example, string1 string2.
...	the absolute value of the value between vertical bars (no algebraic sign). For example the absolute value of C(A) plus C(Y) is represented as $ C(A) + C(Y) $.

COMMON ATTRIBUTES OF INSTRUCTIONS

Illegal Modification

If an "illegal" modifier is used with any instruction, an Illegal Procedure Fault with a subcode class of Illegal Modifier occurs.

Parity Indicator

The Parity Indicator is turned ON at the end of a main store access which has incorrect parity.

INSTRUCTION WORD FORMATS

Basic and EIS Single-Word Instructions

The "Basic" Instructions and "EIS" Single-Word Instructions require exactly one 36 bit Machine Word and are interpreted according to the format shown in Figure 2-1 below.

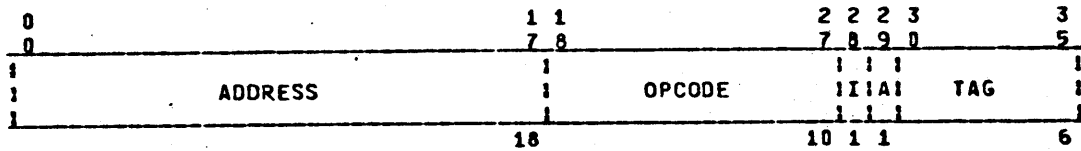


Figure 2-1 Basic and EIS Single-Word Instruction Format

- ADDRESS** The given address of the Operand or Indirect Word. This address may be:
- An 18 bit main store address if A = 0 (Absolute Mode only)
 - An 18 bit offset to the Base Address Register if A = 0 (BAR Mode only)
 - An 18 bit offset relative to the base of the current procedure segment if A = 0 (Appending Mode only)
 - A 3 bit Pointer Register number (n) and a 15 bit offset to C(PRN.WORDNO) if A = 1 (Absolute and Appending Modes only)
 - A 3 bit Address Register number (n) and a 15 bit offset to C(ARN) if A = 1 (All modes depending on instruction type)
 - An 18 bit literal signed or unsigned constant (All modes depending on instruction type and Modifier)
 - An 8 bit Shift Operation count (All modes)
 - An 18 offset to the current value of the Instruction Counter C(PRR.IC) (All modes)
- OPCODE** Instruction operation code.
- I** Program Interrupt inhibit bit. When this bit is set, the Processor will ignore all external Program Interrupt signals. See Section VII, Faults and Interrupts, for details.

EIS Multi-Word Instructions

The "EIS Multi-Word Instructions" require 3 or 4 Machine Words depending on the Operand Descriptor requirements of the individual instructions. The words are interpreted according to the format shown in Figure 2-3 below.

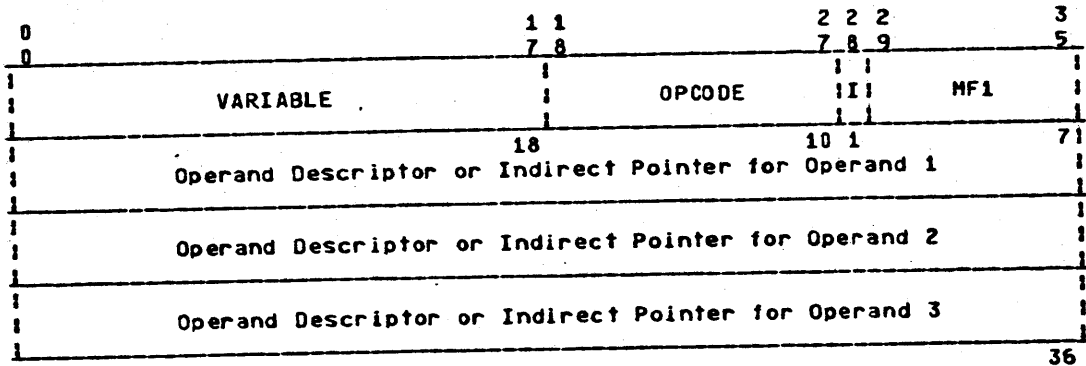


Figure 2-3 EIS Multi-Word Instruction Format

*VARIABLE	This field is interpreted variously according to the requirements of the individual EIS Instructions. Its interpretation is given under FORMAT for each EIS Instruction. The Modification Fields MF2 and MF3 are contained in this field if they are required.
OPCODE	Instruction operation code as for Basic and EIS Single-Word Instructions.
I	Program Interrupt inhibit bit as for Basic and EIS Single-Word Instructions.
MF1	Modification Field for Operand Descriptor 1. See EIS Modification Fields (MF) below for details.

EIS Modification Fields (MF)

Each of the Operand Descriptors following an EIS Multi-Word Instruction Word has a "Modification Field" in the Instruction Word. The Modification Field

controls the interpretation of the Operand Descriptor. The Modification Field is interpreted according to the format shown in Figure 2-4.

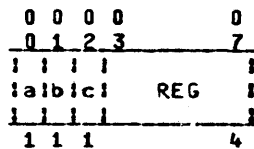


Figure 2-4 EIS Modification Field (MF) Format

- key**
- a AR Address Register flag. This flag controls interpretation of the ADDRESS field of the Operand Descriptor just as the "A" flag controls interpretation of the ADDRESS field of the Basic and EIS Single-Word Instructions.
 - b RL Register length control. If RL = 0, then the Length (N) field of the Operand Descriptor contains the length of the operand. If RL = 1, then the Length (N) field of the Operand Descriptor contains a selector value specifying a register holding the operand length.
 - c ID Indirect descriptor control. If ID = 1 for MF_k, then the kth word following the Instruction Word is an Indirect Pointer to the Operand Descriptor for the kth operand; otherwise, that word is the Operand Descriptor.
- REG** The register number for R-type modification (if any) of ADDRESS of the Operand Descriptor. These modifications are similar to R-type modifications for Basic Instructions and are summarized in Table 2-1 below. Illegal modifiers have the entry "IPR" and cause an Illegal Procedure Fault.

Table 2-1 R-type Modifiers for REG Fields

Octal Code	R-type	MF, REG	Meaning as used in Indirect Operand Descriptor Pointer	C(Operand Descriptor) 32, 35
00	N	N	N	IPR
01	AU	AU	AU	AU
02	QU	QU	QU	QU
03	DU	IPR(a)	IPR	IPR
04	IC	IC(b)	IC(b)	IPR
05	AL	A(c)	AL	A(c)
06	QL	Q(c)	QL	Q(c)
07	DL	IPR	IPR	IPR
10	X0	X0	X0	X0
11	X1	X1	X1	X1
12	X2	X2	X2	X2
13	X3	X3	X3	X3
14	X4	X4	X4	X4
15	X5	X5	X5	X5
16	X6	X6	X6	X6
17	X7	X7	X7	X7

- (a) The DU modifier is permitted only in the second Operand Descriptor of the SCD, SCDR, SCM, and SCMR instructions to specify that the test character(s) reside(s) in bits 0-18 of the Operand Descriptor.
- (b) The IC modifier is permitted only in the REG field of Indirect Pointers and in MF3.REG for the SCD, SCDR, SCM, SCMR, MVT, TCT, and TCTR instructions, that is, the instructions that store summary results of a scan operation. C(IC) is always interpreted as a WORD offset.
- (c) The limit of addressing extent of the processor is $2^{*18} - 1$ words; that is, given any main store address, Y, a modifier may be employed to access a main store word anywhere in the range $(Y - 2^{*18} + 1, Y + 2^{*18} - 1)$, provided other address range constraints are not violated. Since it is desirable to address this same extent as words, characters, and bits it is necessary to provide a register with range greater than the 12 bits of N or the 18 bits of normal R-type modifiers. This is done by extending the range of the A and Q modifiers as follows...

Mode	Range	A,Q bits
9-bit	20	16,35
6-bit	21	15,35
4-bit	21	15,35
bit	24	12,35

The unused high order bits are ignored.

EIS Operand Descriptors and Indirect Pointers

The words following an EIS Multi-Word Instruction Word are either descriptions of the operands or "Indirect Pointers" to the operand descriptions. The interpretation of the words is performed according to the settings of the control bits in the associated Modification Field (MF). The *k*th Word following the Instruction Word is interpreted according to the contents of MF_{*k*}. See EIS Modifications Fields (MF) above for meaning of the various control bits.

See Section III, Data Representation, and Section VI, Effective Address Formation, for further details.

"OPERAND" DESCRIPTOR "INDIRECT" POINTER "FORMAT"

If MF_{*k*}.ID = 1, then the *k*th word following an EIS multi-word Instruction Word is not an Operand Descriptor, but is an Indirect Pointer to an Operand Descriptor and is interpreted as shown in Figure 2-5.

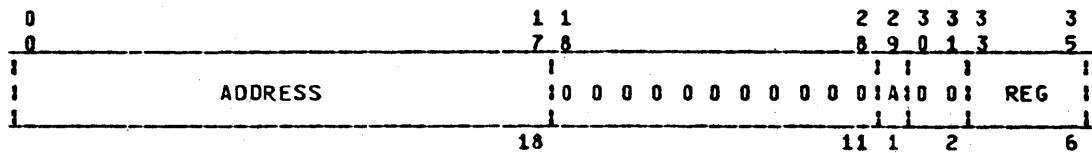


Figure 2-5 Operand Descriptor Indirect Pointer Format

ADDRESS The given address of the Operand Descriptor. This address may be:

- An 18 bit main store address if A = 0 (Absolute Mode only)
- An 18 bit offset relative to the Base Address Register (BAR) if A = 0 (BAR Mode only)
- An 18 bit offset relative to the base of the current procedure segment if A = 0 (Appending Mode only)
- A 3 bit Pointer Register number (p) and a 15 bit offset relative to C(PR_p.WORDNO) if A = 1 (All modes)

A Indirect via Pointer Register flag. This flag controls interpretation of the ADDRESS field of the Indirect Pointer just as the "A" flag controls interpretation of the ADDRESS field of the Basic and EIS Single-Word Instructions.

***REG** Address modifier for ADDRESS. All Register Modifiers except DU and DL may be used. If IC is used, then ADDRESS is an 18 bit offset to value of the Instruction Counter for the Instruction Word. C(REG) is always interpreted as a word offset to ADDRESS.

ALPHANUMERIC OPERAND DESCRIPTOR FORMAT

For any operand of an EIS Multi-word Instruction that requires Alphanumeric Data, the Operand Descriptor is interpreted as shown in Figure 2-6 below.

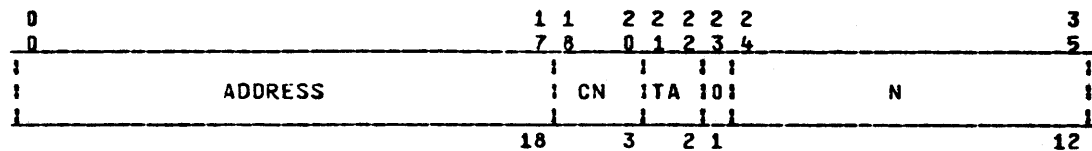


Figure 2-6 Alphanumeric Operand Descriptor Format

ADDRESS The given address of the operand. This address may be (for the kth operand):

- An 18 bit main store address if MF_k.AR = 0 (Absolute Mode only)

An 18 bit offset to the Base Address Register if MF_K.AR = 0 (BAR Mode only)

An 18 bit offset relative to the base of the current procedure segment if MF_K.AR = 0 (Appending Mode only)

A 3 bit Address Register number (n) and a 15 bit word offset to C(AR_n) if MF_K.AR = 1 (All modes)

CN

Character Number. This field gives the character position within the word at ADDRESS of the first operand character. Its interpretation depends on the Data Type (See TA below) of the operand. Table 2-2 below shows the interpretation of the field. A digit in the table indicates the corresponding character position (See Section III, Data Representation, for data formats) and an "x" indicates an invalid code for the Data Type. Invalid codes cause Illegal Procedure Faults.

Table 2-2 Alphanumeric Character Number (CN) Codes

C(CN)	Data Type		
	4-bit	6-bit	9-bit
000	0	0	0
001	1	1	x
010	2	2	1
011	3	3	x
100	4	4	2
101	5	5	x
110	6	x	3
111	7	x	x

TA

Type Alphanumeric. This is the Data Type code for the operand. The interpretation of the field is shown in Table 2-3 below. The code shown as Invalid causes an Illegal Procedure Fault.

Table 2-3 Alphanumeric Data Type (TA) Codes

C(TA)	Data Type
00	9-bit
01	6-bit
10	4-bit
11	Invalid

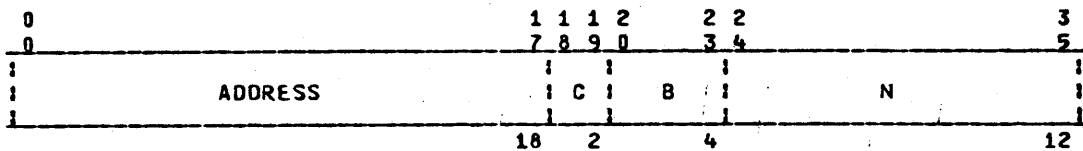
N

Operand length. If MF_K.RL = 0, this field contains the string length of the operand. If MF_K.RL = 1, this field contains the code for a register holding the operand string length. See Table 2-1 and EIS Modification Fields (MF) above for a discussion of register codes.

- SF** Scaling factor. This field contains the two's complement value of the base 10 scaling factor; that is, the value of m for numbers represented as $d \times 10^{**}m$. The decimal point is assumed to the right of the least significant digit of d . Negative values move the decimal point to the left; positive values, to the right. The range of m is (-32,31).
- N** Operand length. If $MF_k.RL = 0$, this field contains the operand length in digits. If $MF_k.RL = 1$, it contains the REG code for the register holding the operand length and $C(REG)$ is treated as a 0 modulo 64 number.

~BIT~STRING~OPERAND~DESCRIPTOR~FORMAT~

For any operand of an EIS Multi-word Instruction that requires Bit-string Data, the Operand Descriptor is interpreted as shown in Figure 2-8 below.



FIXED POINT DATA MOVEMENT LOAD

FIXED POINT ARITHMETIC INSTRUCTIONS

Fixed Point Data Movement Load

EAA Effective Address to A 635 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Y -> C(A)0,17
00...0 -> C(A)18,35

MODIFICATIONS: All except DU, DL

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A) bit 0 = 1, then ON; otherwise OFF

NOTES: The EAA instruction, and the instructions EAQ and EAXn, facilitate interregister data movements; the data source is specified by the address modification, and the data destination by the operation code of the instruction.

Attempted repetition with RPL causes an Illegal Procedure Fault.

EAQ Effective Address to Q 636 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Y -> C(Q)0,17
00...0 -> C(Q)18,35

MODIFICATIONS: All except DU, DL

INDICATORS: (Indicators not listed are not affected)

Zero If C(Q) = 0, then ON; otherwise OFF

Negative If C(Q)0 = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

FIXED POINT DATA MOVEMENT LOAD

LCAQ Load Complement AQ 337 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If $C(Y\text{-pair}) \neq 0$, then $-C(Y\text{-pair}) \rightarrow C(AQ)$
 otherwise, $00\dots0 \rightarrow C(AQ)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF

Negative If $C(AQ)_0 = 1$, then ON; otherwise OFF

Overflow If range of AQ is exceeded, then ON; otherwise OFF

NOTES: The LCAQ instruction changes the number to its negative (if $\neq 0$) while moving it from Y-pair to AQ. The operation is executed by forming the two's complement of the string of 72 bits.

LCQ Load Complement Q 336 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If $C(Y) \neq 0$, then $-C(Y) \rightarrow C(Q)$
 otherwise, $00\dots0 \rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Q) = 0$, then ON; otherwise OFF

Negative If $C(Q)_0 = 1$, then ON; otherwise OFF

Overflow If range of Q is exceeded, then ON; otherwise OFF

NOTES: The LCQ instruction changes the number to its negative (if $\neq 0$) while moving it from Y to Q. The operation is executed by forming the two's complement of the string of 36 bits.

FIXED POINT DATA MOVEMENT LOAD

LCXn Load Complement Xn 32n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
 If C(Y)0,17 ≠ 0, then -C(Y)0,17 → C(Xn)
 otherwise, 00...0 → C(Xn)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(Xn) = 0, then ON; otherwise OFF

Negative If C(Xn)0 = 1, then ON; otherwise OFF

Overflow If range of Xn is exceeded, then ON; otherwise OFF

NOTES: The LCXn instruction changes the number to its negative (if ≠ 0) while moving it from Y0,17 to Xn. The operation is executed by forming the two's complement of the string of 18 bits.

Attempted repetition with RPL and with the same register given as target and modifier causes an Illegal Procedure Fault.

LDA Load A 235 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: *content of computed address*
 C(Y) → C(A)

MODIFICATIONS: All

LDA SP: 26 (instead of any pointer)

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A)0 = 1, then ON; otherwise OFF

FIXED POINT DATA MOVEMENT LOAD

LDAC Load A and Clear 034 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y) -> C(A)
00...0 -> C(Y)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A)0 = 1, then ON; otherwise OFF

NOTES: The LDAC instruction causes a special main store reference that performs the load and clear in one cycle. Thus, this instruction can be used in locking data.

LDAQ Load AQ 237 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y-pair) -> C(AQ)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

LDI Load Indicator Register 634 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y)18,31 -> C(IR)

MODIFICATIONS: All except CI, SC, SCR

FIXED POINT DATA MOVEMENT LOAD

INDICATORS: (Indicators not listed are not affected)

Parity Mask If C(Y)27 = 1, and the Processor is in Absolute or Privileged Mode, then ON; otherwise OFF. This indicator is not affected in the Normal or BAR modes.

Not BAR Mode Cannot be changed by the LDI instruction

Multiword Instruction Fault If C(Y)30 = 1, and the Processor is in Absolute or Privileged mode, then ON; otherwise OFF. This indicator is not affected in Normal or BAR modes.

Absolute Mode Cannot be changed by the LDI instruction

All Other Indicators If corresponding bit in C(Y) is 1, then ON; otherwise, OFF

NOTES:

The relation between C(Y)18,31 and the indicators is given in Table 2-5 below.

The Tally Runout indicator reflects C(Y)25 regardless of what address modification is performed on the LDI instruction for tally operations.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Table 2-5. Relation Between Data Bits and Indicators

<u>Bit Position C(Y)</u>	<u>Indicator</u>
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Undeflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Not BAR Mode
29	Truncation
30	Multiword Instruction Fault (MIF)
31	Absolute Mode

FIXED POINT DATA MOVEMENT LOAD

LDQ	Load Q	236 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	C(Y) -> C(Q)	
MODIFICATIONS:	All	
INDICATORS:	(Indicators not listed are not affected)	
Zero	If C(Q) = 0, then ON; otherwise OFF	
Negative	If C(Q)0 = 1, then ON; otherwise OFF	
LDQC	Load Q and Clear	034 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	C(Y) -> C(Q) 00...0 -> C(Y)	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	(Indicators not listed are not affected)	
Zero	If C(Y) = 0, then ON; otherwise OFF	
Negative	If C(Y)0 = 1, then ON, otherwise OFF	
NOTES:	The LDQC instruction causes a special main store reference that performs the load and clear in one cycle. Thus, this instruction can be used in locking data.	
LDXn	Load Index Register Xn	32n (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	For n = 0, 1, ..., or 7 as determined by operation code C(Y)0,17 -> C(Xn)	
MODIFICATIONS:	All except CI, SC, SCR	

FIXED POINT DATA MOVEMENT LOAD

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Xn) = 0$, then ON; otherwise OFF

Negative If $C(Xn)0 = 1$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL and with the same register given as target and modifier causes an Illegal Procedure Fault.

LREG Load Registers 073 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY:

$C(Y)0,17 \rightarrow C(X0)$	$C(Y)18,35 \rightarrow C(X1)$
$C(Y+1)0,17 \rightarrow C(X2)$	$C(Y+1)18,35 \rightarrow C(X3)$
$C(Y+2)0,17 \rightarrow C(X4)$	$C(Y+2)18,35 \rightarrow C(X5)$
$C(Y+3)0,17 \rightarrow C(X6)$	$C(Y+3)18,35 \rightarrow C(X7)$
$C(Y+4) \rightarrow C(A)$	$C(Y+5) \rightarrow C(Q)$
$C(Y+6)0,7 \rightarrow C(E)$	

where Y must be 0 modulo 8; otherwise, the next smaller such address is used.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LXLn Load Xn from Lower 72n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, \text{ or } 7$ as determined by operation code
 $C(Y)18,35 \rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, SCR

FIXED POINT DATA MOVEMENT LOAD

INDICATORS: (Indicators not listed are not affected)

Zero If $C(X_n) = 0$, then ON; otherwise OFF

Negative If $C(X_n) \neq 0$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL and with the same register given as target and modifier causes an Illegal Procedure Fault.

Fixed Point Data Movement Store

SBAR Store Base Address Register 350 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(BAR) → C(Y)0,17

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SREG Store Registers 753 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(X0) → C(Y)0,17 C(X1) → C(Y)18,35
 C(X2) → C(Y+1)0,17 C(X3) → C(Y+1)18,35
 C(X4) → C(Y+2)0,17 C(X5) → C(Y+2)18,35
 C(X6) → C(Y+3)0,17 C(X7) → C(Y+3)18,35
 C(A) → C(Y+4) C(Q) → C(Y+5)
 C(E) → C(Y+6)0,7 00...0 → C(Y+6)8,35
 C(TR) → C(Y+7)0,26 00...0 → C(Y+7)27,32
 C(RALR) → C(Y+7)33,35

where Y must be a 0 modulo 8 address; otherwise the next lower such address is used.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

FIXED POINT DATA MOVEMENT STORE

STA Store A 755 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(A) -> C(Y)

MODIFICATIONS: All except DU, DL

INDICATORS: None affected

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

STAC Store A Conditional C(Y) = 0 354 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If C(Y) = 0, then C(A) -> C(Y)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If initial C(Y) = 0, then ON; otherwise OFF

NOTES: If the initial C(Y) is nonzero, then C(Y) is not changed by the STAC instruction.

Attempted repetition with RPL causes an Illegal Procedure Fault.

STACQ Store A Conditional C(Y) = C(Q) 654 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If C(Y) = C(Q), then C(A) -> C(Y)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If initial C(Y) = C(Q), then ON; otherwise OFF

FIXED POINT DATA MOVEMENT STORE

NOTES: If the initial C(Y) is \neq C(Q), then C(Y) is not changed by the STACQ instruction.
 Attempted repetition with RPL causes an Illegal Procedure Fault.

STAQ Store AQ 757 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(AQ) \rightarrow C(Y-pair)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

STBA Store Character of A (Nine Bit) 551 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Characters of C(A) \rightarrow Corresponding Characters of C(Y), the character positions affected being specified in the tag field.

MODIFICATIONS: None

INDICATORS: None affected

NOTES: Binary ones in the tag field of this instruction specify the character positions of A and Y that are affected. The control relations are shown in Table 2-6.
 Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

FIXED POINT DATA MOVEMENT STORE

Table 2-6. Control Relations for Store Character Instructions (Nine Bit)

Bit Position Within Tag Field	Bit of Instruction	Structure of A and Y
0	30	Char 0 (bits 0-8)
1	31	Char 1 (bits 9-17)
2	32	Char 2 (bits 18-26)
3	33	Char 3 (bits 27-35)

STBQ Store Character of Q (Nine Bit) 552 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Characters of C(Q) -> Corresponding Characters of C(Y), the character positions affected being specified in the tag field.

MODIFICATIONS: None

INDICATORS: None affected

NOTES: Binary ones in the tag field of this instruction specify the character positions of Q and Y that are affected. The control relations are shown in Table 2-6 above.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

STC1 Store Instruction Counter Plus 1 554 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(PPR.IC) + 1 -> C(Y)0,17
C(IR) -> C(Y)18,31
00...0 -> C(Y)32,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

FIXED POINT DATA MOVEMENT STORE

INDICATORS: None affected

NOTES: The contents of the Instruction Counter and the Indicator Register after address preparation are stored in C(Y)0,17 and C(Y)18,31, respectively. C(Y)25 reflects the state of the Tally Runout indicator prior to modification. The relationship between the C(Y)18,31 and the indicators are given in Table 2-5.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

STC2 Store Instruction Counter Plus 2 750 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(PPR.IC) + 2 -> C(Y)0,17

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

STCA Store Character of A (Six Bit) 751 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Characters of C(A) -> Corresponding Characters of C(Y), the character positions affected being specified in the tag field.

MODIFICATIONS: None

INDICATORS: None affected

NOTES: Binary ones in the tag field of this instruction specify

character positions of A and Y that are affected. The control relations are shown in Table 2-7.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

FIXED POINT DATA MOVEMENT STORE

Table 2-7. Control Relations for Store Character Instructions (Six Bit)

<u>Bit Position Within Tag Field</u>	<u>Bit of Instruction</u>	<u>Structure of A and Y</u>
0	30	Char 0 (bits 0-5)
1	31	Char 1 (bits 6-11)
2	32	Char 2 (bits 12-17)
3	33	Char 3 (bits 18-23)
4	34	Char 4 (bits 24-29)
5	35	Char 5 (bits 30-35)

STCQ Store Character of Q (Six Bit) 752 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Characters of C(Q) -> Corresponding Characters of C(Y),
the character positions affected being specified by the
tag field.

MODIFICATIONS: None

INDICATORS: None affected

NOTES: Binary ones in the tag field of this instruction specify
the character positions of Q and Y that are affected. The
control relations are shown in Table 2-7 above.

Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

STCD Store Control Double 357 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: 00...0 -> C(Y-pair)0,2
 C(PPR.PSR) -> C(Y-pair)3,17
 C(PPR.PRR) -> C(Y-pair)18,20
 00...0 -> C(Y-pair)21,29
 43 (octal) -> C(Y-pair)30,35
 C(PPR.IC)+2 -> C(Y-pair)36,53
 00...0 -> C(Y-pair)54,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes Y17 = 0; no check is made.
 Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

STI Store Indicator Register 754 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(IR) -> C(Y)18,31
 00...0 -> C(Y)32,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The contents of the Indicator Register after address preparation are stored in C(Y)18,31. C(Y)25 reflects the

state of the Tally Runout indicator prior to address preparation. The relation between C(Y)18,31 and the Indicators is given in Table 2-5.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

FIXED POINT DATA MOVEMENT STORE

STQ Store Q 756 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Q) -> C(Y)

MODIFICATIONS: All except DU, DL

INDICATORS: None affected

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

STT Store Timer Register 454 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(TR) -> C(Y)0,26
00...0 -> C(Y)27,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

STXn Store Xn in Upper 74n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
C(Xn) -> C(Y)0,17

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

FIXED POINT DATA MOVEMENT SHIFT

Fixed Point Data Movement Shifts

ALR A Left Rotate 775 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(A) left the number of positions specified Y11,17; enter each bit leaving A0 into A35.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

 Zero If C(A) = 0, then ON; otherwise OFF

 Negative If C(A)0 = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

ALS A Left Shift 735 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(A) left the number of positions specified by Y11,17; fill vacated positions with zeros.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

 Zero If C(A) = 0, then ON; otherwise OFF

 Negative If C(A)0 = 1, then ON; otherwise OFF

 Carry If C(A)0 changes during the shift, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure

Fault.

FIXED POINT DATA MOVEMENT SHIFT

ARL A Right Logic 771 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(A) right the number of positions specified by Y 11,17; fill vacated positions with zeros.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A)0 = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

ARS A Right Shift *arithmetic shift* 731 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(A) right the number of positions specified by Y11,17; fill vacated positions with C(A)0.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A)0 = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

LLR Long Left Rotate 777 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(AQ) left by the number of positions specified by Y11,17; enter each bit leaving AQ0 into AQ71.

FIXED POINT DATA MOVEMENT SHIFT

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ) = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

LLS Long Left Shift 737 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(AQ) left the number of positions specified by Y11,17; fill vacated positions with zeros.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ) = 1, then ON; otherwise OFF

Carry If C(AQ) changes during the shift, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

LRL Long Right Logic 773 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(AQ) right the number of positions specified by Y11,17; fill vacated positions with zeros.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

FIXED POINT DATA MOVEMENT SHIFT

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

LRS Long Right Shift 733 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(AQ) right the number of positions specified by Y11,17; fill vacated positions with C(A)0.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

QLR Q Left Rotate 776 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(Q) the number of positions specified by Y11,17; enter each bit leaving Q0 into Q35.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(Q) = 0, then ON; otherwise OFF

Negative If C(Q)0 = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF
 Negative If $C(AQ)0 = 1$, then ON; otherwise OFF
 Overflow If range of AQ is exceeded, then ON; otherwise OFF
 Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: A 72-bit number is formed from C(Y) in the following manner:

The lower 36 bits (36,71) is identical to C(Y). Each of the upper 36 bits (0,35) is identical to C(Y)0.

This 72-bit number is added to the contents of the combined AQ-register.

ADLA Add Logical to A 035 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A) + C(Y) \rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(A) = 0$, then ON; otherwise OFF
 Negative If $C(A)0 = 1$, then ON; otherwise OFF
 Carry If a carry out of A0 is generated, then ON; otherwise OFF

NOTES: The ADLA instruction is identical to the ADA instruction with the exception that the Overflow indicator is not affected by the ADLA instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

FIXED POINT ADDITION

ADLAQ Add Logical to AQ 037 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(AQ) + C(Y-pair) -> C(AQ)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: The ADLAQ instruction is identical to the ADAQ instruction with the exception that the Overflow indicator is not affected by the ADLAQ instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

ADLQ Add Logical to Q 036 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Q) + C(Y) -> C(Q)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If C(Q) = 0, then ON; otherwise OFF

Negative If C(Q)0 = 1, then ON; otherwise OFF

Carry If a carry out of Q0 is generated, then ON; otherwise OFF

NOTES: The ADLQ instruction is identical to the ADQ instruction

with the exception that the Overflow indicator is not affected by the ADLQ instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

ADLXn Add Logical to Xn 02n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, \text{ or } 7$ as determined by operation code
 $C(Xn) + C(Y)0,17 \rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Xn) = 0$, then ON; otherwise OFF

Negative If $C(Xn)0 = 1$, then ON; otherwise OFF

Carry If a carry out of Xn0 is generated, then ON; otherwise OFF

NOTES: The ADLXn instruction is identical to the ADXn instruction with the exception that the Overflow indicator is not affected by the ADLXn instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

ADQ Add to Q 076 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q) + C(Y) \rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Q) = 0$, then ON; otherwise OFF

Negative If $C(Q)0 = 1$, then ON; otherwise OFF

Overflow If range of Q is exceeded, then ON; otherwise OFF

Carry If a carry out of Q0 is generated, then ON; otherwise OFF

ASA	Add Stored to A	055 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	$C(A) + C(Y) \rightarrow C(Y)$	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	(Indicators not listed are not affected)	
Zero	If $C(Y) = 0$, then ON; otherwise OFF	
Negative	If $C(Y)0 = 1$, then ON; otherwise OFF	
Overflow	If range of Y is exceeded, then ON; otherwise OFF	
Carry	If a carry out of Y0 is generated, then ON; otherwise OFF	
NOTES:	Attempted repetition with RPL causes an Illegal Procedure Fault.	
ASQ	Add Stored to Q	056 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	$C(Q) + C(Y) \rightarrow C(Y)$	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	(Indicators not listed are not affected)	
Zero	If $C(Y) = 0$, then ON; otherwise OFF	
Negative	If $C(Y)0 = 1$, then ON; otherwise OFF	
Overflow	If range of Y is exceeded, then ON; otherwise OFF	
Carry	If a carry out of Y0 is generated, then ON; otherwise OFF	
NOTES:	Attempted repetition with RPL causes an Illegal Procedure Fault.	

FIXED POINT ADDITION

ASXn Add Stored to Xn 04n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, \text{ or } 7$ as determined by operation code
 $C(Xn) + C(Y)_{0,17} \rightarrow C(Y)_{0,17}$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y)_{0,17} = 0$, then ON; otherwise OFF

Negative If $C(Y)_0 = 1$, then ON; otherwise OFF

Overflow If range of $Y_{0,17}$ is exceeded, then ON; otherwise OFF

Carry If a carry out of Y_0 is generated, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

AWCA Add with Carry to A 071 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Carry indicator OFF, then $C(A) + C(Y) \rightarrow C(A)$
 If Carry indicator ON, then $C(A) + C(Y) + 1 \rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(A) = 0$, then ON; otherwise OFF

Negative If $C(A)_0 = 1$, then ON; otherwise OFF

Overflow If range of A is exceeded, then ON; otherwise OFF

Carry If a carry out of A_0 is generated, then ON; otherwise OFF

NOTES: The AWCA instruction is identical to the ADA instruction with the exception that when the Carry indicator is ON at the beginning of the instruction, 1 is added to the sum of $C(A)$ and $C(Y)$.

AWCQ

Add with Carry to Q

072 (0)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

If Carry indicator OFF, then $C(Q) + C(Y) \rightarrow C(Q)$ If Carry indicator ON, then $C(Q) + C(Y) + 1 \rightarrow C(Q)$

MODIFICATIONS:

All

INDICATORS:

(Indicators not listed are not affected)

Zero

If $C(Q) = 0$, then ON; otherwise OFF

Negative

If $C(Q)0 = 1$, then ON; otherwise OFF

Overflow

If range of Q is exceeded, then ON; otherwise OFF

Carry

If a carry out of Q0 is generated, then ON; otherwise OFF

NOTES:

The AWCQ instruction is identical to the ADQ instruction with the exception that when the Carry indicator is ON at the beginning of the instruction, 1 is added to the sum of $C(Q)$ and $C(Y)$.

FIXED POINT SUBTRACTION

Fixed-Point-Subtraction

SBA Subtract from A 175 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A) - C(Y) \rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(A) = 0$, then ON; otherwise OFF

Negative If $C(A)0 = 1$, then ON; otherwise OFF

Overflow If range of A is exceeded, then ON; otherwise OFF

Carry If a carry out of A0 is generated, then ON; otherwise OFF

SBAQ Subtract from AQ 177 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(AQ) - C(Y\text{-pair}) \rightarrow C(AQ)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF

Negative If $C(AQ)0 = 1$, then ON; otherwise OFF

Overflow If range of AQ is exceeded, then ON; otherwise OFF

Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

SBLA Subtract Logical from A 135 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A) - C(Y) \rightarrow C(A)$

FIXED POINT SUBTRACTION

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A)0 = 1, then ON; otherwise OFF

Carry If a carry out of A0 is generated, then ON; otherwise OFF

NOTES: The SBLA instruction is identical to the SBA instruction with the exception that the Overflow indicator is not affected by the SBLA instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

SBLAQ Subtract Logical from AQ 137 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(AQ) - C(Y-pair) -> C(AQ)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: The SBLAQ instruction is identical to the SBAQ instruction with the exception that the Overflow indicator is not affected by the SBLAQ instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

SBLQ Subtract Logical from Q 136 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Q) - C(Y) -> C(Q)

MODIFICATIONS: All

FIXED POINT SUBTRACTION

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Q) = 0$, then ON; otherwise OFF

Negative If $C(Q) \oplus 1 = 1$, then ON; otherwise OFF

Carry If a carry out of Q_0 is generated, then ON; otherwise OFF

NOTES: The SBLQ instruction is identical to the SBQ instruction with the exception that the Overflow indicator is not affected by the SBLQ instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

SBLXn Subtract Logical from Xn 12n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, \text{ or } 7$ as determined by operation code
 $C(Xn) - C(Y)_{0,17} \rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Xn) = 0$, then ON; otherwise OFF

Negative If $C(Xn) \oplus 1 = 1$, then ON; otherwise OFF

Carry If a carry out of Xn_0 is generated, then ON; otherwise OFF

NOTES: The SBLXn instruction is identical to the SBXn instruction with the exception that the Overflow indicator is not affected by the SBLXn instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

SBQ Subtract from Q 176 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q) - C(Y) \rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

- Zero If $C(Q) = 0$, then ON; otherwise OFF
- Negative If $C(Q)0 = 1$, then ON; otherwise OFF
- Overflow If range of Q is exceeded, then ON; otherwise OFF
- Carry If a carry out of Q0 is generated, then ON; otherwise OFF

SBXn Subtract from Xn 16n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, \text{ or } 7$ as determined by operation code
 $C(Xn) - C(Y)0,17 \rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

- Zero If $C(Xn) = 0$, then ON; otherwise OFF
- Negative If $C(Xn)0 = 1$, then ON; otherwise OFF
- Overflow If range of Xn is exceeded, then ON; otherwise OFF
- Carry If a carry out of Xn0 is generated, then ON; otherwise OFF

SSA Subtract Stored from A 155 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A) - C(Y) \rightarrow C(Y)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

- Zero If $C(Y) = 0$, then ON; otherwise OFF
- Negative If $C(Y)0 = 1$, then ON; otherwise OFF
- Overflow If range of Y is exceeded, then ON; otherwise OFF
- Carry If a carry out of Y0 is generated, then ON; otherwise OFF

FIXED POINT SUBTRACTION

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

SSQ Subtract Stored from Q 156 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q) - C(Y) \rightarrow C(Y)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y) = 0$, then ON; otherwise OFF

Negative If $C(Y)0 = 1$, then ON; otherwise OFF

Overflow If range of Y is exceeded, then ON; otherwise OFF

Carry If a carry out of Y0 is generated, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

SSXn Subtract Stored from Xn 14n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, \text{ or } 7$ as determined by operation code
 $C(Xn) - C(Y)0,17 \rightarrow C(Y)0,17$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y)0,17 = 0$, then ON; otherwise OFF

Negative If $C(Y)0 = 1$, then ON; otherwise OFF

Overflow If range of Y0,17 exceeded, then ON; otherwise OFF

Carry If a carry out of Y0 is generated, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

SWCA Subtract with Carry from A 171 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Carry indicator ON, then $C(A) - C(Y) \rightarrow C(A)$
 If Carry indicator OFF, then $C(A) - C(Y) - 1 \rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(A) = 0$, then ON; otherwise OFF
 Negative If $C(A)0 = 1$, then ON; otherwise OFF
 Overflow If range of A is exceeded, then ON; otherwise OFF
 Carry If a carry out of AD is generated, then ON; otherwise OFF

NOTES: The SWCA instruction is identical to the SBA instruction with the exception that when the Carry indicator is OFF at the beginning of the instruction, +1 is subtracted from the difference of C(A) minus C(Y). The SWCA instruction treats the Carry indicator as the complement of a borrow indicator; due to the implementation of negative numbers in two's complement form.

SWCQ Subtract with Carry from Q 172 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Carry indicator ON, then $C(Q) - C(Y) \rightarrow C(Q)$
 If Carry indicator OFF, then $C(Q) - C(Y) - 1 \rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Q) = 0$, then ON; otherwise OFF
 Negative If $C(Q)0 = 1$, then ON; otherwise OFF
 Overflow If range of Q is exceeded, then ON; otherwise OFF
 Carry If a carry out of Q0 is generated, then ON; otherwise OFF

FIXED POINT SUBTRACTION

NOTES:

The SWCQ instruction is identical to the SBQ instruction with the exception that when the Carry indicator is OFF at the beginning of the instruction, +1 is subtracted from the difference of C(Q) minus C(Y). The SWCQ instruction treats the Carry indicator as the complement of a borrow indicator; due to the implementation of negative numbers in two's complement form.

Fixed-Point-Multiplication

MPF Multiply Fraction 401 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(A) x C(Y) -> C(AQ), left adjusted

MODIFICATIONS: All except CI, SC, SCR

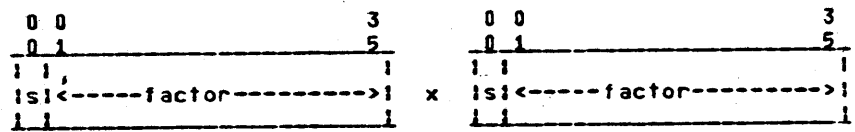
INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)D = 1, then ON; otherwise OFF

Overflow If range of AQ is exceeded, then ON; otherwise OFF

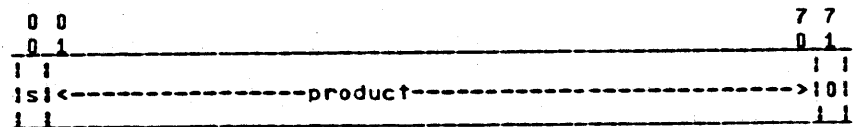
NOTES: Two 36-bit fractional factors (including sign) are multiplied to form a 71-bit fractional product (including sign), which is stored left-adjusted in the AQ-register. AQ71 contains a zero. Overflow can occur only in the case of A and Y containing all ones and the result exceeding the combined AQ-register.



A Register

Main Store Location Y

yielding



Combined AQ Register

MPY Multiply Integer 402 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Q) x C(Y) -> C(AQ), right adjusted

FIXED POINT MULTIPLICATION

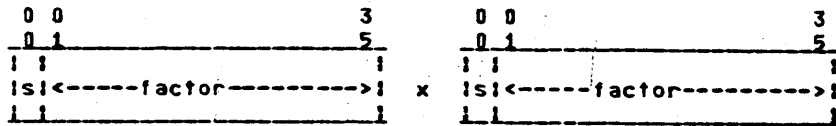
MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ) = 1, then ON; otherwise OFF

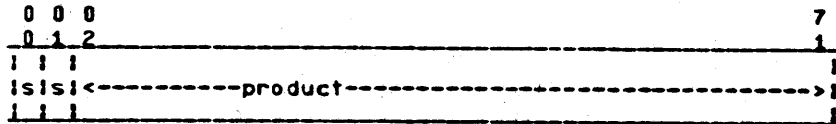
NOTES: Two 36-bit integer factors (including sign) are multiplied to form a 71-bit integer product (including sign), which is stored in AQ, right-adjusted. AQ0 is filled with an "extended sign bit".



Q Register

Main Store Location Y

yielding



Combined AQ Register

In the case of $(-2^{35}) \times (-2^{35}) = +2^{70}$, AQ1 is used to represent the product rather than the sign. No overflow can occur.

Fixed-Point-Division

DIV Divide Integer 506 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

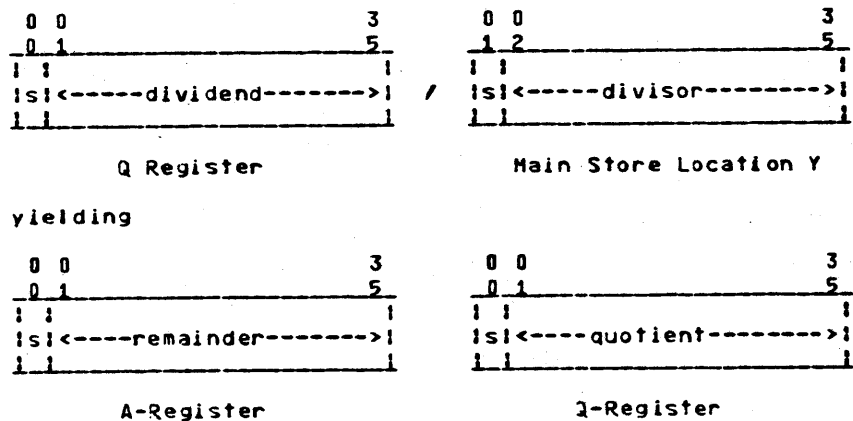
SUMMARY: C(Q) / C(Y) integer quotient -> C(Q)
integer remainder -> C(A)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

	<u>If division takes place:</u>	<u>If no division takes place:</u>
Zero	If C(Q) = 0, then ON; otherwise OFF	If divisor = 0, then ON; otherwise OFF
Negative	If C(Q)0 = 1, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF

NOTES: A 36-bit integer dividend (including sign) is divided by a 36-bit integer divisor (including sign) to form a 36-bit integer quotient (including sign) and a 36-bit integer remainder (including sign). The remainder sign is equal to the dividend sign unless the remainder is zero.



If the dividend = -2**35 and the divisor = -1 or if the divisor = 0, then division does not take place. Instead, a Divide Check fault occurs, C(Q) contains the dividend magnitude, and the Negative indicator reflects the dividend sign.

FIXED POINT DIVISION

DVF

Divide Fraction

507 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

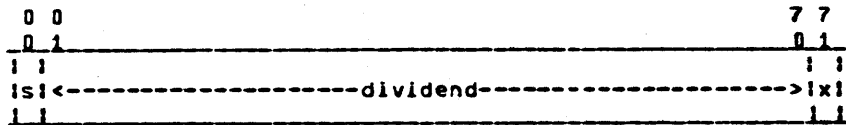
SUMMARY: C(AQ) / C(Y) fractional quotient -> C(A)
fractional remainder -> C(Q)

MODIFICATIONS: All

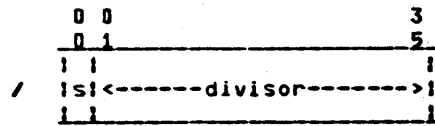
INDICATORS: (Indicators not listed are not affected)

	<u>If division takes place:</u>	<u>If no division takes place:</u>
Zero	If C(A) = 0, then ON; otherwise OFF	If divisor = 0, then ON; otherwise OFF
Negative	If C(A)0 = 1, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF

NOTES: A 71-bit fractional dividend (including sign) is divided by a 36-bit fractional divisor yielding a 36-bit fractional quotient (including sign) and a 36-bit fractional remainder (including sign). C(AQ)71 is ignored; bit position 35 of the remainder corresponds to bit position 70 of the dividend. The remainder sign is equal to the dividend sign unless the remainder is zero.

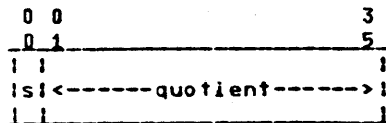


Combined AQ-Register

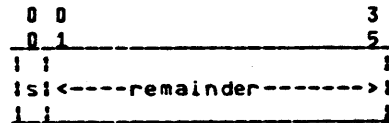


Main Store Location Y

yielding



A-Register



Q-Register

FIXED POINT DIVISION

If $|dividend| \geq |divisor|$ or if the divisor = 0, division does not take place. Instead, a Divide Check Fault occurs, C(AQ) contains the dividend magnitude in absolute, and the Negative indicator reflects the dividend sign.

Fixed-Point-Comparison

CMG Compare Magnitude 405 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $IC(A)_i \geq IC(Y)_i$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $IC(A)_i = IC(Y)_i$, then ON; otherwise OFF

Negative If $IC(A)_i < IC(Y)_i$, then ON; otherwise OFF

CMK Compare Masked 211 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 35$

$$C(Z)_i = \bar{C}(Q)_i \& (C(A)_i \circ C(Y)_i)$$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Z) = 0$, then ON; otherwise OFF

Negative If $C(Z)_0 = 1$, then ON; otherwise OFF

NOTES: The CMK instruction compares the contents of bit positions of A and Y for identity that are not masked by a 1 in the corresponding bit position of Q.

The Zero indicator is set ON if the comparison is successful for all bit positions; i.e., if for all $i = 0, 1, \dots, 35$ there is either: $C(A)_i = C(Y)_i$ (the identical case) or $C(Q)_i = 1$ (the masked case); otherwise, Zero

Indicator is set OFF.

The Negative Indicator is set ON if the comparison is unsuccessful for bit position 0; i.e., if $C(A)_0 \circ C(Y)_0$ (they are nonidentical) as well as $C(Q)_0 = 0$ (they are unmasked); otherwise, Negative indicator is set OFF.

FIXED POINT COMPARISON

CMPA Compare with A 115 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(A) :: C(Y)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

The Zero (Z), Negative (N), and Carry (C) indicators are set as follows.

Algebraic Comparison (Signed Binary Operands)

Z	N	C	Relation	Sign
0	0	0	C(A) > C(Y)	C(A)0 = 0, C(Y)0 = 1
0	0	1	C(A) > C(Y)	
1	0	1	C(A) = C(Y)	C(A)0 = C(Y)0
0	1	0	C(A) < C(Y)	
0	1	1	C(A) < C(Y)	C(A)0 = 1, C(Y)0 = 0

Logical Comparison (Unsigned Positive Binary Operands)

Z	C	Relation
0	0	C(A) < C(Y)
1	1	C(A) = C(Y)
0	1	C(A) > C(Y)

CMPAQ Compare with AQ 117 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(AQ) :: C(Y-pair)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

The Zero (Z), Negative (N), and Carry (C) indicators are set as follows.

Algebraic Comparison (Signed Binary Operands)

Z	N	C	Relation	Sign
0	0	0	$C(AQ) > C(Y\text{-pair})$	$C(AQ)0 = 0, C(Y\text{-pair})0 = 1$
0	0	1	$C(AQ) > C(Y\text{-pair})$	
1	0	1	$C(AQ) = C(Y\text{-pair})$	$C(AQ)0 = C(Y\text{-pair})0$
0	1	0	$C(AQ) < C(Y\text{-pair})$	
0	1	1	$C(AQ) < C(Y\text{-pair})$	$C(AQ)0 = 1, C(Y\text{-pair})0 = 0$

Logical Comparison (Unsigned Positive Binary Operands)

Z	C	Relation
0	0	$C(AQ) < C(Y\text{-pair})$
1	1	$C(AQ) = C(Y\text{-pair})$
0	1	$C(AQ) > C(Y\text{-pair})$

CMPQ Compare with Q 116 (D)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q) :: C(Y)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

The Zero (Z), Negative (N), and Carry (C) indicators are set as follows.

FIXED POINT COMPARISON

Algebraic Comparison (Signed Binary Operands)

Z	N	C	Relation	Sign
0	0	0	$C(Q) > C(Y)$	$C(Q)0 = 0, C(Y)0 = 1$
0	0	1	$C(Q) > C(Y)$	
1	0	1	$C(Q) = C(Y)$	$C(Q)0 = C(Y)0$
0	1	0	$C(Q) < C(Y)$	
0	1	1	$C(Q) < C(Y)$	$C(Q)0 = 1, C(Y)0 = 0$

Logical Comparison (Unsigned Positive Binary Operands)

Z	C	Relation
0	0	$C(Q) < C(Y)$
1	1	$C(Q) = C(Y)$
0	1	$C(Q) > C(Y)$

CMPXn Compare with Xn 10n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
 $C(Xn) :: C(Y)0,17$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

The Zero (Z), Negative (N), and Carry (C) indicators are set as follows.

Algebraic Comparison (Signed Binary Operands)

Z	N	C	Relation	Sign
0	0	0	$C(Xn) > C(Y)0,17$	$C(Xn)0 = 0, C(Y)0 = 1$
0	0	1	$C(Xn) > C(Y)0,17$	
0	1	0	$C(Xn) = C(Y)0,17$	$C(Xn)0 = C(Y)0$
1	0	1	$C(Xn) < C(Y)0,17$	
0	1	1	$C(Xn) < C(Y)0,17$	$C(Xn)0 = 1, C(Y)0 = 0$

Logical Comparison (Unsigned Positive Binary Operands)

Z	C	Relation
0	0	$C(X_n) < C(Y)_{0,17}$
1	1	$C(X_n) = C(Y)_{0,17}$
0	1	$C(X_n) > C(Y)_{0,17}$

CWL Compare with Limits 111 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Y) :: \text{closed interval } [C(A);C(Q)]$
 $C(Y) :: C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(A) \leq C(Y) \leq C(Q)$ or $C(A) \geq C(Y) \geq C(Q)$, then ON; otherwise OFF.

The Negative (N) and Carry (C) indicators are set as follows.

N	C	Relation	Sign
0	0	$C(Q) > C(Y)$	$C(Q)_0 = 0, C(Y)_0 = 1$
0	1	$C(Q) \geq C(Y)$	$C(Q)_0 = C(Y)_0$
1	0	$C(Q) < C(Y)$	$C(Q)_0 = C(Y)_0$
1	1	$C(Q) < C(Y)$	$C(Q)_0 = 1, C(Y)_0 = 0$

NOTES: The CWL instruction tests the value of C(Y) to determine if it is within the range of values set by C(A) and C(Q). The comparison of C(Y) with C(Q) locates C(Y) with respect to the interval if C(Y) is not contained within the interval.

FIXED POINT MISCELLANEOUS

Fixed-Point-Miscellaneous

SZN Set Zero and Negative Indicators 234 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Set indicators according to C(Y)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If C(Y) = 0, then ON; otherwise OFF

Negative If C(Y)0 = 1, then ON; otherwise OFF

SZNC Set Zero and Negative Indicators and Clear 214 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Set indicators according to C(Y)

00...0 => C(Y)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(Y) = 0, then ON, otherwise OFF

Negative If C(Y)0 = 1, then ON; otherwise OFF

BOOLEAN OPERATION INSTRUCTIONS

Boolean AND

ANA AND to A 375 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(A)_i & C(Y)_i → C(A)_i for i = (0, 1, ..., 35)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

 Zero If C(A) = 0, then ON; otherwise OFF

 Negative If C(A)₀ = 1, then ON; otherwise OFF

ANAQ AND to AQ 377 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(AQ)_i & C(Y-pair)_i → C(AQ)_i for i = (0, 1, ..., 71)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

 Zero If C(AQ) = 0, then ON; otherwise OFF

 Negative If C(AQ)₀ = 1, then ON; otherwise OFF

ANQ AND to Q 376 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Q)_i & C(Y)_i → C(Q)_i for i = (0, 1, ..., 35)

MODIFICATIONS: All

BOOLEAN AND

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Q) = 0$, then ON; otherwise OFF

Negative If $C(Q) = 1$, then ON; otherwise OFF

ANSA AND to Storage A 355 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A)_i \& C(Y)_i \rightarrow C(Y)_i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y) = 0$, then ON; otherwise OFF

Negative If $C(Y) = 1$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

ANSQ AND to Storage Q 356 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q)_i \& C(Y)_i \rightarrow C(Y)_i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y) = 0$, then ON; otherwise OFF

Negative If $C(Y) = 1$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

BOOLEAN OR

Boolean OR

ORA OR to A 275 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A)_i \vee C(Y)_i \rightarrow C(A)_i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(A) = 0$, then ON; otherwise OFF

Negative If $C(A)_0 = 1$, then ON; otherwise OFF

ORAQ OR to AQ 277 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(AQ)_i \vee C(Y\text{-pair})_i \rightarrow C(AQ)_i$ for $i = (0, 1, \dots, 71)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF

Negative If $C(AQ)_0 = 1$, then ON; otherwise OFF

ORQ OR to Q 276 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q)_i \vee C(Y)_i \rightarrow C(Q)_i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Q) = 0$, then ON; otherwise OFF

Negative If $C(Q)0 = 1$, then ON; otherwise OFF

ORSA OR to Storage A 255 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A)i \mid C(Y)i \rightarrow C(Y)i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y) = 0$, then ON; otherwise OFF

Negative If $C(Y)i = 1$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

ORSQ^{*} OR to Storage Q 256 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q)i \mid C(Y)i \rightarrow C(Y)$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y) = 0$, then ON; otherwise OFF

Negative If $C(Y)0 = 1$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

Boolean Exclusive OR

ERA EXCLUSIVE OR to A 675 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A)_i \oplus C(Y)_i \rightarrow C(A)_i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(A) = 0$, then ON; otherwise OFF

Negative If $C(A) = 1$, then ON; otherwise OFF

ERAQ EXCLUSIVE OR to AQ 677 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(AQ)_i \oplus C(Y\text{-pair})_i \rightarrow C(AQ)_i$ for $i = (0, 1, \dots, 71)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF

Negative If $C(AQ) = 1$, then ON; otherwise OFF

ERQ EXCLUSIVE OR to Q 676 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q)_i \oplus C(Y)_i \rightarrow C(Q)_i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Q) = 0$, then ON; otherwise OFF

BOOLEAN EXCLUSIVE OR

Negative If $C(Q)0 = 1$, then ON; otherwise OFF

ERSA EXCLUSIVE OR to Storage A 655 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(A)i \oplus C(Y)i \rightarrow C(Y)i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y) = 0$, then ON; otherwise OFF

Negative If $C(Y)0 = 1$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

ERSQ EXCLUSIVE OR to Storage Q 656 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Q)i \oplus C(Y)i \rightarrow C(Y)i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y) = 0$, then ON; otherwise OFF

Negative If $C(Y)0 = 1$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

ERSXn EXCLUSIVE OR to Storage Xn 64n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
 $C(Xn)_i \oplus C(Y)_i \rightarrow C(Y)_i$ for i = (0, 1, ..., 17)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y)_{0,17} = 0$, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

ERXn EXCLUSIVE OR to Xn 66n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
 $C(Xn)_i \oplus C(Y)_i \rightarrow C(Xn)_i$ for i = (0, 1, ..., 17)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Xn) = 0$, then ON; otherwise OFF

Negative If $C(Xn)_0 = 1$, then ON; otherwise OFF

Boolean Comparative AND

CANA Comparative AND with A 315 (0)

 FORMAT: Basic Instruction Format (See Figure 2-1).

 SUMMARY: $C(Z)_i = C(A)_i \& C(Y)_i$ for $i = (0, 1, \dots, 35)$

 MODIFICATIONS: All

 INDICATORS: (Indicators not listed are not affected)

 ZERO If $C(Z) = 0$, then ON; otherwise OFF

 Negative If $C(Z)_0 = 1$, then ON; otherwise OFF

CANAQ Comparative AND with AQ 317 (0)

 FORMAT: Basic Instruction Format (See Figure 2-1).

 SUMMARY: $C(Z)_i = C(AQ)_i \& C(Y\text{-pair})_i$ for $i = (0, 1, \dots, 71)$

 MODIFICATIONS: All except DU, DL, CI, SC, SCR

 INDICATORS: (Indicators not listed are not affected)

 Zero If $C(Z) = 0$, then ON; otherwise OFF

 Negative If $C(Z)_0 = 1$, then ON; otherwise OFF

CANQ Comparative AND with Q 316 (0)

 FORMAT: Basic Instruction Format (See Figure 2-1).

 SUMMARY: $C(Z)_i = C(Q)_i \& C(Y)_i$ for $i = (0, 1, \dots, 35)$

 MODIFICATIONS: All

 INDICATORS: (Indicators not listed are not affected)

 ZERO If $C(Z) = 0$, then ON; otherwise OFF

Negative If $C(Z)0 = 1$, then ON; otherwise OFF

CANXn Comparative AND with Xn 30n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, \text{or } 7$ as determined by operation code
 $C(Z)i = C(Xn)i \ \& \ C(Y)i$ for $i = (0, 1, \dots, 17)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Z) = 0$, then ON; otherwise OFF

Negative If $C(Z)0 = 1$, then ON; otherwise OFF

BOOLEAN COMPARATIVE NOT

CNA A Comparative NOT with A 215 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Z)_i = C(A)_i \& \sim C(Y)_i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

 Zero If $C(Z) = 0$, then ON; otherwise OFF

 Negative If $C(Z)_0 = 1$, then ON; otherwise OFF

CNA AQ Comparative NOT with AQ 217 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Z)_i = C(AQ)_i \& \sim C(Y\text{-pair})_i$ for $i = (0, 1, \dots, 71)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

 Zero If $C(Z) = 0$, then ON; otherwise OFF

 Negative If $C(Z)_0 = 1$, then ON; otherwise OFF

CNA Q Comparative NOT with Q 216 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Z)_i = C(Q)_i \& \sim C(Y)_i$ for $i = (0, 1, \dots, 35)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

 Zero If $C(Z) = 0$, then ON; otherwise OFF

 Negative If $C(Z)_0 = 1$, then ON; otherwise OFF

CNAXn Comparative NOT with Xn 20n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
 $C(Z)_i = C(X)_i \& \sim C(Y)_i$ for i = (0, 1, ..., 17)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(Z) = 0, then ON; otherwise OFF

Negative If C(Z)0 = 1, then ON; otherwise OFF

FLOATING POINT DATA MOVEMENT LOAD

FLOATING POINT ARITHMETIC INSTRUCTIONS

Floating Point Data Movement Loads

DFLO Double Precision Floating Load 433 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y-pair)0,7 → C(E)
C(Y-pair)8,71 → C(AQ)0,63
00...0 → C(AQ)64,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

FLD Floating Load 431 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y)0,7 → C(E)
C(Y)8,35 → C(AQ)0,27
00...0 → C(AQ)30,71

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative IF C(AQ)0 = 1, then ON; otherwise OFF

~Floating Point Data Movement Store~

DFST Double Precision Floating Store 457 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(E) -> C(Y-pair)0,7
 C(AQ)0,63 -> C(Y-pair)8,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

DFSTR Double Precision Floating Store Rounded 472 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(EAQ) rounded -> C(Y-pair)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

 Zero If C(Y-pair) = floating point 0, then ON; otherwise OFF

 Negative If C(Y-pair)8 = 1, then ON; otherwise OFF

 Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

 Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

NOTES: The DFSTR instruction performs a double precision true round and normalization on C(EAQ) as it is stored.

The definition of true round is located under the description of the Floating Round (FRD) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

Except for the precision of the stored result, the DFSTR instruction is identical to the FSTR instruction.

FLOATING POINT DATA MOVEMENT STORE

Attempted repetition with RPL causes an Illegal Procedure Fault.

FST Floating Store 455 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(E) -> C(Y)0,7
C(A)0,27 -> C(Y)8,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

FSTR Floating Store Rounded 470 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(EAQ) rounded -> C(Y)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(Y) = floating point 0, then ON; otherwise OFF
Negative If C(Y)8 = 1, then ON; otherwise OFF
Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF
Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

NOTES: The FSTR instruction performs a true round and normalization on C(EAQ) as it is stored.
The definition of true round is located under the description of the Floating Round (FRD) instruction.
The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

FLOATING POINT DATA MOVEMENT STORE

Steps in the execution may be thought of as follows:

Execute FNO

Execute FST

Restore C(EAQ) to original values.

Attempted repetition with RPL causes an Illegal Procedure Fault.

FLOATING POINT ADDITION

Floating Point Addition

DFAD Double Precision Floating Add 477 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) + C(Y\text{-pair})$ normalized $\rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF

Negative If $C(AQ)0 = 1$, then ON; otherwise OFF

Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: The DFAD instruction may be thought of as a Double Precision Unnormalized Floating Add (DUFA) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

DUFA Double Precision Unnormalized Floating Add 437 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) + C(Y\text{-pair}) \rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

FLOATING POINT ADDITION

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: Except for the precision of the mantissa of the operand from main store, the DUFA instruction is identical to the UFA instruction.

FAD Floating Add 475 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(EAQ) + C(Y) normalized -> C(EAQ)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: The FAD instruction may be thought of as an Unnormalized Floating Add (UFA) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

FLOATING POINT ADDITION

UFA Unnormalized Floating Add 435 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) + C(Y) \rightarrow C(EAQ)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF

Negative If $C(AQ)0 = 1$, then ON; otherwise OFF

Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: The UFA instruction is executed as follows:

The mantissas are aligned by shifting the mantissa of the operand having the algebraically smaller exponent to the right the number of places equal to the absolute value of the difference in the two exponents. Bits shifted beyond the bit position equivalent to AQ71 are lost.

The algebraically larger exponent replaces C(E).

The sum of the mantissas replaces C(AQ).

If an overflow occurs during addition, then;

C(AQ) are shifted one place to the right.

C(AQ)0 is inverted to restore the sign.

C(E) is increased by one.

Floating Point Subtraction

DFSB Double Precision Floating Subtract 577 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) - C(Y\text{-pair})$ normalized $\rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

- Zero If $C(AQ) = 0$, then ON; otherwise OFF
- Negative If $C(AQ)0 = 1$, then ON; otherwise OFF
- Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF
- Exponent Underflow If exponent is less than -128, then ON; otherwise OFF
- Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: The DFSB instruction is identical to the Double Precision Floating Add (DFAD) instruction with the exception that the 2's complement of the mantissa of the operand from main store is used.

DUFS Double Precision Unnormalized Floating Subtract 537 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) - C(Y\text{-pair}) \rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

- Zero If $C(AQ) = 0$, then ON; otherwise OFF
- Negative If $C(AQ)0 = 1$, then ON; otherwise OFF
- Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF
- Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

Carry If a carry out of AQ0 is generated, then ON; otherwise OFF

NOTES: The UFS instruction is identical to the Unnormalized Floating Add (UFA) instruction with the exception that the 2's complement of the mantissa of the operand from main store is used.

FLOATING POINT MULTIPLICATION

Floating Point Multiplication

DFMP Double Precision Floating Multiply 463 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) \times C(Y\text{-pair})$ normalized $\rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected).

Zero If $C(AQ) = 0$, then ON; otherwise OFF
Negative If $C(AQ)0 = 1$, then ON; otherwise OFF
Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF
Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

NOTES: The DFMP instruction may be thought of as a Double Precision Unnormalized Floating Multiply (DUFM) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

DUFM Double Precision Unnormalized Floating Multiply 423 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) \times C(Y\text{-pair}) \rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF
Negative If $C(AQ)0 = 1$, then ON; otherwise OFF
Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF
Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

NOTES: Except for the precision of the mantissa of the operand from main store, the DUFM instruction is identical to the Unnormalized Floating Multiply (UFM) instruction.

FMP Floating Multiply 461 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) \times C(Y)$ normalized $\rightarrow C(EAQ)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(AQ) = 0$, then ON; otherwise OFF

Negative If $C(AQ)D = 1$, then ON; otherwise OFF

Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

NOTES: The FMP instruction may be thought of as an Unnormalized Floating Multiply (UFM) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

UFM Unnormalized Floating Multiply 421 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(EAQ) \times C(Y) \rightarrow C(EAQ)$

MODIFICATIONS: All except CI, SC, SCR

FLOATING POINT MULTIPLICATION

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If C(AQ)0 = 1, then ON; otherwise OFF
Exponent Overflow	If exponent is greater than +127, then ON; otherwise OFF
Exponent Underflow	If exponent is less than -128, then ON; otherwise OFF

NOTES: The UFM instruction is executed as follows:

$C(E) + C(Y)0,7 \rightarrow C(E)$

$(C(AQ) \times C(Y)8,35)0,71 \rightarrow C(AQ)$

A normalization is performed only in the case of both factor mantissas being 100...0 which is the 2's complement approximation to the decimal value -1.0.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

~Floating Point Division~

DFDI Double Precision Floating Divide Inverted 527 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y-pair) / C(EAQ) -> C(EAQ)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

	<u>If division takes place:</u>	<u>If no division takes place:</u>
Zero	If C(AQ) = 0, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If C(AQ)0 = 1, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exponent Overflow	If exponent is greater than +127, then ON; otherwise OFF	
Exponent Underflow	If exponent is less than -128, then ON; otherwise OFF	

NOTES: Except for the interchange of the roles of the operands, the execution of the DFDI instruction is identical to the execution of the Double Precision Floating Divide (DFDV) instruction.

If the divisor mantissa C(AQ) is zero, the division does not take place. Instead, a Divide Check Fault occurs and all registers remain unchanged.

DFDV Double Precision Floating Divide 567 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(EAQ) / C(Y-pair) -> C(EAQ)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

FLOATING POINT DIVISION

INDICATORS:	(Indicators not listed are not affected)	
	<u>If division takes place:</u>	<u>If no division takes place:</u>
Zero	If C(AQ) = 0, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If C(AQ)0 = 1, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exponent Overflow	If exponent is greater than +127, then ON; otherwise OFF	
Exponent Underflow	If exponent is less than -128, then ON; otherwise OFF	

NOTES: The DFDV instruction is executed as follows:

The dividend mantissa C(AQ) is shifted right and the dividend exponent C(E) increased accordingly until $|C(AQ)0,63| < |C(Y\text{-pair})8,71|$.

$C(E) - C(Y\text{-pair})0,7 \rightarrow C(E)$

$C(AQ) / C(Y\text{-pair})8,71 \rightarrow C(AQ)0,63$

$00...0 \rightarrow C(Q)64,71$

If the divisor mantissa C(Y-pair)8,71 is zero, the division does not take place. Instead, a Divide Check fault occurs, C(AQ) contains the dividend magnitude, and the Negative indicator reflects the dividend sign.

FDI Floating Divide Inverted 525 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y) / C(EAQ) -> C(EA)
00...0 -> C(Q)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

	<u>If division takes place:</u>	<u>If no division takes place:</u>
Zero	If C(A) = 0, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If C(A)D = 0, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exponent Overflow	If exponent is greater than +127, then ON; otherwise OFF	
Exponent Underflow	If exponent is less than -128, then ON; otherwise OFF	

NOTES: Except for the interchange of roles of the operands, the execution of the FDI instruction is identical to the execution of the Floating Divide (FDV) instruction.

If the divisor mantissa C(AQ) is zero, the division does not take place. Instead, a Divide-Check fault occurs and all the registers remain unchanged.

FDV Floating Divide 565 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY C(EAQ) / C(Y) -> C(EA)
00...0 -> C(Q)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

	<u>If division takes place:</u>	<u>If no division takes place:</u>
Zero	If C(A) = 0, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If C(A)D = 1, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exponent Overflow	If exponent is greater than +127, then ON; otherwise OFF	
Exponent Underflow	If exponent is less than -128, then ON; otherwise OFF	

FLOATING POINT DIVISION

NOTES:

The FDV instruction is executed as follows:

The dividend mantissa C(AQ) is shifted right and the dividend exponent C(E) increased accordingly until $|C(AQ)_{0,27}| < |C(Y)_{8,35}|$.

$C(E) - C(Y)_{0,7} \rightarrow C(E)$

$C(AQ) / C(Y)_{8,35} \rightarrow C(A)$

$00...0 \rightarrow C(Q)$

If the divisor mantissa C(Y)_{8,35} is zero, the division does not take place. Instead, a Divide Check fault occurs, C(AQ) contains the dividend magnitude, and the Negative indicator reflects the dividend sign.

Floating Point Negate

FNEG Floating Negate 513 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $-C(AQ)$ normalized $\rightarrow C(AQ)$

MODIFICATIONS: All, but none affect instruction execution.

INDICATORS: (Indicators not listed are not affected)

- Zero If $C(AQ) = 0$, then ON; otherwise OFF
- Negative If $C(AQ) = 1$, then ON; otherwise OFF
- Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF
- Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

NOTES: This instruction changes the number in $C(EAQ)$ to its normalized negative (if $C(AQ) \neq 0$). The operation is executed by first forming the two's complement of $C(AQ)$, and then normalizing $C(EAQ)$.

Even if originally $C(EAQ)$ were normalized, an exponent overflow can still occur, namely when $C(E) = +127$ and $C(AQ) = 100...0$ which is the 2's complement approximation for the decimal value -1.0.

The definition of normalization may be found under the description of the Floating Normalize (FNO) instruction.

Attempted repetition with RPL causes an Illegal Procedure Fault.

FLOATING POINT NORMALIZE

Floating Point Normalize

FNO Floating Normalize 573 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(EAQ) normalized -> C(EAQ)

MODIFICATIONS: All, but none affect instruction execution.

INDICATORS: (Indicators not listed are not affected)

Zero	If C(EAQ) = floating point 0, then ON; otherwise OFF
Negative	If C(AQ)0 = 1, then ON; otherwise OFF
Exponent Overflow	If exponent is greater than +127, then ON; otherwise OFF
Exponent Underflow	If exponent is less than -128, then ON otherwise OFF
Overflow	Set OFF

NOTES: The FNO instruction normalizes the number in C(EAQ) if C(AQ) ≠ 0 and the Overflow indicator is OFF.

A normalized floating number is defined as one whose mantissa lies in the interval [0.5,1.0] such that

$$0.5 \leq |C(AQ)| < 1.0$$

which, in turn, requires that C(AQ)0 ≠ C(AQ)1.

If the Overflow indicator is ON, then C(AQ) is shifted one place to the right, C(AQ)0 is inverted to reconstitute the actual sign, and the Overflow indicator is set OFF.

Normalization is performed by shifting C(AQ)1,71 one place to the left and reducing C(E) by 1, repeatedly, until the conditions for C(AQ)0 and C(AQ)1 are met. Bits shifted out of AQ1 are lost.

If C(AQ) = 0, then C(E) is set to -128 and the Zero indicator is set ON.

The FNO instruction can be used to correct overflows that occur with fixed point numbers.

Attempted repetition with RPL causes an Illegal Procedure Fault.

~Floating Point Round@

DFRD Double Precision Floating Round 473 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(EAQ) rounded to 64 bits -> C(EAQ)

MODIFICATIONS: All, but none affect instruction execution.

INDICATORS: (Indicators not listed are not affected)

Zero If C(EAQ) = floating point 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

NOTES: The DFRD instruction is identical to the Floating Round (FRD) instruction except that the rounding constant used is (11...1)65,71 instead of (11...1)29,71.

Attempted repetition with RPL causes an Illegal Procedure Fault.

FRD Floating Round 471 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(EAQ) rounded to 28 bits -> C(EAQ)

MODIFICATIONS: All, but none affect instruction execution.

INDICATORS: (Indicators not listed are not affected)

Zero If C(EAQ) = floating point 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1 then ON; otherwise OFF

Exponent Overflow If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow If exponent is less than -128, then ON; otherwise OFF

FLOATING POINT ROUND

NOTES:

If $C(AQ) \neq 0$, the FRD instruction performs a true round to a precision of 28 bits and a normalization on $C(EAQ)$.

A true round is a rounding operation such that the sum of the result of applying the operation to two numbers of equal magnitude but opposite sign is exactly zero.

The FRD instruction is executed as follows:

$C(AQ) + (11\dots1)_{29,71} \rightarrow C(AQ)$

If $C(AQ)_{71} = 0$, then a carry is added at AQ_{71}

If overflow occurs, $C(AQ)$ is shifted one place to the right and $C(E)$ is increased by 1.

If overflow does not occur, $C(EAQ)$ is normalized.

If $C(AQ) = 0$, $C(E)$ is set to -128 and the Zero indicator is set ON.

Attempted repetition with RPL causes an Illegal Procedure Fault.

~Floating~Point~Compare~

DFCMG Double Precision Floating Compare Magnitude 427 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: IC(E,AQ0,63) :: IC(Y-pair)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

 Zero If IC(E,AQ0,63) = IC(Y-pair), then ON; otherwise OFF

 Negative If IC(E,AQ0,63) < IC(Y-pair), then ON; otherwise OFF

NOTES: The DFCMG instruction is identical to the Double Precision Floating Compare (DFCMP) instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

DFCMP Double Precision Floating Compare 517 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(E,AQ0,63) :: C(Y-pair)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

 Zero If C(E,AQ0,63) = C(Y-pair), then ON; otherwise OFF

 Negative If C(E,AQ0,63) < C(Y-pair), then ON; otherwise OFF

NOTES: The DFCMP instruction is identical to the Floating Compare (FCMP) instruction except for the precision of the mantissas actually compared.

FLOATING POINT COMPARE

FCMG Floating Compare Magnitude 425 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $|C(E, A_{Q0}, 27)| :: |C(Y)|$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $|C(E, A_{Q0}, 27)| = |C(Y)|$, then ON; otherwise OFF

Negative If $|C(E, A_{Q0}, 27)| < |C(Y)|$, then ON; otherwise OFF

NOTES: The FCMG instruction is identical to the Floating Compare (FCMP) instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

FCMP Floating Compare 515 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(E, A_{Q0}, 27) :: C(Y)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If $C(E, A_{Q0}, 27) = C(Y)$, then ON; otherwise OFF

Negative If $C(E, A_{Q0}, 27) < C(Y)$, then ON; otherwise OFF

NOTES: The FCMP instruction is executed as follows:

The mantissas are aligned by shifting the mantissa of the operand with the algebraically smaller exponent to the right the number of places equal to the difference in the two exponents.

The aligned mantissas are compared and the indicators set accordingly.

FLOATING POINT MISCELLANEOUS

INDICATORS: (Indicators not listed are not affected)

Zero Set OFF

Negative Set OFF

STE Store Exponent

456 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(E) -> C(Y)0,7
00...0 -> C(Y)8,17

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

TRANSFER INSTRUCTIONS

CALL6 Call (Using PR6 and PR7) 713 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY:

```

If C(TPR.TRR) < C(PPR.PRR) then
  C(DSBR.STACK) !! C(TPR.TRR) -> C(PR7.SNR)

If C(TPR.TRR) = C(PPR.PRR) then
  C(PR6.SNR) -> C(PR7.SNR)

C(TPR.TRR) -> C(PR7.RNR)

If C(TPR.TRR) = 0 then
  C(SDW.P) -> C(PPR.P);
  otherwise 0 -> C(PPR.P)

00...0 -> C(PR7.WORDNO)
00...0 -> C(PR7.BITNO)
C(TPR.TRR) -> C(PPR.PRR)
C(TPR.TSR) -> C(PPR.PSR)
C(TPR.CA) -> C(PPR.IC)

```

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES:

If $C(TPR.TRR) > C(PPR.PRR)$, an Access Violation Fault, Outward Call, occurs and the CALL6 instruction is not executed.

If the CALL6 instruction is executed with the Processor in Absolute Mode with bit 29 of the instruction word equal to 0 and without indirection through an ITP or ITS pair, then...

the Appending Mode is entered for the address preparation of the CALL6 operand address and is retained if the instruction executes successfully, and...

the Effective Segment Number generated for the SDW

fetch and subsequent loading into $C(TPR.TSR)$ is equal to $C(PPR.PSR)$ and may be undefined in Absolute Mode, and...

the Effective Ring Number loaded into $C(TPR.TRR)$ prior to the SDW fetch is equal to $C(PPR.PRR)$ (which is 0 in Absolute Mode) implying that the Access Violation checks for Outward Call and Bad Outward Call are ineffective and that an Access Violation, Out of Call Brackets will occur if $C(SDW.R1) \neq 0$.

RTCD

Return Control Double

610 (0)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

C(Y-pair)3,17 -> C(PPR.PSR)

Maximum of

C(Y-pair)18,20; C(TPR.TRR); C(SDW.R1) -> C(PPR.PRR)

C(Y-pair)36,53 -> C(PPR.IC)

If C(PPR.PRR) = 0 then C(SDW.P) -> C(PPR.P);
otherwise 0 -> C(PPR.P)

C(PPR.PRR) -> C(PRn.RNR) for n = (0, 1, ..., 7)

MODIFICATIONS:

All except DU, DL, CI, SC, SCR

INDICATORS:

None affected

NOTES:

The hardware assumes that C(Y)17 = 0; no check is made.

If an access violation occurs when fetching the SDW for location Y, the C(PPR.PSR) and C(PPR.PRR) are not altered.

If the RTCD instruction is executed with the Processor in Absolute Mode with bit 29 of the instruction word equal to 0 and without indirection through an ITP or ITS pair, then...

the Appending Mode is entered for the address preparation of the RTCD operand address and is retained if the instruction executes successfully, and...

the Effective Segment Number generated for the SDW fetch and subsequent loading into C(TPR.TSR) is equal to C(PPR.PSR) and may be undefined in Absolute Mode, and...

the Effective Ring Number loaded into C(TPR.TRR) prior to the SDW fetch is equal to C(PPR.PRR) (which is 0 in Absolute Mode) implying that control is always transferred into Ring 0.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TMI	Transfer on Minus	604 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	If Negative Indicator ON then C(TPR.CA) -> C(PPR.IC) C(TPR.TSR) -> C(PPR.PSR) otherwise, no change to C(PPR)	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	None affected	
NOTES:	Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	
THOZ	Transfer On Minus or Zero	604 (1)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	If Negative or Zero Indicator ON then C(TPR.CA) -> C(PPR.IC) C(TPR.TSR) -> C(PPR.PSR) otherwise, no change to C(PPR)	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	None affected	
NOTES:	Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	
TNC	Transfer on No Carry	602 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	If Carry Indicator OFF then C(TPR.CA) -> C(PPR.IC)	

TRANSFER

C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

TNZ Transfer On Not Zero 601 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Zero indicator OFF then
C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS: All except, DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

TOV Transfer On Overflow 617 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Overflow indicator ON then
C(TPR.CA) C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)

otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

	Overflow	Set OFF	
	NOTES:	Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	
TPL		Transfer on Plus	605 (0)
	FORMAT:	Basic Instruction Format (See Figure 2-1).	
	SUMMARY:	If Negative indicator OFF, then C(TPR.CA) -> C(PPR.IC) C(PTR.TSR) -> C(PPR.PSR) otherwise, no change to C(PPR)	
	MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
	INDICATORS:	None affected	
	NOTES:	Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	
TPNZ		Transfer on Plus and Nonzero	605 (1)
	FORMAT:	Basic Instruction Format (See Figure 2-1).	
	SUMMARY:	If Negative and Zero indicators are OFF then C(TPR.CA) -> C(PPR.IC) C(TPR.TSR) -> C(PPR.PSR) otherwise, no change to C(PPR)	
	MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
	INDICATORS:	None affected	
	NOTES:	Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	

TRANSFER

TRA	Transfer Unconditionally	710 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	C(TPR.CA) --> C(PPR.IC) C(TPR.TSR) --> C(PPR.PSR)	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	None affected	
NOTES:	Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	
TRC	Transfer on Carry	603 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	If Carry indicator ON then C(TPR.CA) C(PPR.IC) C(TPR.TSR) -> C(PPR.PSR) otherwise, no change to C(PPR)	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	None affected	
NOTES:	Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	
TRTF	Transfer on Truncation Indicator OFF	601 (1)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	If Truncation Indicator OFF then C(TPR.CA) -> C(PPR.IC) C(TPR.TSR) -> C(PPR.PSR) otherwise, no change to C(PPR)	

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TRTN Transfer on Truncation Indicator ON 600 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Truncation Indicator ON then

C(TPR.CA) -> C(PPR.IC)

C(TPR.TSR) -> C(PPR.PSR)

otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Truncation Set OFF

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TSP0	Transfer and Set PR0	270 (0)
TSP1	Transfer and Set PR1	271 (0)
TSP2	Transfer and Set PR2	272 (0)
TSP3	Transfer and Set PR3	273 (0)
TSP4	Transfer and Set PR4	670 (0)
TSP5	Transfer and Set PR5	671 (0)
TSP6	Transfer and Set PR6	672 (0)
TSP7	Transfer and Set PR7	673 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(PPR.PRR) -> C(PRn.RNR)

C(PPR.PSR) -> C(PRn.SNR)

C(PPR.IC) +1 -> C(PRn.WORDNO)

00...0 -> C(PRn.BITNO)

TRANSFER

C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TSS Transfer and Set Slave 715 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected (except as noted below)

NOTES: If the TSS instruction is executed with the Processor not in BAR mode, the Absolute indicator is set OFF, and the Not BAR Mode indicator is set OFF to signal that subsequent addressing is to be done in the BAR Mode. The Base Address Register (BAR) is used in the address preparation of the transfer, and the BAR will be used in address preparation for all subsequent instructions until a fault or interrupt occurs.

If the TSS instruction is executed with the Not BAR Mode Indicator already OFF, it functions as a Transfer (TR) instruction and no indicators are changed.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TSXn Transfer and Set Index Register Xn 70n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
C(PPR.IC) + 1 -> C(Xn)
C(TPR.CA) -> C(PPR.IC)

C(TPR.TSR) -> C(PPR.PSR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TTF Transfer on Tally Runout Indicator OFF 607 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Tally Runout Indicator OFF then

C(TPR.CA) -> C(PPR.IC)

C(TPR.TSR) -> C(PPR.PSR)

otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TTN Transfer on Tally Runout Indicator ON 606 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Tally Runout Indicator ON then

C(TPR.CA) -> C(PPR.IC)

C(TPR.TSR) -> C(PPR.PSR)

otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TRANSFER

TZE

Transfer On Zero

600 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Zero indicator ON then
C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

POINTER REGISTER INSTRUCTIONS

Pointer Register Data Movement Load

EASP0	Effective Address to Segment Number of PR0	311 (0)
EASP1	Effective Address to Segment Number of PR1	310 (1)
EASP2	Effective Address to Segment Number of PR2	313 (0)
EASP3	Effective Address to Segment Number of PR3	312 (1)
EASP4	Effective Address to Segment Number of PR4	331 (0)
EASP5	Effective Address to Segment Number of PR5	330 (1)
EASP6	Effective Address to Segment Number of PR6	333 (0)
EASP7	Effective Address to Segment Number of PR7	332 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
C(TPR.CA) -> C(PRn.SNR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

*NOTES: Attempted execution in BAR Mode causes an Illegal Procedure Fault.
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EAWP0	Effective Address to Word/Bit Number of PR0	310 (0)
EAWP1	Effective Address to Word/Bit Number of PR1	311 (1)
EAWP2	Effective Address to Word/Bit Number of PR2	312 (0)
EAWP3	Effective Address to Word/Bit Number of PR3	313 (1)
EAWP4	Effective Address to Word/Bit Number of PR4	330 (0)
EAWP5	Effective Address to Word/Bit Number of PR5	331 (1)
EAWP6	Effective Address to Word/Bit Number of PR6	332 (0)
EAWP7	Effective Address to Word/Bit Number of PR7	333 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(TPR.CA) -> C(PRn.WORDNO)

C(TPR.TBR) / 9 -> C(PRn.CHAR)

C(TPR.TBR) modulo 9 -> C(PRn.BITNO)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

POINTER REGISTER DATA MOVEMENT LOAD

INDICATORS: None affected

NOTES: Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EPBP0	Effective Pointer at Base to PR0	350 (1)
EPBP1	Effective Pointer at Base to PR1	351 (0)
EPBP2	Effective Pointer at Base to PR2	352 (1)
EPBP3	Effective Pointer at Base to PR3	353 (0)
EPBP4	Effective Pointer at Base to PR4	370 (1)
EPBP5	Effective Pointer at Base to PR5	371 (0)
EPBP6	Effective Pointer at Base to PR6	372 (1)
EPBP7	Effective Pointer at Base to PR7	373 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(TPR.TRR) -> C(PRn.RNR)

C(TPR.ISR) -> C(PRn.SNR)

00...0 -> C(PRn.WORDNO)

00 -> C(PRn.CHAR)

0000 -> C(PRn.BITNO)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EPP0	Effective Pointer to PR0	350 (0)
EPP1	Effective Pointer to PR1	351 (1)

EPP2	Effective Pointer to PR2	352 (0)
EPP3	Effective Pointer to PR3	353 (1)
EPP4	Effective Pointer to PR4	370 (0)
EPP5	Effective Pointer to PR5	371 (1)
EPP6	Effective Pointer to PR6	372 (0)
EPP7	Effective Pointer to PR7	373 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

POINTER REGISTER DATA MOVEMENT LOAD

SUMMARY: For $n = 0, 1, \dots, \text{or } 7$ as determined by operation code

C(TPR.TRR) \rightarrow C(PRn.RNR)

C(TPR.TSR) \rightarrow C(PRn.SNR)

C(TPR.CA) \rightarrow C(PRn.WORDNO)

C(TPR.TBR) / 9 \rightarrow C(PRn.CHAR)

C(TPR.TBR) modulo 9 \rightarrow C(PRn.BITNO)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LPRI Load Pointer Registers from ITS Pairs 173 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, 7$

Maximum of
C(Y+2n-pair)18,20; C(SDW.RI); C(TPR.TRR) \rightarrow C(PRn.RNR)

C(Y+2n-pair)3,17 \rightarrow C(PRn.SNR)

C(Y+2n-pair)36,53 \rightarrow C(PRn.WORDNO)

C(Y+2n-pair)57,62 / 9 \rightarrow C(PRn.CHAR)

C(Y+2n-pair)57,62 modulo 9 \rightarrow C(PRn.BITNO)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None Affected

NOTES: Starting at location Y, the contents of eight word pairs (in ITS pair format) replace the contents of Pointer Registers 0 through 7 as shown. The hardware assumes that Y14,17 = 0000 and addressing is incremented accordingly; no check is made.

Since C(TPR.TRR) and C(SDW.RI) are both equal to zero in Absolute mode, C(Y+2n-pair)18,20 are loaded into PRn.RNR in Absolute mode.

POINTER REGISTER DATA MOVEMENT LOAD

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LPRPn Load PRn Packed 76n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(TPR.TRR) -> C(PRn.RNR)

If C(Y)0,2 ≠ 11, then

C(Y)0,5 / 9 -> C(PRn.CHAR)

C(Y)0,5 modulo 9 -> C(PRn.BITNO);

otherwise, generate Command Fault

If C(Y)6,17 = 11...1, then 111 -> C(PRn.SNR)0,2

otherwise, 000 -> C(PRn.SNR)0,2

C(Y)6,17 -> C(PRn.SNR)3,14

C(Y)18,35 -> C(PRn.WORDNO)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Binary "1"s in C(Y)0,2 correspond to an illegal BITNO, that is, a bit position beyond the extent of C(Y). Detection of these bits causes a Command Fault.

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Pointer Register Data Movement Store

SPBP0	Store Segment Base Pointer of PR0	250 (1)
SPBP1	Store Segment Base Pointer of PR1	251 (0)
SPBP2	Store Segment Base Pointer of PR2	252 (1)
SPBP3	Store Segment Base Pointer of PR3	253 (0)
SPBP4	Store Segment Base Pointer of PR4	650 (1)
SPBP5	Store Segment Base Pointer of PR5	651 (0)
SPBP6	Store Segment Base Pointer of PR6	652 (1)
SPBP7	Store Segment Base Pointer of PR7	653 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(PRn.SNR) -> C(Y-pair)3,17

C(PRn.RNR) -> C(Y-pair)18,20

000 -> C(Y-pair)0,2

00...0 -> C(Y-pair)21,29

43 (octal) -> C(Y-pair)30,35

00...0 -> C(Y-pair)36,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes Y bit 17 = 0; no check is made.

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPRI Store Pointer Registers as ITS Pairs 254 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., 7

000 -> C(Y+2n-pair)0,2

C(PRn.SNR) -> C(Y+2n-pair)3,17

C(PRn.RNR) -> C(Y+2n-pair)18,20

00...0 -> C(Y+2n-pair)21,29

POINTER REGISTER DATA MOVEMENT STORE

43 (octal) -> C(Y+2n-pair)30,35
C(PRn.WORDNO) -> C(Y+2n-pair)36,53
000 -> C(Y+2n-pair)54,56
9 * C(PRn.CHAR) + C(PRn.BITNO) -> C(Y+2n-pair)57,62
00...0 -> C(Y+2n-pair)63,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Starting at location Y, the contents of Pointer Registers 0 through 7 replace the contents of eight word pairs (in ITS pair format). The hardware assumes Y bits 14 to 17 = 0000 and addressing is incremented accordingly; no check is made.

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPRI0	Store PR0 as ITS Pair	250 (0)
SPRI1	Store PR1 as ITS Pair	251 (1)
SPRI2	Store PR2 as ITS Pair	252 (0)
SPRI3	Store PR3 as ITS Pair	253 (1)
SPRI4	Store PR4 as ITS Pair	650 (0)
SPRI5	Store PR5 as ITS Pair	651 (1)
SPRI6	Store PR6 as ITS Pair	652 (0)
SPRI7	Store PR7 as ITS Pair	653 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

000 -> C(Y-pair)0,2
C(PRn.SNR) -> C(Y-pair)3,17
C(PRn.RNR) -> C(Y-pair)18,20
00...0 -> C(Y-pair)21,29

43 (octal) -> C(Y-pair)30,35
C(PRn.WORDNO) -> C(Y-pair)36,53
000 -> C(Y-pair)54,56
9 * C(PRn.CHAR) + C(PRn.BITNO) -> C(Y-pair)57,62

POINTER REGISTER DATA MOVEMENT STORE

00...0 -> C(Y-pair)63,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes Y bit 17 = 0; no check is made.
Attempted execution in BAR Mode causes an Illegal Procedure Fault.
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPRn Store PRn Packed 54n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
 $9 * C(PRn.CHAR) + C(PRn.BITNO) \rightarrow C(Y)0,5$
 $C(PRn.SNR)3,14 \rightarrow C(Y)6,17$
 $C(PRn.WORDNO) \rightarrow C(Y)18,35$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: If C(PRn.SNR)0,2 are nonzero, and C(PRn.SNR) \neq 11...1, then a Store Fault, Illegal Pointer, will occur and C(Y) will not be changed.
Attempted execution in BAR Mode causes an Illegal Procedure Fault.
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

POINTER REGISTER ADDRESS ARITHMETIC

Pointer Register Address Arithmetic

ADWP0	Add to Word Register of PR0	050 (0)
ADWP1	Add to Word Register of PR1	051 (0)
ADWP2	Add to Word Register of PR2	052 (0)
ADWP3	Add to Word Register of PR3	053 (0)
ADWP4	Add to Word Register of PR4	150 (0)
ADWP5	Add to Word Register of PR5	151 (0)
ADWP6	Add to Word Register of PR6	152 (0)
ADWP7	Add to Word Register of PR7	153 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, \text{ or } 7$ as determined by operation code

$C(Y)0,17 + C(PRn.WORDNO) \rightarrow C(PRn.WORDNO)$

$00 \rightarrow C(PRn.CHAR)$

$0000 \rightarrow C(PRn.BITNO)$

MODIFICATIONS: All except DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Pointer Register Miscellaneous

EPAQ	Effective Pointer to AQ Register	213 (0)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	00...0 -> C(AQ)0,2 C(TPR.TSR) -> C(AQ)3,17 00...0 -> C(AQ)18,32 C(TPR.TRR) -> C(AQ)33,35 C(TPR.CA) -> C(AQ)36,53 00...0 -> C(AQ)54,65 C(TPR.TBR) -> C(AQ)66,71	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	(Indicators not listed are not affected)	
Zero	If C(AQ) = 0, then ON; otherwise OFF	
NOTES:	Attempted execution in BAR Mode causes an Illegal Procedure Fault. Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	

CALENDAR CLOCK

MISCELLANEOUS INSTRUCTIONS

Calendar Clock

RCCL Read Calendar Clock 633 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: 00...0 -> C(AQ)0,19
C(Calendar Clock) -> C(AQ)20,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: C(TPR.CA)0,2 specify which Processor port (i.e., which System Controller) is to be used. The contents of the clock in the designated System Controller replace the contents of the AQ-register as shown.

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with PRT, RPD, or RPL causes an Illegal Procedure Fault.

~~Derails~~

DRL

Derail

002 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Causes a fault which fetches and executes, in Absolute Mode, the instruction pair at main store location C+14(octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

MODIFICATIONS: All, but none affect instruction execution

INDICATORS: None affected

NOTES: Except for the different constant used for fetching the instruction pair from main store, the DRL instruction is identical to the Master Mode Entry (MME) instruction.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EXECUTE

Execute

XEC

Execute

716 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Fetch and execute the instruction in C(Y)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The XEC instruction itself does not affect any indicator. However, the execution of the instruction from C(Y) may affect indicators.

If the execution of the instruction from C(Y) modifies C(PPR.IC), then a transfer of control occurs; otherwise, the next instruction to be executed is fetched from C(PPR.IC)+1.

To execute a Repeat Double (RPD) instruction, the XEC instruction must be in an odd location. The instruction pair repeated is that instruction pair at C(PPR.IC)+1, that is, the instruction pair immediately following the XEC instruction. C(PPR.IC) is adjusted during the execution of the repeated instruction pair so that the next instruction fetched for execution is from the first word following the repeated instruction pair.

EIS Multiword instructions may be executed but the required Data Descriptors must be located immediately after the XEC instruction, that is, starting at C(PPR.IC) + 1. C(PPR.IC) is adjusted during execution of the EIS Multiword instruction so that the next instruction fetched for execution is from the first word following the EIS Data Descriptors.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

XED Execute Double 717 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Fetch and execute the instruction pair at C(Y-pair)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES:

The XED instruction itself does not affect any indicator. However, the execution of the instruction pair from C(Y-pair) may affect indicators.

The even instruction from C(Y-pair) must not alter C(Y-pair)36,71, and must not be another XED instruction.

If the execution of the instruction pair from C(Y-pair) alters C(PPR.IC), then a transfer of control occurs; otherwise, the next instruction to be executed is fetched from C(PPR.IC)+1. If the even instruction from C(Y-pair) alters C(PPR.IC), then the transfer of control is effective immediately and the odd instruction is not executed.

To execute an instruction pair having a Repeat Double (RPD) instruction as the odd instruction, the XED must be located at an odd address. The instruction pair repeated is that instruction pair at C(PPR.IC) + 1, that is, the instruction pair immediately following the XED instruction. C(PPR.IC) is adjusted during the execution of the repeated instruction pair so the the next instruction fetched for execution is from the first word following the repeated instruction pair.

An attempt to execute an EIS Multiword instruction will cause an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an illegal Procedure Fault.

MASTER MODE ENTRY

Master Mode Entry

MME Master Mode Entry 001 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Causes a fault that fetches and executes, in Absolute Mode, the instruction pair at main store location C+4(octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

MODIFICATIONS: All, but none affect instruction execution

INDICATORS: None affected

NOTES: Execution of the MME instruction implies the following conditions:

During the execution of the MME instruction and the two instructions fetched, the Processor is temporarily in Absolute Mode independent of the value of the Absolute Mode indicator. The Processor stays in Absolute Mode if the Absolute Mode indicator is ON after the execution of the instructions.

The instruction at C+4 must not alter the contents of main store location C+5, and must not be an XED instruction.

If the contents of the instruction counter (PPR.IC) are changed during execution of the instruction pair at C+4, the next instruction is fetched from the modified C(PPR.IC); otherwise, the next instruction is fetched from C(PPR.IC)+1.

If the instruction at C+4 alters C(PPR.IC), then this transfer of control is effective immediately, and the instruction at C+5 is not executed.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

MME2 Master Mode Entry 2 004 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Causes a fault that fetches and executes, in Absolute Mode, the instruction pair at main store location C+52(octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

MODIFICATIONS: All, but none affect instruction execution

INDICATORS: None affected

NOTES: Attempted execution in BAR mode causes an Illegal Procedure, Illegal Opcode Fault. Except for the different constant used for fetching the instruction pair from main store, the MME2 instruction is identical to the Master Mode Entry (MME) instruction. Attempted repetition with RPT, RPD, or RPL causes an illegal procedure Fault.

MME3 Master Mode Entry 3 005 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Causes a fault that fetches and executes, in Absolute Mode, the instruction pair at main store location C+54(octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

MODIFICATIONS: All, but none affect instruction execution

INDICATORS: None affected

NOTES: Attempted execution in BAR mode causes an Illegal Procedure, Illegal Opcode Fault. Except for the different constant used for fetching the instruction pair from main store, the MME3 instruction is identical to the Master Mode Entry (MME) instruction. Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

MASTER MODE ENTRY

MME4

Master Mode Entry 4

007 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Causes a fault that fetches and executes, in Absolute Mode, the instruction pair at main store location C+56(octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

MODIFICATIONS: All, but none affect instruction execution

INDICATORS: None affected

NOTES: Attempted execution in BAR mode causes an Illegal Procedure, Illegal Opcode Fault.

Except for the different constant used for fetching the instruction pair from main store, the MME4 instruction is identical to the Master Mode Entry (MME) instruction.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

No-Operation

NOP No Operation 011 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: No operation takes place

MODIFICATIONS: All

INDICATORS: None affected (except as noted below)

NOTES: No operation takes place but address preparation is performed according to the specified modifier, if any. If modification other than DU or DL is used, the effective addresses generated may cause Store Faults.

The use of Indirect and Tally modifiers causes changes in the address and tally fields of the referenced Indirect Words and the Tally Runout indicator may be set ON as a result.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

PULS1 Pulse One 012 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: No operation takes place

MODIFICATIONS: All

INDICATORS: None affected (except as noted below)

NOTES: The PULS1 instruction is identical to the No Operation (NOP) instruction except that it causes certain unique synchronizing signals to appear in the Processor logic circuitry.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

NO OPERATION

PULS2

Pulse Two

013 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: No operation takes place

MODIFICATIONS: All

INDICATORS: None affected (except as noted below)

NOTES: The PULS2 instruction is identical to the No Operation (NOP) instruction except that it causes certain unique synchronizing signals to appear in the Processor logic circuitry.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Repeat

RPD

Repeat Double

560 (0)

FORMAT:

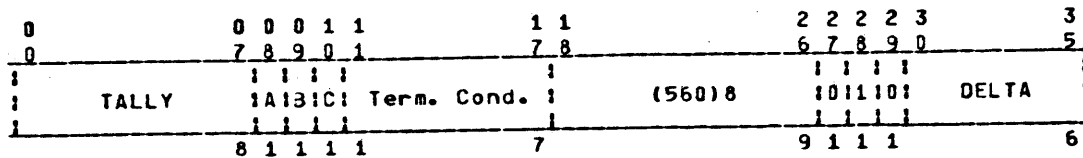


Figure 2-9 Repeat Double (RPD) Instruction Word Format

SUMMARY: Execute the pair of instructions at C(PPR.IC)+1 either a specified number of times or until a specified termination condition is met.

MODIFICATIONS: None

INDICATORS: (Indicators not listed are not affected)

Tally Runout If C(X)0,7 = 0 at termination, then ON; otherwise, OFF

All other Indicators None affected. However, the execution of the repeated instructions may affect indicators.

NOTES: The RPD instruction must be stored in an odd main store location except when accessed via the XEC or XED instructions, in which case the XEC or XED instruction must itself be in an odd main store location.

Both repeated instructions must use R or RI modifiers and only X1, X2, ..., X7 are permitted. For the purposes of this description, the even repeated instruction shall use X-even and the odd repeated instruction shall use X-odd. X-even and X-odd may be the same register.

If C = 1, then C(RPD instruction word)0,17 -> C(X); otherwise, C(X) unchanged prior to execution.

The termination condition and tally fields of C(X) control the repetition of the instruction pair. An initial tally of zero is interpreted as 256.

The repetition cycle consists of the following steps:

- a. Execute the pair of repeated instructions
- b. C(X)0,7 - 1 -> C(X)0,7
Modify C(X-even) and C(X-odd) as described below.
- c. If C(X)0,7 = 0, then set Tally Runout indicator ON and terminate.

REPEAT

- d. If a terminate condition has been met, then set Tally Runout Indicator OFF and terminate.
- e. Go to step a.

If a Processor Fault occurs during the execution of the instruction pair, the repetition loop is terminated and control passes to the Fault Trap according to the conditions for the Processor Fault. C(X0), C(X-even), and C(X-odd) are not updated for the repetition cycle in which the fault occurs. Note in particular that certain Processor Faults occurring during execution of the even instruction preclude the execution of the odd instruction for the faulting repetition cycle.

EIS Multiword instructions cannot be repeated. All other instructions may be repeated except as noted for individual instructions or those that ...

Explicitly alter C(X0)

The effective address, Y, of the operand (in the case of R modification) or indirect word (in the case of RI modification) is determined as follows:

For the first execution of the repeated instruction pair ...

```
C(C(PPR.IC)+1)0,17 + C(X-even) -> Y; Y-even ->
C(X-even)
C(C(PPR.IC)+2)0,17 + C(X-odd) -> Y-odd; Y-odd ->
C(X-odd)
```

For all successive executions of the repeated instruction pair ...

```
if C(X0)8 = 1, then C(X-even) + Delta -> Y-even,
Y-even -> C(X-even); otherwise, C(X-even) -> Y-even

if C(X0)9 = 1, then C(X-odd) + Delta -> Y-odd, Y-odd
-> C(X-odd); otherwise, C(X-odd) -> Y-odd
```

C(X0)8,9 correspond to Control Bits A and B, respectively, of the RPD instruction.

In the case of RI modification, only one indirect reference is made per repeated execution. The tag field of the indirect word is not interpreted. The indirect word is treated as though it had R modification with R = N.

The bit configuration in C(X0)11,17 defines the conditions

for which the repetition loop is terminated. The terminate conditions are examined at the completion of execution of the odd instruction. If more than one condition is specified, the repeat terminates if any of the specified conditions are met.

Bit 17 = 0 Ignore all overflows. Do not set Overflow Indicator; inhibit Overflow Fault.

REPEAT

- Bit 17 = 1 If Overflow Mask indicator is ON, then set Overflow indicator and terminate; otherwise, cause an Overflow Fault.
- Bit 16 = 1 Terminate if Carry indicator OFF.
- Bit 15 = 1 Terminate if Carry indicator ON.
- Bit 14 = 1 Terminate if Negative indicator OFF.
- Bit 13 = 1 Terminate if Negative indicator ON.
- Bit 12 = 1 Terminate if Zero indicator OFF.
- Bit 11 = 1 Terminate if Zero indicator ON.

At the time of terminations:

C(XD)0,7 contain the Tally Residue; that is, the number of repeats remaining until a Tally Runout would have occurred.

If the RPD instruction is interrupted (by any fault) before termination, the Tally Runout indicator is OFF.

C(X-even) and C(X-odd) contain the effective addresses of the next operands or indirect words that would have been used had the repetition loop not terminated.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

REPEAT

RPL

Repeat Link

500 (0)

FORMAT:

0	0 0 0 1 1	1 1	2 2 2 2 3	3
0	7 8 9 0 1	7 8	6 7 8 9 0	5
	TALLY	10 01C1 Term. Cond.	(500)8	1011010 0 0 0 0 01
	8 2 1 1	7	9 1 1 1	6

Figure 2-10 Repeat Link (RPL) Instruction Word Format

SUMMARY: Execute the instruction at C(PPR.IC)+1 either a specified number of times or until a specified termination condition is met.

MODIFICATIONS: None

INDICATORS: (Indicators not listed are not affected)

Tally Runout If C(X0)0,7 = 0 or link address C(Y) = 0 at termination, then ON; otherwise OFF

All other Indicators None affected. However, the execution of the repeated instruction may affect indicators.

NOTES: The repeated instruction must use an R modifier and only X1, X2, ..., X7 are permitted. For the purposes of this description, the repeated instruction shall use Xn.

If C = 1, then C(RPL instruction word)0,17 -> C(X0); otherwise, C(X0) unchanged prior to execution.

The termination condition and tally fields of C(X0) control the repetition of the instruction. An initial tally of zero is interpreted as 256.

The repetition cycle consists of the following steps:

- a. Execute the repeated instruction
- b. C(X0)0,7 - 1 -> C(X0)0,7
Modify C(Xn) as described below.
- c. If C(X0)0,7 = 0 or C(Y)0,17 = 0, then set Tally Runout indicator ON and terminate.
- d. If a terminate condition has been met, then set Tally Runout indicator OFF and terminate.
- e. Go to step a.

If a Processor Fault occurs during the execution of the instruction, the repetition loop is terminated and control passes to the Fault Trap according to the conditions for the Processor Fault. $C(X0)$ and $C(Xn)$ are not updated for the repetition cycle in which the fault occurs.

EIS Multiword instructions cannot be repeated. All other instructions may be repeated except as noted for individual instructions or those that ...

Explicitly alter $C(X0)$

Explicitly alter the link address, $C(Y)0,17$

The effective address, Y , of the operand is determined as follows:

For the first execution of the repeated instruction ...

$C((PPR.IC)+1)0,17 + C(Xn) \rightarrow Y; Y \rightarrow C(Xn)$

For all successive executions of the repeated instruction ...

$C(Xn) \rightarrow Y$
if $C(Y)0,17 \neq 0$, then $C(Y)0,17 \rightarrow C(Xn)$; otherwise,
no change to $C(Xn)$

$C(Y)0,17$ is known as the link address and is the effective address of the next entry in a threaded list of operands to be referenced by the repeated instruction.

The operand data is formed as

$(00...0)0,17 \parallel C(Y)18,p$

where p is 35 for single precision operands and 71 for double precision operands.

The bit configuration in $C(X0)11,17$ and the link address, $C(Y)0,17$, define the conditions for which the repetition loop is terminated. The terminate conditions are examined at the completion of execution of the instruction. If more than one condition is specified, the repeat terminates if any of the specified conditions are met.

$C(Y)0,17 = 0$ Set Tally Runout indicator ON and terminate.

Bit 17 = 0 Ignore all overflows. Do not set Overflow Indicator; inhibit Overflow Fault.

Bit 17 = 1 If Overflow Mask indicator is ON, then set

Overflow indicator and terminate; otherwise, cause an Overflow Fault.

Bit 16 = 1 Terminate if Carry indicator OFF.

Bit 15 = 1 Terminate if Carry indicator ON.

Bit 14 = 1 Terminate if Negative indicator OFF.

REPEAT

Bit 13 = 1 Terminate if Negative indicator ON.

Bit 12 = 1 Terminate if Zero indicator OFF.

Bit 11 = 1 Terminate if Zero indicator ON.

At the time of termination:

C(X0)0,7 contain the Tally Residue; that is, the number of repeats remaining until a Tally Runout would have occurred.

If the RPL instruction is interrupted (by any fault) before termination, the Tally Runout indicator is OFF.

C(Xn) contain the last link address, that is, the effective address of the list word containing the last operand data and the next link address.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

RPT

Repeat

520 (0)

FORMAT:

0	0 0 0 1 1	1 1	2 2 2 2 3	3
0	7 8 9 0 1	7 8	6 7 8 9 0	5
	TALLY 0 0 C Term. Cond.	(520) 8	0 1 1 0	DELTA
	8 2 1 1	7	9 1 1 1	6

Figure 2-11 Repeat (RPT) Instruction Word Format

SUMMARY: Execute the instruction at C(PPR.IC)+1 either a specified number of times or until a specified termination condition is met.

MODIFICATIONS: None

INDICATORS: (Indicators not listed are not affected)

Tally Runout If C(X0)0,7 = 0 at termination, then ON; otherwise, OFF

All other Indicators None affected. However, the execution of the repeated instruction may affect indicators.

NOTES: The repeated instruction must use an R or RI modifier and only X1, X2, ..., X7 are permitted. For the purposes of this description, the repeated instruction shall use Xn.

If C = 1, then C(RPT instruction word)0,17 -> C(X0); otherwise, C(X0) unchanged prior to execution.

The termination condition and tally fields of C(X0) control the repetition of the instruction pair. An initial tally of zero is interpreted as 256.

The repetition cycle consists of the following steps:

- a. Execute the repeated instruction
- b. C(X0)0,7 - 1 -> C(X0)0,7
Modify C(Xn) as described below
- c. If C(X0)0,7 = 0, then set Tally Runout indicator ON and terminate
- d. If a terminate condition has been met, then set Tally Runout indicator OFF and terminate
- e. Go to step a

REPEAT

If a Processor Fault occurs during the execution of the instruction, the repetition loop is terminated and control passes to the Fault Trap according to the conditions for the Processor Fault. C(X0) and C(Xn) are not updated for the repetition cycle in which the fault occurs.

EIS Multiword instructions cannot be repeated. All other instructions may be repeated except as noted for individual instructions or those that ...

Explicitly alter C(X0)

Explicitly alter C(PPR.IC)+2

The effective address, Y, of the operand (in the case of R modification) or indirect word (in the case of RI modification) is determined as follows:

For the first execution of the repeated instruction ...

$C(C(PPR.IC)+1)0,17 + C(Xn) \rightarrow Y; Y \rightarrow C(Xn)$

For all successive executions of the repeated instruction ...

if C(X0)8 = 1, then $C(Xn) + \Delta \rightarrow Y, Y \rightarrow C(Xn)$;
otherwise, $C(Xn) \rightarrow Y$

C(X0)8 corresponds to Control Bit A of the RPD instruction.

In the case of RI modification, only one indirect reference is made per repeated execution. The tag field of the indirect word is not interpreted. The indirect word is treated as though it had R modification with R = N.

The bit configuration in C(X0)11,17 defines the conditions for which the repetition loop is terminated. The terminate conditions are examined at the completion of execution of the instruction. If more than one condition is specified, the repeat terminates if any of the specified conditions are met.

Bit 17 = 0 Ignore all overflows. Do not set Overflow Indicator; inhibit Overflow Fault.

Bit 17 = 1 If Overflow Mask indicator is ON, then set Overflow indicator and terminate; otherwise, cause an Overflow Fault.

Bit 16 = 1 Terminate if Carry indicator OFF.

Bit 15 = 1 Terminate if Carry indicator ON.

Bit 14 = 1 Terminate if Negative indicator OFF.

Bit 13 = 1 Terminate if Negative indicator ON.

Bit 12 = 1 Terminate if Zero indicator OFF.

Bit 11 = 1 Terminate if Zero indicator ON.

At the time of termination:

C(XD)0,7 contain the Tally Residue; that is, the number of repeats remaining until a Tally Runout would have occurred.

If the RPT instruction is interrupted (by any fault) before termination, the Tally Runout indicator is OFF.

C(Xn) contain the effective address of the next operand or indirect word that would have been used had the repetition loop not terminated.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

RING ALARM REGISTER

Ring Alarm Register

SRA Store Ring Alarm 754 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: DD...0 -> C(Y)0,32
 C(RALR) -> C(Y)33,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted execution in BAR Mode causes an Illegal Procedure Fault.

 Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Store Base Address Register

SBAR Store Base Address Register 550 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(BAR) -> C(Y)0,17

MODIFICATIONS: All except DU, DL, CI, SC, and SCR

INDICATORS: None affected

TRANSLATION

Translation

BCD

Binary to Binary-Coded-Decimal

505 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(A) left three positions

$C(A) / C(Y) \rightarrow$ 4-bit quotient plus remainder

Shift C(Q) left six positions

4-bit quotient \rightarrow C(Q) 32,35

remainder \rightarrow C(A)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON

Negative If C(A)0 = 1 before execution, then ON; otherwise OFF

NOTES: The BCD instruction carries out one step in an algorithm for the conversion of a binary number to a string of Binary-Coded-Decimal (BCD) digits. The algorithm requires the repeated short division of the binary number or last remainder by a set of constants $C(i) = 8^{*i} \times 10^{*(n-i)}$ for $i = 1, 2, \dots, n$ with n being defined by:

$$10^{*(n-1)} \leq \langle \text{binary number} \rangle \leq 10^{*n} - 1.$$

The values in the table that follows are the conversion constants to be used with the BCD instruction. Each vertical column represents the set of constants to be used depending on the initial value of the binary number to be converted. The instruction is executed once per digit while traversing the appropriate column from top to bottom.

An alternate use of the table for conversion involves the use of the constants in the row corresponding to conversion step 1. If, after each execution, the contents of the accumulator are shifted right 3 positions, the constants in the first row, starting at the appropriate

column, may be used while traversing the row from left to right.

Because there is a limit on range, a full 36 bit word cannot be converted. The largest binary number that may be converted correctly is $2^{*33} - 1$ yielding ten decimal digits.

Attempted repetition with RPL causes an Illegal Procedure Fault.

For $10^{*(n-1)} \leq |C(AR)| \leq 10^{*n} - 1$ and $n = \dots$

i =	10	9	8	7	6	5	4	3	2	1
1	8000000000	800000000	80000000	8000000	800000	80000	8000	800	80	8
2	6400000000	640000000	64000000	6400000	640000	64000	6400	640	64	
3	5120000000	512000000	51200000	5120000	512000	51200	5120	512		
4	4096000000	409600000	40960000	4096000	409600	40960	4096			
5	3276800000	327680000	32768000	3276800	327680	32768				
6	2621440000	262144000	26214400	2621440	262144					
7	2097152000	209715200	20971520	2097152						
8	1677721600	167772160	16777216							
9	1342177280	134217728								
10	1073741824									

GTB

Gray to Binary

774 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(A) converted from Gray Code to a 36 bit binary number

MODIFICATIONS: None

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A)0 = 1, then ON; otherwise OFF

NOTES: This conversion is defined by the following algorithm:

$C(A)0 \rightarrow C(A)0$

$C(A)(i) \oplus C(A)(i-1) \rightarrow C(A)(i)$ for $i = 1, 2, \dots, 35$

Attempted repetition with RPL causes an Illegal Procedure Fault.

PRIVILEGED - REGISTER LOAD

PRIVILEGED INSTRUCTIONS

Privileged - Register Load

LBAR Load Base Address Register 230 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y)0,17 -> C(BAR)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Attempted execution in BAR Mode causes a Illegal Procedure Fault.

LCPR Load Central Processor Register 674 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Load selected register as noted

MODIFICATIONS: None. The instruction TAG field is used for register selection as follows.

C(TAG) Data and Register(s)

02 C(Y) -> C(Cache Mode Register)0,35

04 C(Y) -> C(Mode Register)0,35

03 00...0 -> C(CU, OU, DU, and APU History Register)0,71

07 11...1 -> C(CU, OU, DU, and APU History Register)0,71

INDICATORS: None affected

NOTES: See Section IV, Program Accessible Registers, for descriptions and use of the various registers.

For TAG values 03 and 07, the History Register loaded is selected by the current value of a Cyclic Counter for each Unit. All four Cyclic Counters are advanced by one count

for each execution of the instruction.

Use of TAG values other than those defined above causes an Illegal Procedure Fault.

Attempted execution in Normal or BAR Mode causes a Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LDBR Load Descriptor Segment Base Register 232 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If SDWAM is enabled, then
 0 -> C(SDWAM(i).FULL) for i = 0, 1, ..., 15
 (i) -> C(SDWAM(i).USE) for i = 0, 1, ..., 15
 If PTWAM is enabled, then
 0 -> C(PTWAM(i).FULL) for i = 0, 1, ..., 15
 (i) -> C(PTWAM(i).USE) for i = 0, 1, ..., 15
 C(Y-pair)0,23 -> C(DSBR.ADDR)
 C(Y-pair)37,50 -> C(DSBR.BOUND)
 C(Y-pair)55 -> C(DSBR.U)
 C(Y-pair)60,71 -> C(DSBR.STACK)

MODIFICATIONS: All except DU, DL, CI, SC, and SCR

INDICATORS: None affected

NOTES: The hardware assumes Y17 = 0; no check is made.
 The Associative Memories are cleared (FULL indicators reset) if they are enabled.
 See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging, for description

and use, respectively, of the SDWAM, PTWAM, and DSBR.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

PRIVILEGED - REGISTER LOAD

LDT Load Timer Register 637 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y)0,26 -> C(TR)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: None Affected

NOTES: Attempted execution in Normal or BAR Mode causes a Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LPTP Load Page Table Pointers 257 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$
 $m = C(PTWAM(i).USE)$
 $C(Y+m)0,14 \rightarrow C(PTWAM(m).POINTER)$
 $C(Y+m)15,26 \rightarrow C(PTWAM(m).PAGE)$
 $C(Y+m)27 \rightarrow C(PTWAM(m).F)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes $Y14,17 = 0000$; no check is made.

The Associative Memory is ignored (forced to "no match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging, for description and use, respectively, of the PTWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LPTR	Load Page Table Registers	173 (1)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	For $i = 0, 1, \dots, 15$ $m = C(PTWAM(i).USE)$ $C(Y+m)0,17 \rightarrow C(PTWAM(m).ADDR)$ $C(Y+m)29 \rightarrow C(PTWAM(m).M)$	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	None affected	
NOTES:	The hardware assumes $Y14,17 = 0000$; no check is made. The Associative Memory is ignored (forced to "no match") during Address Preparation. See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging, for description and use, respectively, of the PTWAM. Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault. Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	
LRA	Load Ring Alarm Register	774 (1)
FORMAT:	Basic Instruction Format (See Figure 2-1).	
SUMMARY:	$C(Y)33,35 \rightarrow C(RALR)$	
MODIFICATIONS:	All except DU, DL, CI, SC, SCR	
INDICATORS:	None affected	
NOTES:	Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault. Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.	

LSDP Load Segment Descriptor Pointers 257 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$
 $m = C(SDWAM(i).USE)$
 $C(Y+m)0,14 \rightarrow C(SDWAM(m).POINTER)$
 $C(Y+m)17 \rightarrow C(SDWAM(m).P)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes $Y14,17 = 0000$; no check is made.
The Associative Memory is ignored (forced to "no match") during Address Preparation.
See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging, for description and use, respectively, of the SDWAM.
Attempted execution in Normal or 3AR Mode causes an Illegal Procedure Fault.
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LSDR Load Segment Descriptor Registers 232 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$

$m = C(SDWAM(i).USE)$

$C(Y+2m)0,23 \rightarrow C(SDWAM(m).ADR)$

$C(Y+2m)24,32 \rightarrow C(SDWAM(m).R1, R2, R3)$

$C(Y+2m)37,50 \rightarrow C(SDWAM(m).BOUND)$

$C(Y+2m)52,57 \rightarrow C(SDWAM(m).R, E, W, P, U, G, C)$

$C(Y+2m)58,71 \rightarrow C(SDWAM(m).CL)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None Affected

NOTES:

The hardware assumes $Y14,17 = 0000$; no check is made.

The Associative Memory is ignored (forced to "no-match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the SDWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

RCU Restore Control Unit 613 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: $C(Y\text{-block8})$ words 0 to 7 \rightarrow C(Control Unit Data)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

PRIVILEGED - REGISTER LOAD

NOTES:

See Section IV, Program Accesible Registers, for description and use of Control Unit Data.

The hardware assumes Y15,17 = 000 and addressing is incremented accordingly; no check is made.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Privileged - Register Store

SCPR Store Central Processor Register 452 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Store selected register as noted

MODIFICATIONS: None. The instruction TAG field is used for register selection as follows.

C(TAG)	MEANING
00	C(APU History Register) -> C(Y-pair)
01	C(Fault Register) -> C(Y-pair)0,35 00...0 -> C(Y-pair)36,71
06	C(Mode Register) -> C(Y-pair)0,35 C(Cache Mode Register) -> C(Y-pair)36,71
20	C(CU History Register) -> C(Y-pair)
40	C(DU History Register) -> C(Y-pair)
60	C(DU History Register) -> C(Y-pair)

INDICATORS: None affected

NOTES: See Section IV, Program Accessible Registers, for description and use of the various registers.

For TAG values 00, 20, 40, and 60, the History Register stored is selected by the current value of a Cyclic Counter for each Unit. The individual Cyclic Counters are advanced by one count for each execution of the instruction.

The use of TAG values other than those defined above causes an Illegal Procedure Fault.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

PRIVILEGED - REGISTER STORE

SCU Store Control Unit 657 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Control Unit Data) -> C(Y-block8) words 0 to 7

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: See Section IV, Program Accessible Registers, for description and use of Control Unit Data.

The SCU instruction safe-stores control information required to service a Processor fault. The Control Unit Data is not, in general, valid at any time except when safe-stored by the first instruction of a fault/interrupt vector.

The hardware assumes Y15,17 = 000 and addressing is incremented accordingly; no check is made.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SDBR Store Descriptor Segment Base Register 154 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(DSBR.ADDR) -> C(Y-pair)0,23
 00...0 -> C(Y-pair)24,36
 C(DSBR.BOUND) -> C(Y-pair)37,50
 0000 -> C(Y-pair)51,54
 C(DSBR.U) -> C(Y-pair)55
 000 -> C(Y-pair)56,59
 C(DSBR.STACK) -> C(Y-pair)60,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes Y 17 = 0; no check is made.

C(DSBR) is unchanged.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the DBR.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPTP Store Page Table Pointers 557 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$

- C(PHWAM(i).POINTER) -> C(Y+i)0,14
- C(PHWAM(i).PAGE) -> C(Y+i)15,26
- C(PHWAM(i).F) -> C(Y+i)27
- 0000 -> C(Y+i)28,31
- C(PHWAM(i).USE) -> C(Y+i)32,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES:

- The hardware assumes that $Y14,17 = 0000$, and addressing is incremented accordingly; no check is made.
- The contents of PHWAM(m) remain unchanged.
- The Associative Memory is ignored (forced to a "no match") during Address Preparation.
- See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the PHWAM.
- Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPTR

Store Page Table Registers

154 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$

$C(PTWAM(i).ADDR) \rightarrow C(Y+1)0,17$

$00\dots0 \rightarrow C(Y+1)18,28$

$C(PTWAM(i).M) \rightarrow C(Y+1)29$

$00\dots0 \rightarrow C(Y+1)30,35$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES:

The hardware assumes that $Y14,17 = 0000$, and addressing will be incremented accordingly; no check is made.

The contents of $PTWAM(m)$ are unchanged.

The Associative Memory is ignored (forced to a "no match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the $PTWAM$.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SSDP Store Segment Descriptor Pointers 557 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$

C(SDWAM(i).POINTER) -> C(Y+i)0,14

00...0 -> C(Y+i)15,26

C(SDWAM(i).F) -> C(Y+i)27

0000 -> C(Y+i)28,31

C(SDWAM(i).USE) -> C(Y+i)32,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes $Y_{14,17} = 0000$, and addressing is incremented accordingly; no check is made.

The contents of SDWAM(i) are unchanged.

The Associative Memory is ignored (forced to a "no match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the SDWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SSDR

Store Segment Descriptor Registers

254 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$

$C(SDWAM(i).ADDR) \rightarrow C(Y+2i-pair)0,23$

$C(SDWAM(i).R1, R2, R3) \rightarrow C(Y+2i-pair)24,32$

$0000 \rightarrow C(Y+2i-pair)33,36$

$C(SDWAM(i).BOUND) \rightarrow C(Y+2i-pair)37,50$

$C(SDWAM(i).R, E, P, U, G, C) \rightarrow C(Y+2i-pair)51,57$

$C(SDWAM(i).CL) \rightarrow C(Y+2i-pair)58,71$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes $Y13,17 = 00000$, and addressing is incremented accordingly; no check is made.

The contents of $SDWAM(i)$ are unchanged.

The Associative Memory is ignored (forced to a "no match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the $SDWAM$.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

PRIVILEGED - CLEAR ASSOCIATIVE MEMORY

Privileged - Clear Associative Memory

CAMP Clear Associative Memory Paged 532 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$
0 -> C(PTWAM(i).F)
(i) -> C(PTWAM(i).USE)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The Full/Empty bit of each PTWAM Register is set to 0, and the usage counters (PTWAM.USE) are set to their pre-assigned values of 0 through 15. The remainder of C(PTWAM(i)) is unchanged.

The execution of this instruction enables the PTWAM if it is disabled and C(TPR.CA)16,17 = 01.

The execution of this instruction disables the PTWAM if C(TPR.CA)16,17 = 10.

If C(TPR.CA)15 = 1, a selective clear of cache is executed. Any cache block for which the upper 14 bits of the directory entry equal C(TPR.CA)0,13 will have its Full/Empty bit set to Empty.

See Section IV, Program Accessible Registers and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the PTWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure fault.

CAHS Clear Associative Memory Segments 532 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$
 $0 \rightarrow C(SDWAM(i).F)$
 $(i) \rightarrow C(SDWAM(i).USE)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The Full/Empty bit of each SDWAM Register is set to zero, and the usage counters (SDWAM.USE) are initialized to their pre-assigned values of 0 through 15. The remainder of C(SDWAM(i)) are unchanged.

The execution of this instruction enables the SDWAM if it is previously disabled and if C(TPR.CA)16,17 = 01.

The execution of this instruction disables the SDWAM if C(TPR.CA)16,17 = 10.

The execution of this instruction sets the Full/Empty bits of all cache blocks to Empty if C(TPR.CA)15 = 1.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the SDWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

PRIVILEGED - CONFIGURATION AND STATUS

Privileged - Configuration and Status

RNCH Read Memory Controller Mask Register 233 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For the selected System Controller:

If the Processor has a Mask Register assigned, then

C(MR)0,15 -> C(AQ)0,15

00...0 -> C(AQ)16,31

C(MR)32,35 -> C(AQ)32,35

C(MR)36,51 -> C(AQ)36,51

00...0 -> C(AQ)52,67

C(MR)68,71 -> C(AQ)68,71

otherwise, 00...0 -> C(AQ)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ)0 = 1, then ON; otherwise OFF

NOTES: The contents of the Mask Register remain unchanged.

C(TPR.CA)0,2 specify which Processor Port (i.e., which System Controller) is used.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

RSCR

Read System Controller Register

412 (0)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

The effective address, Y, is used to select a system controller (SCU) and the function to be performed as follows:

Effective Address	Function
y0000x	C(SCU Mode Register) -> C(AQ)
y0001x	C(SCU Configuration Switches) -> C(AQ)
y0002x	C(Interrupt Mask Port 0) -> C(AQ)
y0012x	C(Interrupt Mask Port 1) -> C(AQ)
y0022x	C(Interrupt Mask Port 2) -> C(AQ)
y0032x	C(Interrupt Mask Port 3) -> C(AQ)
y0042x	C(Interrupt Mask Port 4) -> C(AQ)
y0052x	C(Interrupt Mask Port 5) -> C(AQ)
y0062x	C(Interrupt Mask Port 6) -> C(AQ)
y0072x	C(Interrupt Mask Port 7) -> C(AQ)
y0003x	C(Interrupt Cells) -> C(AQ)
y0004x or y0005x	C(System Clock) -> C(AQ)
y0006x or y0007x	C(Store Unit Mode Register) -> C(AQ)
where:	y = octal value of Y0,2 as used to select SCU x = any octal digit

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: See Section IV, Program Accessible Registers, for description and use of the various registers.

For effective addresses y0006x and y0007x, Store Unit selection is done by the normal address decoding function of the System Controller.

PRIVILEGED - CONFIGURATION AND STATUS

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPL causes an Illegal Procedure Fault.

RSW Read Switches 231 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: The effective address, Y, is used to select certain Processor switches whose settings are read into C(A).

The switches selected are as follows:

Effective

Address Function

xxxxx0 C(Data Switches) -> C(A)

xxxxx1 C(Config. Switches, ports A, B, C, D) -> C(A)

xxxxx2 00...0 -> C(A)0,5
C(Fault Base Switches) -> C(A)6,12
00...0 -> C(A)13,26
C(Processor ID) -> C(A)27,33
C(Processor Number Switches) -> C(A)34,35

xxxxx3 C(Config. Switches, ports E, F, G, H) -> C(A)

xxxxx4 00...0 -> C(A)0,12
C(Port Interface and Size Switches) -> C(A)13,28
00...0 -> C(A)29,35

MODIFICATIONS: All, but none affect instruction execution

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A)0 = 1, then ON; otherwise OFF

NOTES: See Section IV, Program Accessible Registers for description and use of the switch data.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Privileged - System Control

CIOC Connect I/O Channel 015 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: The System Controller addressed by Y (i.e., contains the word at Y) sends a connect pulse to the port specified by C(Y)33,35.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

 Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SMCH Set Memory Controller Mask Register 553 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For the selected System Controller:

 If the Processor has a Mask Register assigned, then

 C(AQ)0,15 -> C(MR)0,15

 C(AQ)32,35 -> C(MR)32,35

 C(AQ)36,51 -> C(MR)36,51

 C(AQ)68,71 -> C(MR)68,71

 otherwise, a Store Fault, Not Control, occurs.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: C(AQ) are unchanged.

 C(TPR.CA)0,2 specify which Processor Port (i.e., which System Controller) is used.

 Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

PRIVILEGED - SYSTEM CONTROL

Attempted repetition with RPL causes an Illegal Procedure Fault.

SHIC Set Memory Controller Interrupt Cells 451 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For $i = 0, 1, \dots, 15$ and $C(A)_{35} = 0$:
if $C(A)_i = 1$, then set Interrupt Cell i ON
For $i = 0, 1, \dots, 15$ and $C(A)_{35} = 1$:
if $C(A)_i = 1$, then set Interrupt Cell $16+i$ ON

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: $C(TPR.CA)_{0,2}$ specify which Processor Port (i.e., which System Controller) is used.
Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

SSCR

Set System Controller Register

057 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: The effective address, Y, is used to select a System Controller (SCU) and the function to be performed as follows:

<u>Effective Address</u>	<u>Function</u>
y0000x	C(AQ) -> C(SCU Mode Register)
y0001x	Reserved
y0002x	C(AQ) -> C(Interrupt Mask Port 0)
y0012x	C(AQ) -> C(Interrupt Mask Port 1)
y0022x	C(AQ) -> C(Interrupt Mask Port 2)
y0032x	C(AQ) -> C(Interrupt Mask Port 3)
y0042x	C(AQ) -> C(Interrupt Mask Port 4)
y0052x	C(AQ) -> C(Interrupt Mask Port 5)
y0062x	C(AQ) -> C(Interrupt Mask Port 6)
y0072x	C(AQ) -> C(Interrupt Mask Port 7)
y0003x	C(AQ)0,15 -> C(Interrupt Cells)(0,15) C(AQ)36,51 -> C(Interrupt Cells)(16,31)
y0006x or y0007x	C(AQ) -> C(Store Unit Mode Register)
where:	y = octal value of Y0,2 as used to select SCU x = any octal digit

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: If the Processor does not have a Mask Register assigned in the selected System Controller, a Store Fault, Not Control, will occur.

For effective addresses y0006x and y0007x, Store Unit selection is done by the normal address decoding function of the System Controller.

PRIVILEGED - SYSTEM CONTROL

See Section IV, Program Accessible Registers, for description and use of the various registers.

Attempted execution on Normal or BAR Mode causes an Illegal Procedure Fault.

Privileged - Miscellaneous

ABSA Absolute Address to Accumulator 212 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Final Main Store Address -> C(A)0,23
00...0 -> C(A)24,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A)0 = 1, then ON; otherwise OFF

NOTES: If the ABSA instruction is executed in Absolute mode, C(A) will be undefined.

Attempted execution in Normal or BAR modes causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

DIS Delay Until Interrupt Signal 616 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: No operation takes place, and the Processor does not continue with the next instruction; it waits for a program interrupt signal.

MODIFICATIONS: All, but none affect instruction execution

INDICATORS: None affected

NOTES: Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - ADDRESS REGISTER LOAD

EXTENDED INSTRUCTION SET (EIS)

EIS - Address Register Load

AARn Alphanumeric Descriptor to ARn 56n (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

 C(Y)0,17 -> C(PRn.WORDNO)

 If C(Y)21,22 = 00 (TA code = 0), then

 C(Y)18,19 -> C(PRn.CHAR)

 0000 -> C(PRn.BITNO)

 If C(Y)21,22 = 01 (TA code = 1), then

$(6 * C(Y)18,20) / 9 \rightarrow C(PRn.CHAR)$

$(6 * C(Y)18,20) \text{ modulo } 9 \rightarrow C(PRn.BITNO)$

 If C(Y)21,22 = 10 (TA code = 2), then

$C(Y)18,20 / 2 \rightarrow C(PRn.CHAR)$

$4 * (C(Y)18,20 \text{ modulo } 2) + 1 \rightarrow C(PRn.BITNO)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected.

NOTES: An alphanumeric descriptor is fetched from Y and C(Y)21,22 (TA field) is examined to determine the data type described.

 If TA = 0 (9-bit data), C(Y)18,19 goes to C(PRn.CHAR) and zeros fill C(PRn.BITNO).

 If TA = 1 (6-bit data) or TA = 2 (4-bit data), C(Y)18,20 is appropriately translated into an equivalent character and bit position that goes to C(PRn.CHAR) and C(PRn.BITNO).

 If C(Y)21,22 = 11 (TA code = 3) an Illegal Procedure Fault occurs.

 If C(Y)23 = 1 an Illegal Procedure Fault occurs.

 If C(Y)21,22 = 00 (TA code = 0) and C(Y)20 = 1 an Illegal Procedure Fault occurs.

 If C(Y)21,22 = 01 (TA code = 1) and C(Y)18,20 = 110 or 111 an Illegal Procedure Fault occurs.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LARn Load Address Register n 76n (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
C(Y)0,23 -> C(ARn)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LAREG Load Address Registers 463 (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., 7
C(Y+n)0,23 -> C(ARn)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes Y15,17 = 000 and addressing is incremented accordingly; no check is made.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LPL Load Pointers and Lengths 467 (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: C(Y-block8) -> C(Decimal Unit Control Jata)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

EIS - ADDRESS REGISTER LOAD

INDICATORS: None affected

NOTES: See Section IV, Program Accessible Registers, for description and use of Decimal Unit Control Data.

The hardware assumes Y15,17 = 000 and addressing is incremented accordingly; no check is made.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

NARn Numeric Descriptor to ARn 66n (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(Y)0,17 → C(PRn.WORDNO)

If C(Y)21 = 0 (TN code = 0), then

C(Y)18,20 → C(PRn.CHAR)

0000 → C(PRn.BITNO)

If C(Y)21 = 1 (TN code = 1), then

(C(Y)18,20) / 2 → C(PRn.CHAR)

4 * (C(Y)18,20 modulo 2) + 1 → C(PRn.BITNO)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: A numeric descriptor is fetched from Y and C(Y)21 (TN bit) is examined.

If TN = 0 (9-bit data), then C(Y)18,19 go to C(PRn.CHAR) and zeros fill C(PRn.BITNO).

If TN = 1 (4-bit data), C(Y)18,20 is appropriately translated to an equivalent character and bit position that goes to C(PRn.CHAR) and C(PRn.BITNO).

If C(Y)21 = 0 (TN code = 0) and C(Y)20 = 1 an Illegal Procedure Fault occurs.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Address Register Store

ARAN ARn to Alphanumeric Descriptor 54n (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
 $C(\text{PRn.WORDNO}) \rightarrow C(\text{Y})_{0,17}$
 If $C(\text{Y})_{21,22} = 00$ (TA code = 0), then
 $C(\text{PRn.CHAR}) \rightarrow C(\text{Y})_{18,19}$
 $0 \rightarrow C(\text{Y})_{20}$
 If $C(\text{Y})_{21,22} = 01$ (TA code = 1), then
 $(9 * C(\text{PRn.CHAR}) + C(\text{PRn.BITNO}) / 6 \rightarrow C(\text{Y})_{18,20}$
 If $C(\text{Y})_{21,22} = 10$ (TA code = 2), then
 $(9 * C(\text{PRn.CHAR}) + C(\text{PRn.BITNO}) - 1) / 4 \rightarrow C(\text{Y})_{18,20}$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: This instruction is the inverse of AARn.
 The alphanumeric descriptor is fetched from Y and $C(\text{Y})_{21,22}$ (TA field) is examined to determine the data type described.
 If TA = 0 (9-bit data), $C(\text{PRn.CHAR})$ goes to $C(\text{Y})_{18,19}$.
 If TA = 1 (6-bit data) or TA = 2 (4-bit data), $C(\text{PRn.CHAR})$ and $C(\text{PRn.BITNO})$ are translated to an equivalent character position that goes to $C(\text{Y})_{18,20}$.
 If $C(\text{Y})_{21,22} = 11$ (TA code = 3) or $C(\text{Y})_{23} = 1$ (unused bit), an Illegal Procedure Fault occurs.
 Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - ADDRESS REGISTER STORE

ARNn ARn to Numeric Descriptor 64n (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

 C(PRn.WORDNO) -> C(Y)0,17

 If C(Y)21 = 0 (TN code = 0), then

 C(PRn.CHAR) -> C(Y)18,19

 0 -> C(Y)20

 If C(Y)21 = 1 (TN code = 1), then

 (9 * C(PRn.CHAR) + C(PRn.BITNO) - 1) / 4 -> C(Y)18,20

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: This instruction is the inverse of NARN.

 The numeric descriptor is fetched from Y and C(Y)21 (TN bit) is examined.

 If TN = 0 (9-bit data), then C(PRn.CHAR) goes to C(Y)18,19.

 If TN = 1 (4-bit data), then C(PRn.CHAR) and C(PRn.BITNO) are translated to an equivalent character position that goes to C(Y)18,20.

 Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SARN Store Address Register n 74n (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

 C(PRn.AR) -> C(Y)0,23

 00...0 -> C(Y)24,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

EIS - ADDRESS REGISTER STORE

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SAREG Store Address Registers 443 (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For $n = 0, 1, \dots, 7$
 $C(ARn) \rightarrow C(Y+n)0,23$
 $00\dots0 \rightarrow C(Y+n)24,35$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTE: The hardware assumes $Y15,17 = 000$ and addressing is incremented accordingly; no check is made.
 Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPL Store Pointers and Lengths 447 (1)

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: $C(\text{Decimal Unit Control Data}) \rightarrow C(Y-\text{block}8)$

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes $Y15,17 = 000$ and addressing is incremented accordingly; no check is made.
 See Section IV, Program Accessible Registers, for description and use of Decimal Unit Control Data.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

EIS - Address Register Special Arithmetic

A4BD

Add 4-Bit Displacement to Address Register

502 (1)

FORMAT:

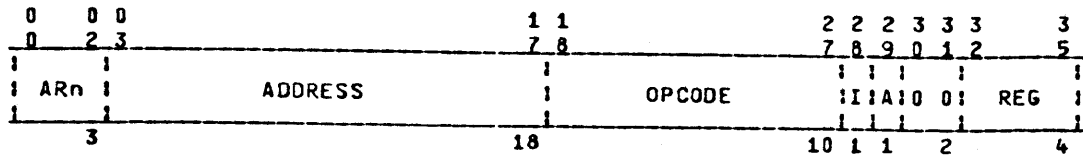


Figure 2-12 EIS Address Register Special Arithmetic Instruction Format

- ARn Number of Address Register selected
- ADDRESS Literal word displacement value
- OPCODE Instruction operation code
- I Program Interrupt inhibit bit
- A Use Address Register contents flag
- REG Any Register Modifier except DU, DL, and IC

SUMMARY:

If A = 0, then

$ADDRESS + C(REG) / 4 \rightarrow C(PRn.WORDNO)$

$C(REG) \text{ modulo } 4 \rightarrow C(PRn.CHAR)$

$4 * (C(REG) \text{ modulo } 2) + 1 \rightarrow C(PRn.BITNO)$

If A = 1, then

$C(PRn.WORDNO) + ADDRESS + (9 * C(PRn.CHAR) + 4 * C(REG) + C(PRn.BITNO)) / 36 \rightarrow C(PRn.WORDNO)$

$((9 * C(PRn.CHAR) + 4 * C(REG) + C(PRn.BITNO)) \text{ modulo } 36) / 9 \rightarrow C(PRn.CHAR)$

$4 * (C(PRn.CHAR) + 2 * C(REG) + C(PRn.BITNO) / 4) \text{ modulo } 2 + 1 \rightarrow C(PRn.BITNO)$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn

INDICATORS: None affected

NOTES: The steps described in SUMMARY define special 4-bit addition arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal procedure Fault.

A680

Add 6-Bit Displacement to Address Register

501 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

SUMMARY:

If A = 0, then

$ADDRESS + C(REG) / 6 \rightarrow C(PRn.WORDNO)$

$((6 * C(REG)) \text{ modulo } 36) / 9 \rightarrow C(PRn.CHAR)$

$(6 * C(REG)) \text{ modulo } 9 \rightarrow C(PRn.BITNO)$

If A = 1, then

$C(PRn.WORDNO) + ADDRESS + (9 * C(PRn.CHAR) + 6 * C(REG) + C(PRn.BITNO)) / 36 \rightarrow C(PRn.WORDNO)$

$((9 * C(PRn.CHAR) + 6 * C(REG) + C(PRn.BITNO)) \text{ modulo } 36) / 9 \rightarrow C(PRn.CHAR)$

$(9 * C(PRn.CHAR) + 6 * C(REG) + C(PRn.BITNO)) \text{ modulo } 9 \rightarrow C(PRn.BITNO)$

MODIFICATIONS: None except AU, QU, AL, QL, and Xn

INDICATORS: None Affected

NOTES:

The steps described in SUMMARY define special 6-bit addition arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

A98D Add 9-Bit Displacement to Address Register 500 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

SUMMARY: If A = 0, then
ADDRESS + C(REG) / 4 -> C(PRn.WORDNO)
C(REG) modulo 4 -> C(PRn.CHAR)
If A = 1, then
C(PRn.WORDNO) + ADDRESS +
(C(REG) + C(PRn.CHAR)) / 4 -> C(PRn.WORDNO)
(C(PRn.CHAR) + C(REG)) modulo 4 -> C(PRn.CHAR)
0000 -> C(PRn.BITNO)

MODIFICATIONS: None except AU, QU, AL, QL, and Xn

INDICATORS: None affected

NOTES: The steps described in SUMMARY define special 9-bit addition arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), and C(PRn.CHAR).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

ABD Add Bit Displacement to Address Register 503 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

SUMMARY: If A = 0, then
ADDRESS + C(REG) / 36 -> C(PRn.WORDNO)

(C(REG) modulo 36) / 9 -> C(PRn.CHAR)

C(REG) modulo 9 -> C(PRn.BITNO)

If A = 1, then

C(PRn.WORDNO) + ADDRESS + (9 * C(PRn.CHAR) + 36 * C(REG) + C(PRn.BITNO)) / 36 -> C(PRn.WORDNO)

EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

$(9 * C(\text{PRn.CHAR}) + 36 * C(\text{REG}) + C(\text{PRn.BITNO})) \text{ modulo } 36 / 9 \rightarrow C(\text{PRn.CHAR})$

$(9 * C(\text{PRn.CHAR}) + 36 * C(\text{REG}) + C(\text{PRn.BITNO})) \text{ modulo } 9 \rightarrow C(\text{PRn.BITNO})$

MODIFICATIONS: None except AU, QU, AL, QL, or XN

INDICATORS: None affected

NOTES: The steps described in SUMMARY define special bit addition arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

AND Add Word Displacement to Address Register 507 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction (See Figure 2-12).

SUMMARY: If A = 0, then
ADDRESS + C(REG) → C(PRn.WORDNO)
If A = 1, then
C(PRn.WORDNO) + ADDRESS + C(REG) → C(PRn.WORDNO)
00 → C(PRn.CHAR)
0000 → C(PRn.BITNO)

MODIFICATIONS: None except AU, QU, AL, QL, and Xn

INDICATORS: None affected

NOTES: The use of an Address Register is inherent; the value of

bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

S48D Subtract 4-bit Displacement from Address Register 522 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

SUMMARY: If A = 0, then

- $(ADDRESS + C(REG) / 4) \rightarrow C(PRn.WORDNO)$
- $C(REG) \text{ modulo } 4 \rightarrow C(PRn.CHAR)$
- $4 * (C(REG) \text{ modulo } 2) + 1 \rightarrow C(PRn.BITNO)$

If A = 1, then

- $C(PRn.WORDNO) - ADDRESS + (9 * C(PRn.CHAR) - 4 * C(REG) + C(PRn.BITNO)) / 36 \rightarrow C(PRn.WORDNO)$
- $((9 * C(PRn.CHAR) - 4 * C(REG) + C(PRn.BITNO)) \text{ modulo } 36) / 9 \rightarrow C(PRn.CHAR)$
- $4 * (C(PRn.CHAR) - 2 * C(REG) + C(PRn.BITNO) / 4) \text{ modulo } 2 + 1 \rightarrow C(PRn.BITNO)$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn

INDICATORS: None affected

NOTES: The steps described in SUMMARY define special 4-bit subtraction arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal procedure Fault.

S68D Subtract 6-Bit Displacement from Address Register 521(1)

FORMAT: EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

SUMMARY: If A = 0, then

- $(ADDRESS + C(REG) / 6) \rightarrow C(PRn.WORDNO)$
- $((6 * C(REG)) \text{ modulo } 36) / 9 \rightarrow C(PRn.CHAR)$
- $((6 * C(REG)) \text{ modulo } 9) \rightarrow C(PRn.BITNO)$

EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

If A = 1, then

$$C(\text{PRn.WORDNO}) - \text{ADDRESS} + (9 * C(\text{PRn.CHAR}) - 6 * C(\text{REG}) + C(\text{PRn.BITNO})) / 36 \rightarrow C(\text{PRn.WORDNO})$$

$$((9 * C(\text{PRn.CHAR}) - 6 * C(\text{REG}) + C(\text{PRn.BITNO})) \text{ modulo } 36) / 9 \rightarrow C(\text{PRn.CHAR})$$

$$(9 * C(\text{PRn.CHAR}) - 6 * C(\text{REG}) + C(\text{PRn.BITNO})) \text{ modulo } 9 \rightarrow C(\text{PRn.BITNO})$$

MODIFICATIONS: None except AU, QU, AL, QL, and Xn

INDICATORS: None Affected

NOTES: The steps described in SUMMARY define special 6-bit subtraction arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

S9BD

Subtract 9-Bit Displacement from Address Register 520 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction (See Figure 2-12).

SUMMARY: If A = 0, then

$$- (\text{ADDRESS} + C(\text{REG}) / 4) \rightarrow C(\text{PRn.WORDNO})$$

$$- C(\text{REG}) \text{ modulo } 4 \rightarrow C(\text{PRn.CHAR})$$

If A = 1, then

$$C(\text{PRn.WORDNO}) - \text{ADDRESS} + (C(\text{PRn.CHAR}) - C(\text{REG})) / 4 \rightarrow C(\text{PR.CHAR})$$

$$(C(\text{PRn.CHAR}) - C(\text{REG})) \text{ modulo } 4 \rightarrow C(\text{PRn.CHAR})$$

$$0000 \rightarrow C(\text{PRn.BITNO})$$

MODIFICATIONS: None except AU, QU, AL, QU, or Xn

INDICATORS: None affected

NOTES: The steps described in SUMMARY define special 9-bit subtraction arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), and C(PRn.CHAR).

EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SBD Subtract Bit Displacement from Address Register 523 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

SUMMARY:

If A = 0, then

- $(\text{ADDRESS} + \text{C}(\text{REG}) / 36) \rightarrow \text{C}(\text{PRn.WORDNO})$

- $(\text{C}(\text{REG}) \text{ modulo } 36) / 9 \rightarrow \text{C}(\text{PRn.CHAR})$

- $\text{C}(\text{REG}) \text{ modulo } 9 \rightarrow \text{C}(\text{PRn.BITNO})$

If A = 1, then

$\text{C}(\text{PRn.WORDNO}) - \text{ADDRESS} + (9 * \text{C}(\text{PRn.CHAR}) - 36 * \text{C}(\text{REG}) + \text{C}(\text{PRn.BITNO})) / 36 \rightarrow \text{C}(\text{PRn.WORDNO})$

$((9 * \text{C}(\text{PRn.CHAR}) - 36 * \text{C}(\text{REG}) + \text{C}(\text{PRn.BITNO})) \text{ modulo } 36) / 9 \rightarrow \text{C}(\text{PRn.CHAR})$

$(9 * \text{C}(\text{PRn.CHAR}) - 36 * \text{C}(\text{REG}) + \text{C}(\text{PRn.BITNO})) \text{ modulo } 9 \rightarrow \text{C}(\text{PRn.BITNO})$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn

INDICATORS: None affected

NOTES:

The steps described in SUMMARY define special bit subtraction arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SWD Subtract Word Displacement from Address Register 527 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

SUMMARY: If A = 0, then
 - (ADDRESS + C(REG)) -> C(PRn.WORDNO)
 If A = 1, then
 C(PRn.WORDNO) - (ADDRESS + C(REG)) -> C(PRn.WORDNO)
 00 -> C(PRn.CHAR)
 0000 -> C(PRn.BITNO)

MODIFICATIONS: None except AU, QU, AL, QL, or Xn

INDICATORS: None Affected

NOTES: The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Alphanumeric Compare

CMPC

Compare Alphanumeric Character Strings

106 (1)

FORMAT:

0	0 0 1 1	1 1	2 2 2 2 2	2 2 2	3
0	8 9 0 1	7 8	0 1 2 3 4	7 8 9	5
	FILL	MF2	106 (1)	MF1	
	9 2	7		10 1	7
	Y-char ₁	CN1	TA1	N1	
	Y-char ₂	CN2	10 0 0 1	N2	
		18	3 2 1		12

Figure 2-13 Compare Alphanumeric Strings (CMPC) EIS Multi-Word Instruction Format

- FILL Fill character for string extension
- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- I Program Interrupt inhibit bit
- Y-char₁ Address of "left-hand" string
- CN1 First character position of "left-hand" string
- TA1 Data type of "left-hand" string
- N1 Length of "left-hand" string
- Y-char₂ Address of "right-hand" string
- CN2 First character position of "right-hand" string
- N2 Length of "right-hand" string

ALM Coding Format:

cmcp (MF1),(MF2)[,fill(octal expression)]
 descna Y-char₁[(CN1)],N1 n = 4, 6, or 9 (TA1 = 2, 1, or 0)

descna Y-char₂[(CN2)],N2 n = 4, 6, or 9 (TA2 is ignored)

SUMMARY: For i = 1, 2, ..., minimum (N1,N2)
 C(Y-char₁)_{i-1} :: C(Y-char₂)_{i-1}

If $N1 < N2$, then for $i = N1+1, N1+2, \dots, N2$

$C(FILL) :: C(Y-char_2)_{i-1}$

If $N2 < N1$, then for $i = N2+1, N2+2, \dots, N1$

$C(Y-char_1)_{i-1} :: C(FILL)$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y-char_1)_i = C(Y-char_2)_i$ for all i , then ON;
otherwise, OFF

Carry If $C(Y-char_1)_i < C(Y-char_2)_i$ for any i , then OFF;
otherwise ON

NOTES:

Both strings are treated as the data type given for the "left-hand" string, TA1. The data type given for the "right-hand" string, TA2, is ignored.

Comparison is made on full 9-bit fields. If the given data type is not 9-bit (TA1 \neq 0), then characters from $C(Y-char_1)$ and $C(Y-char_2)$ are high-order zero filled. All 9 bits of C(FILL) are used.

Instruction execution proceeds until an inequality is found or the larger string length count is exhausted.

If MFk.RL = 1, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MFk.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SCD

Scan Characters Double

120 (1)

FORMAT:

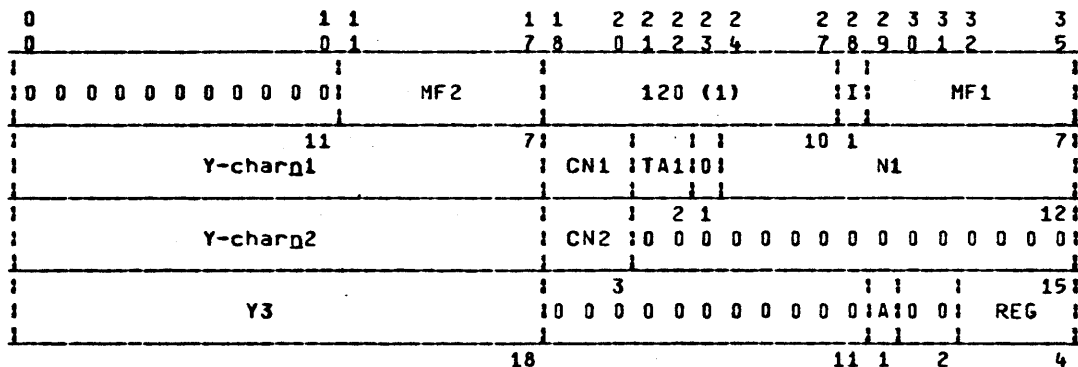


Figure 2-14 Scan Characters Double (SCD) EIS Multi-Word Instruction Format

- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- I Program Interrupt inhibit bit
- Y-char_n1 Address of string
- CN1 First character position of string
- TA1 Data type of string
- N1 Length of string
- Y-char_n2 Address of test character pair
- CN2 First character position of test character pair
- Y3 Address of compare count word
- A Indirect via Pointer Register flag for Y3
- REG Register modifier for Y3

ALM Coding Format:

scd	{MF1},{MF2}	
desc _{na}	Y-char _n 1[{CN1}],N1	n = 4, 6, or 9 (TA1 = 2, 1, or 0)
desc _{pa}	Y-char _n 2[{CN2}]	n = 4, 6, or 9 (TA2 is ignored)
arg	Y3[,tag]	

SUMMARY: For $i = 1, 2, \dots, N1-1$
 $C(Y\text{-char}1)_{i-1,i} :: C(Y\text{-char}2)_{0,1}$
 On instruction completion
 $00\dots0 \rightarrow C(Y3)_{0,11}$
 $i \rightarrow C(Y3)_{12,35}$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and REG
 None except DU, AU, QU, AL, QL, or Xn for MF2

INDICATORS: (Indicators not listed are not affected)

Tally If the string length count is exhausted without a match,
Runout or if $N1 = 1$, then ON; otherwise OFF

NOTES: Both the string and the test character pair are treated as the data type given for the string, TA1. The data type given for the test character pair, TA2, is ignored.

Instruction execution proceeds until a character pair match is found or the string length count is exhausted.

If $MFk.RL = 1$, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFk.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

If $MF2.ID = 0$ and $MF2.REG = DU$, then the second word following the Instruction Word does not contain an Operand Descriptor for the test character pair; instead, it contains the test character pair as a Direct Upper operand in bits 0,17.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SCDR Scan Characters Double in Reverse 121 (1)

FORMAT: Same as Scan Characters Double (SCD) (See Figure 2-14).

SUMMARY: For $i = 1, 2, \dots, N1-1$
 $C(Y\text{-char}1)_{N1-i-1,N1-i} :: C(Y\text{-char}2)_{0,1}$

EIS - ALPHANUMERIC COMPARE

On instruction completion

00...0 -> C(Y3)0,11

i -> C(Y3)12,35

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and REG
None except DU, AU, QU, AL, QL, or Xn for MF2

INDICATORS: (Indicators not listed are not affected)

Tally If the string length count is exhausted without a match,
Runout or if N1 = 1, then ON; otherwise OFF

NOTES: Both the string and the test character pair are treated as
the data type given for the string, TA1. The data type
given for the test character pair, TA2, is ignored.

Instruction execution proceeds until a character pair
match is found or the string length count is exhausted.

If MF_k.RL = 1, then N_k does not contain the operand
length; instead, it contains a register code for a
register holding the operand length.

If MF_k.ID = 1, then the kth word following the Instruction
Word does not contain an Operand Descriptor; instead, it
contains an Indirect Pointer to the Operand Descriptor.

If MF2.ID = 0 and MF2.REG = DU, then the second word
following the Instruction Word does not contain an Operand
Descriptor for the test character pair; instead, it
contains the test character pair as a Direct Upper operand
in bits 0,17.

Attempted execution with XED causes an Illegal Procedure
Fault.

Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

SCM

Scan with Mask

124 (1)

FORMAT:

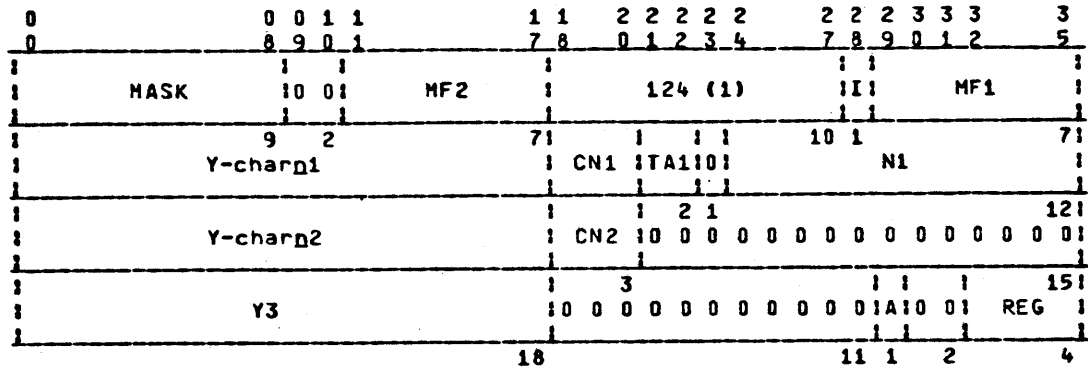


Figure 2-15 Scan with Mask (SCM) EIS Multi-Word Instruction Format

- MASK Comparison bit mask
- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- I Program Interrupt inhibit bit
- Y-char_{D1} Address of string
- CN1 First character position of string
- TA1 Data type of string
- N1 Length of string
- Y-char_{D2} Address of test character
- CN2 First character position of test character
- Y3 Address of compare count word
- A Indirect via Pointer Register flag for Y3
- REG Register modifier for Y3

ALM Coding Format:

scm	(MF1),(MF2)[,mask(octalexpression)]
desc _{pa}	Y-char _{D1} [(CN1)],N1 D = 4, 6, or 9 (TA1 = 2, 1, or 0)
desc _{pb}	Y-char _{D2} [(CN2)] D = 4, 6, or 9 (TA2 is ignored)
arg	Y3[,tag]

EIS - ALPHANUMERIC COMPARE

SUMMARY: For characters $i = 1, 2, \dots, N1$
For bits $j = 0, 1, \dots, 8$
 $C(Z)j = \sim C(MASK)j \&$
 $((C(Y-charn1)i-1)j \oplus (C(Y-charn2)i)j)$
If $C(Z)0,8 = 00\dots0$, then
 $00\dots0 \rightarrow C(Y3)0,11$
 $i \rightarrow C(Y3)12,35$
otherwise, continue scan of $C(Y-charn1)$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and REG
None except DU, AU, QU, AL, QL, or Xn for MF2

INDICATORS: (Indicators not listed are not affected)

Tally Runout If the string length count exhausts, then ON;
otherwise, OFF

NOTES: Both the string and the test character are treated as the data type given for the string, TA1. The data type given for the test character, TA2, is ignored.

Instruction execution proceeds until a masked character match is found or the string length count is exhausted.

Masking and comparison is done on full 9-bit fields. If the given data type is not 9-bit ($TA1 \neq 0$), then characters from $C(Y-charn1)$ and $C(Y-charn2)$ are high-order zero filled. All 9 bits of $C(MASK)$ are used.

If $MF1.RL = 1$, then $N1$ does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFk.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

If $MF2.ID = 0$ and $MF2.REG = DU$, then the second word following the Instruction Word does not contain an Operand Descriptor for the test character; instead, it contains the test character as a Direct Upper operand in bits 0,8.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SCMR

Scan with Mask in Reverse

125 (1)

FORMAT:

Same as Scan with Mask (SCM) (See Figure 2-15).

SUMMARY:For characters $i = 1, 2, \dots, N1$ For bits $j = 0, 1, \dots, 8$

$$C(Z)_{j,8} = \sim C(\text{MASK})_{j,8} \& \{ (C(\text{Y-char}_{i1})_{N1-i}) \oplus (C(\text{Y-char}_{i2})_{j,8}) \}$$

If $C(Z)_{j,8} = 00\dots 0$, then $00\dots 0 \rightarrow C(Y3)_{j,11}$ $i \rightarrow C(Y3)_{12,35}$ otherwise, continue scan of $C(\text{Y-char}_{i1})$ **MODIFICATIONS:**None except AU, QU, AL, QL, or Xn for MF1 and REG
None except DU, AU, QU, AL, QL, or Xn for MF2**INDICATORS:**

(Indicators not listed are not affected)

Tally
RunoutIf the string length count exhausts, then ON;
otherwise, OFF**NOTES:**

Both the string and the test character are treated as the data type given for the string, TA1. The data type given for the test character, TA2, is ignored.

Instruction execution proceeds until a masked character match is found or the string length count is exhausted.

Masking and comparison is done on full 9-bit fields. If the given data type is not 9-bit ($TA1 \neq 0$), then characters from $C(\text{Y-char}_{i1})$ and $C(\text{Y-char}_{i2})$ are high-order zero filled. All 9 bits of $C(\text{MASK})$ are used.If $MF1.RL = 1$, then $N1$ does not contain the operand length; instead, it contains a register code for a register holding the operand length.If $MFk.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.If $MF2.ID = 0$ and $MF2.REG = DU$, then the second word following the Instruction Word does not contain an Operand Descriptor for the test character; instead, it contains the test character as a Direct Upper operand in bits 0,8.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TCT

Test Character and Translate

164 (1)

FORMAT:

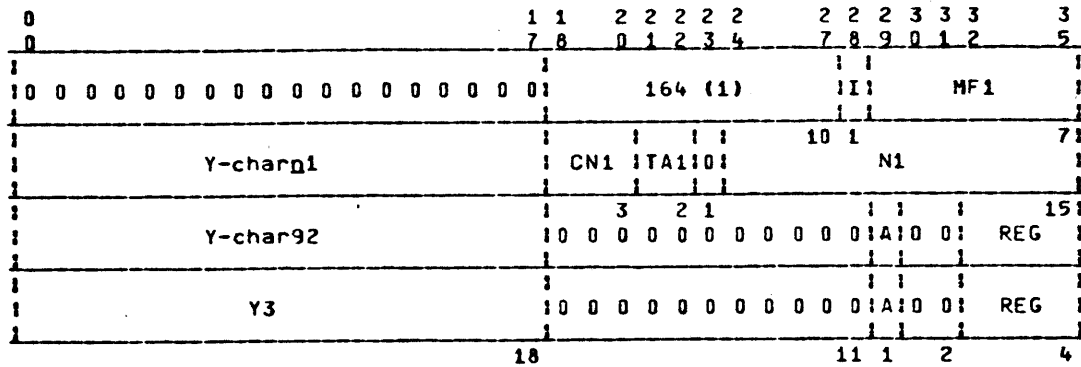


Figure 2-16 Test Character and Translate (TCT) EIS Multi-Word Instruction Format

- MF1 Modification Field for Operand Descriptor 1
- I Program Interrupt inhibit bit
- Y-char_n1 Address of string
- CN1 First character position of string
- TA1 Data type of string
- N1 Length of string
- Y-char92 Address of character translation table
- Y3 Address of result word
- A Indirect via Pointer Register flag for Y2 and Y3
- REG Register modifier for Y2 and Y3

ALM Coding Format:

tct (MF1)
 desc_na Y-char_n1[(CN1)],N1 n = 4, 6, or 9 (TA1 = 2, 1, or 0)
 arg Y-char92[,tag]
 arg Y3[,tag]

SUMMARY:

For i = 1, 2, ..., N1

$$m = C(Y\text{-char}_{n1})_{i-1}$$

If $C(Y\text{-char}_{92})_{m-1} \neq 00\dots 0$, then

$$C(Y\text{-char}_{92})_{m-1} \rightarrow C(Y3)_{0,8}$$

000 -> C(Y3)9,11

i -> C(Y3)12,35

otherwise, continue scan of C(y-char_{n1})

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and REG

INDICATORS: (Indicators not listed are not affected)

Tally If the string length count exhausts, then ON;
Runout otherwise, OFF

NOTES: If the data type of the string to be scanned is not 9-bit
(IA1 ≠ 0), then characters from C(Y-char_{n1})_i are
high-order zero filled in forming the table index, m.

Instruction execution proceeds until a non-zero table
entry is found or the string length count is exhausted.

If MF1.RL = 1, then N1 does not contain the operand
length; instead, it contains a register code for a
register holding the operand length.

If MF1.ID = 1, then the first word following the
Instruction Word does not contain an Operand Descriptor;
instead, it contains an Indirect Pointer to the Operand
Descriptor.

Attempted execution with XED causes an Illegal Procedure
Fault.

Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

TCTR Test Character and Translate in Reverse 165 (1)

FORMAT: Same as Test Character and Translate (TCT)
(See Figure 2-16).

SUMMARY: For i = 1, 2, ..., N1
 m = C(Y-char_{n1})_{N1-i}
 If C(Y-char₉₂)_{m-1} ≠ 00...0, then

C(Y-char₉₂)_{m-1} -> C(Y3)0,8

000 -> C(Y3)9,11

i -> C(Y3)12,35

otherwise, continue scan of C(y-char_{n1})

EIS - ALPHANUMERIC COMPARE

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and REG

INDICATORS: (Indicators not listed are not affected)

Tally Runout If the string length count exhausts, then ON; otherwise, OFF

NOTES: If the data type of the string to be scanned is not 9-bit (TA1 ≠ 0), then characters from C(Y-char_i) are high-order zero filled in forming the table index, m.

Instruction execution proceeds until a non-zero table entry is found or the string length count is exhausted.

If MF1.RL = 1, then N1 does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF1.ID = 1, then the first word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Alphanumeric Move

MLR Move Alphanumeric Left to Right 100 (1)

FORMAT:

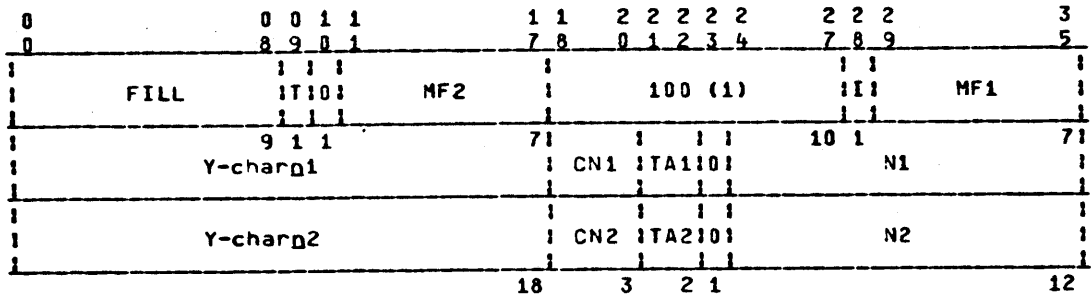


Figure 2-17 Move Alphanumeric Left to Right (MLR) EIS Multi-Word Instruction Format

- FILL Fill character for string extension
- T Truncation Fault enable bit
- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- Y-char₁ Address of sending string
- CN1 First character position of sending string
- TA1 Data type of sending string
- N1 Length of sending string
- Y-char₂ Address of receiving string
- CN2 First character position of receiving string
- TA2 Data type of receiving string
- N2 Length of receiving string

ALM Coding Format:

mlr (MF1),(MF2)[,fill(octalexpression)][,enablefault]
 desc₁ Y-char₁[(CN1)],N1 D = 4, 6, or 9 (TA1 = 2, 1, or 0)
 desc₂ Y-char₂[(CN2)],N2 D = 4, 6, or 9 (TA2 = 2, 1, or 0)

SUMMARY: For i = 1, 2, ..., minimum (N1,N2)
 C(Y-char₁)_{i-1} -> C(Y-char₂)_{i-1}

EIS - ALPHANUMERIC MOVE

If $N1 < N2$, then for $i = N1+1, N1+2, \dots, N2$

$C(\text{FILL}) \rightarrow C(\text{Y-char}_2)_{i-1}$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Truncation If $N1 > N2$ then ON; otherwise OFF

NOTES: If data types are dissimilar ($TA1 \neq TA2$), each character is high-order truncated or zero filled, as appropriate, as it is moved. No character conversion takes place.

If $N1 > N2$, then $(N1-N2)$ trailing characters of $C(\text{Y-char}_1)$ are not moved and the Truncation indicator is set ON.

If $N1 < N2$ and $TA2 = 2$ (4-bit data) or 1 (6-bit data), then FILL characters are high-order truncated as they are moved to $C(\text{Y-char}_2)$. No character conversion takes place.

If $N1 < N2$, $C(\text{FILL})_0 = 1$, $TA1 = 1$, and $TA2 = 2$, then $C(\text{Y-char}_1)_{N1}$ is examined for a GBCD overpunch sign. If a negative overpunch sign is found, then the minus sign character is placed in $C(\text{Y-char}_2)_{N2}$; otherwise, a plus sign character is placed in $C(\text{Y-char}_2)_{N2}$.

If $MF_k.RL = 1$, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MF_k.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(\text{Y-char}_1)$ and $C(\text{Y-char}_2)$ may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string $C(\text{Y-char}_1)$ data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlaid string ($C(\text{Y-char}_2)$) is not returned to main store until the unit of Y-block8 words is

filled or the instruction completes.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

MRL Move Alphanumeric Right to Left 101 (1)

FORMAT: Same as Move Alphanumeric Left to Right (MLR) (See Figure 2-17).

SUMMARY: For $i = 1, 2, \dots$, minimum (N1,N2)
 $C(Y\text{-char}_{\eta 1})N1-i \rightarrow C(Y\text{-char}_{\eta 2})N2-i$
 If $N1 < N2$, then for $i = N1+1, N2+1, \dots, N2$
 $C(FILL) \rightarrow C(Y\text{-char}_{\eta 2})N2-i$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Truncation If $N1 > N2$ then ON; otherwise OFF

NOTES: If data types are dissimilar ($TA1 \neq TA2$), each character is high-order truncated or zero filled, as appropriate, as it is moved. No character conversion takes place.

If $N1 > N2$, then (N1-N2) leading characters of $C(Y\text{-char}_{\eta 1})$ are not moved and the Truncation indicator is set ON.

If $N1 < N2$ and $TA2 = 2$ (4-bit data) or 1 (6-bit data), then FILL characters are high-order truncated as they are moved to $C(Y\text{-char}_{\eta 2})$. No character conversion takes place.

If $MFk.RL = 1$, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFk.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char}_{\eta 1})$ and $C(Y\text{-char}_{\eta 2})$ may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be

exercised in the construction of the Operand Descriptors so that sending string $C(Y\text{-char}_{\eta 1})$ data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlaid string $C(Y\text{-char}_{\eta 2})$ is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

EIS - ALPHANUMERIC MOVE

If T = 1 and the Truncation Indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

MVE Move Alphanumeric Edited 020 (1)

FORMAT:

0 0 0	0 0 1 1	1 1	2 2 2 2 2	2 2 2	3
0 1 2	8 9 0 1	7 8	0 1 2 3 4	7 8 9	5
I	MF3	MF2	020 (1)	I	MF1
2	Y-char ₀₁	7 2	CN1 TA1 N1	10 1	7
	Y-char ₉₂		CN2 TA1 N2		
	Y-char ₀₃		CN3 TA3 N3		
		18	3 2 1		12

Figure 2-18 Move Alphanumeric Edited (MVE) EIS Multi-Word Instruction Format

- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- MF3 Modification Field for Operand Descriptor 3
- I Program Interrupt inhibit bit
- Y-char₀₁ Address of sending string
- CN1 First character position of sending string
- TA1 Data type of sending string
- N1 Length of sending string

- Y-char₉₂ Address of MOP control string
- CN2 First character position of MOP control string
- N2 Length of MOP control string
- Y-char₀₃ Address of receiving string

CN3 First character position of receiving string
 TA3 Data type of receiving string
 N3 Length of receiving string

ALM Coding Format:

mve	(MF1),(MF2),(MF3)	
desc _a	Y-char _{n1} [(CN1)],N1	n = 4, 6, or 9 (TA1 = 2, 1, or 0)
desc _{9a}	Y-char ₉₂ [(CN2)],N2	
desc _a	Y-char _{n3} [(CN3)],N3	n = 4, 6, or 9 (TA3 = 2, 1, or 0)

SUMMARY: C(Y-char_{n1}) -> C(Y-char_{n3}) under C(Y-char₉₂) MOP control
 See "Micro Operations for Edit Instructions" later in this section for details of editing under MOP control.

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1, MF2, and MF3

INDICATORS: None affected

NOTES: If data types are dissimilar (TA1 ≠ TA3), each character of C(Y-char_{n1}) is high-order truncated or zero filled, as appropriate, as it is moved. No character conversion takes place.

If the data type of the receiving string is not 9-bit (TA3 ≠ 0), then Insertion Characters are high-order truncated as they are inserted.

The maximum string length is 63. The count fields N1, N2, and N3 are treated as modulo 64 numbers.

The instruction completes normally only if N3 = minimum (N1,N2,N3), that is, if the receiving string is the first to exhaust; otherwise, an Illegal Procedure Fault occurs.

If MF_k.RL = 1, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF_k.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char_{n1}) and C(Y-char_{n3}) may be overlapping strings; no check is made. This feature is useful for replication of

substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string (C(Y-char_{n1})) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block₈ words and that the overlaid string (C(Y-char_{n3})) is not returned to main store until the unit of Y-block₈ words is

FORMAT:

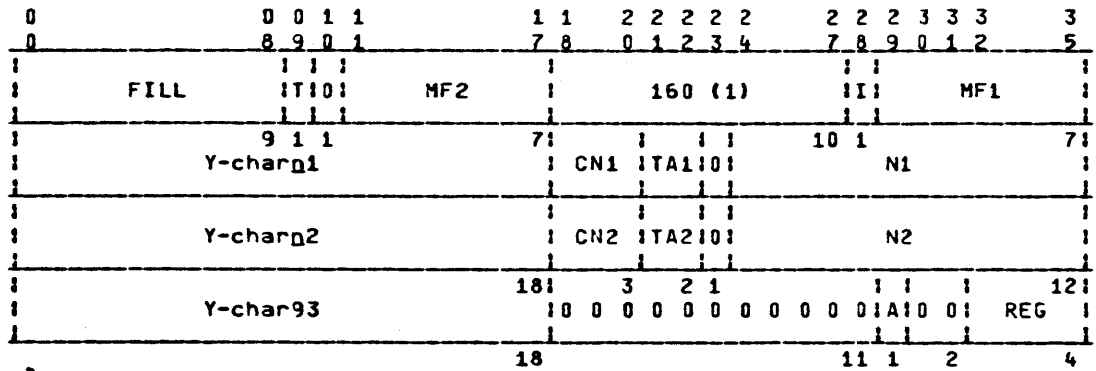


Figure 2-19 Move Alphanumeric with Translation (MVT) EIS Multi-Word Instruction Format

- FILL Fill character for string extension
- T Truncation Fault enable bit
- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- Y-char₀₁ Address of sending string
- CN1 First character position of sending string
- TA1 Data type of sending string
- N1 Length of sending string

- Y-char₀₂ Address of receiving string
- CN2 First character position of receiving string
- TA2 Data type of receiving string
- N2 Length of receiving string
- Y-char₉₃ Address of character translation table

A Indirect via Pointer Register flag for Y-char93

REG Register modifier for Y-char93

ALM Coding Format:

```

mvt      (MF1),(MF2)[,fill(octal expression)][,enablefault]
descpa   Y-charn1[(CN1)],N1      n = 4, 6, or 9 (TA1 = 2, 1, or 0)
descpa   Y-charn2[(CN2)],N2      n = 4, 6, or 9 (TA2 = 2, 1, or 0)
arg      Y-char93[,tag]

```

SUMMARY: For $i = 1, 2, \dots$, minimum (N1,N2)

$$m = C(Y\text{-char}_{n1})_{i-1}$$

$$C(Y\text{-char}_{93})_{m-1} \rightarrow C(Y\text{-char}_{n2})_{i-1}$$

If $N1 < N2$, then for $i = N1+1, N1+2, \dots, N2$

$$m = C(FILL)$$

$$C(Y\text{-char}_{93})_{m-1} \rightarrow C(Y\text{-char}_{n2})_{i-1}$$

MODIFICATIONS: None except AU, QU, AL, QL, and Xn for MF1, MF2, and REG

INDICATORS: (Indicators not listed are not affected)

Truncation If $N1 > N2$ then ON; otherwise OFF

NOTES: If the data type of the receiving field is not 9-bit (TA2 \neq 0), then characters from C(Y-char93) are high-order truncated, as appropriate, as they are removed.

If the data type of the sending field is not 9-bit (TA1 \neq 0), then characters from C(Y-char_{n1}) are high-order zero filled when forming the table index.

If $N1 > N2$, then (N1-N2) leading characters of C(Y-char_{n1}) are not moved and the Truncation indicator is set ON.

If MF_k.RL = 1, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF_k.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char_{n1}) and C(Y-char_{n2}) may be over-lapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string (C(Y-char_{n1})) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in

EIS - ALPHANUMERIC MOVE

unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlaid string (C(Y-char α 2)) is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Numeric Compare

CMPN Compare Numeric 303 (1)

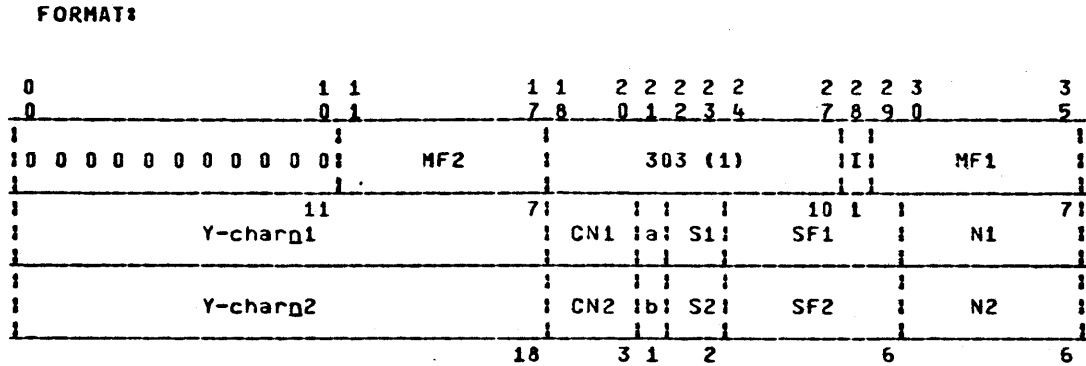


Figure 2-20 Compare Numeric (CMPN) EIS Multi-Word Instruction Format

key

- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- I Program Interrupt inhibit bit
- Y-char₁ Address of "left-hand" number
- CN1 First character position of "left-hand" number
- a TN1 Data type of "left-hand" number
- S1 Sign and decimal type of "left-hand" number
- SF1 Scaling factor of "left-hand" number
- N1 Length of "left-hand" number
- Y-char₂ Address of "right-hand" number
- CN2 First character position of "right-hand" number
- b TN2 Data type of "right-hand" number
- S2 Sign and decimal type of "right-hand" number

- SF2 Scaling factor of "right-hand" number
- N2 Length of "right-hand" string

EIS - NUMERIC COMPARE

ALM Coding Format:

cmpr	(MF1),(MF2)	
descp[fl,ls,ns,ts]	Y-char ₁ [(CN1)],N1,SF1	D = 4 or 9
descp[fl,ls,ns,ts]	Y-char ₂ [(CN2)],N2,SF2	D = 4 or 9

SUMMARY: C(Y-char₁) :: C(Y-char₂) as numeric values

MODIFICATIONS: None except AU, QU, AL, QI, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Y-char ₁) = C(Y-char ₂), then ON; otherwise OFF
Negative	If C(Y-char ₁) > C(Y-char ₂), then ON; otherwise OFF
Carry	If !C(Y-char ₁)! > !C(Y-char ₂)!, then OFF, otherwise ON

NOTES: Comparison is made on 4-bit numeric values contained in each character of C(Y-char_k). If either given data type is 9-bit (TN_k = 0), characters from C(Y-char_{9k}) are high-order truncated to 4 bits before comparison.

Sign characters are located according to information in CN_k, S_k, and N_k and interpreted as 4-bit fields; 9-bit sign characters are high-order truncated before interpretation. The sign character 15 (octal) is interpreted as a minus sign; all other legal sign characters are interpreted as plus signs.

The position of the decimal point in C(Y-char_k) is determined from information in CN_k, S_k, SF_k, and N_k.

Comparison begins at the decimal position corresponding to the first digit of the operand with the larger number of integer digits and ends with the last digit of the operand with the larger number of fraction digits.

Four-bit numeric zeros are used to represent digits to the left of the first given digit of the operand with the smaller number of integer digits.

Four-bit numeric zeros are used to represent digits to the right of the last given digit of the operand with the smaller number of fraction digits.

Instruction execution proceeds until an inequality is found or the larger string length count is exhausted.

If MF_k.RL = 1, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF_k.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - NUMERIC MOVE

EIS - Numeric Move

MVN

Move Numeric

300 (1)

FORMAT:

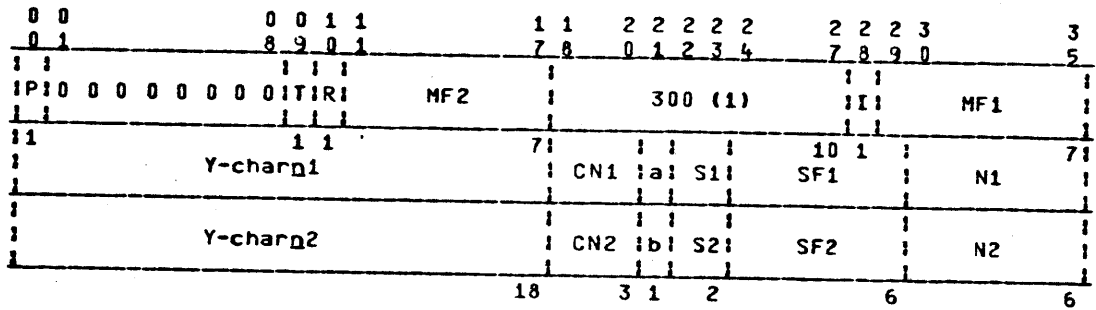


Figure 2-21 Move Numeric (MVN) EIS Multi-Word Instruction Format

key

- P 4-bit data sign character control
- T Truncation Fault enable bit
- R Rounding flag
- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- I Program Interrupt inhibit bit
- Y-char₀₁ Address of sending number
- CN1 First character position of sending number
- a TN1 Data type of sending number
- S1 Sign and decimal type of sending number
- SF1 Scaling factor of sending number
- N1 Length of sending number
- Y-char₀₂ Address of receiving number
- CN2 First character position of receiving number
- b TN2 Data type of receiving number
- S2 Sign and decimal type of receiving number
- SF2 Scaling factor of receiving number
- N2 Length of receiving string

ALM Coding Format:

```

mvn                (MF1),(MF2)[,enable fault][,round]
descn[f1,ls,ns,fs] Y-charn1[(CN1)],N1,SF1      D = 4 or 9
descn[f1,ls,ns,fs] Y-charn2[(CN2)],N2,SF2      D = 4 or 9

```

SUMMARY C(Y-charn1) converted and/or rescaled -> C(Y-charn2)

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If C(Y-charn2) = decimal 0, then ON; otherwise OFF

Negative If a minus sign character is moved to C(Y-charn2), then ON; otherwise OFF

Truncation If low-order digit truncation occurs without rounding, then ON; otherwise OFF

Overflow If fixed point integer overflow occurs, then ON; otherwise unchanged. (See NOTES)

Exponent Overflow If exponent of floating point result exceeds +127, then ON; otherwise unchanged.

Exponent Underflow If exponent of floating point result is less than -128, then ON; otherwise unchanged.

NOTES:

If data types are dissimilar (TN1 ≠ TN2), each character is high-order truncated or filled, as appropriate, as it is moved. The fill data used is "00011"b for digit characters and "00010"b for sign characters.

If TN2 and S2 specify a 4-bit signed number and S2 specify a 4-bit signed number and P = 1, then a legal plus sign character in C(Y-charn1) is converted to 13 (octal) as it is moved.

If N2 is not large enough to hold the integer part of C(Y-charn1) as rescaled by SF2, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If N2 is not large enough to hold all the given digits of

C(Y-charn1) as rescaled by SF2 and R = 0, then a truncation condition exists; data movement stops when C(Y-charn2) is filled and the Truncation indicator is set ON. If R = 1, then the last digit moved is rounded according to the absolute value of the remaining digits of C(Y-charn1) and the instruction completes normally.

If MFk.RL = 1, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFk.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y-char_1)$ and $C(Y-char_2)$ may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string $C(Y-char_1)$ data is not inadvertently destroyed. Difficulties may be encountered because of scaling factors and the special treatment of sign characters and floating point exponents.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlaid string $C(Y-char_2)$ is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

MVNE

Move Numeric Edited

024 (1)

FORMAT:

0 0 0	0 0 1 1	1 1	2 2 2 2 2	2 2 2	3
0 1 2	8 9 0 1	7 8	0 1 2 3 4	7 8 9	5
10 01	MF3	10 01	MF2	024 (1)	MF1
	2	7 2	71	1 1 1	10 1 1
	Y-char ₀₁		CN1	1a1 S1	0 0 0 0 0 0
					N1
	Y-char ₉₂		CN2	1 1 2	6
			CN2	1 0 0 0 0 0 0 0 0 0	N2
	Y-char ₀₃		CN1	1 1 1	9
			CN1	1TA3	1 0 1 0 0 0 0 0 0 0
					N3
		18	3	2 1	6

Figure 2-22 Move Numeric Edited (MVNE) EIS Multi-Word Instruction Format

- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- MF3 Modification Field for Operand Descriptor 3
- I Program Interrupt inhibit bit
- Y-char₀₁ Address of sending string
- CN1 First character position of sending string
- TN1 Data type of sending string
- S1 Sign and decimal type of sending string
- N1 Length of sending string
- Y-char₉₂ Address of MOP control string
- CN2 First character position of MOP control string
- N2 Length of MOP control string
- Y-char₀₃ Address of receiving string
- CN3 First character position of receiving string

- TA3 Data type of receiving string
- N3 Length of receiving string

EIS - NUMERIC MOVE

ALM Coding Format:

mvne	(MF1),(MF2),(MF3)	
descn[ff,ls,ns,ts]	Y-char _{n1} [(CN1)],N1	D = 4 or 9
desc9a	Y-char ₉₂ [(CN2)],N2	
descna	Y-char _{n3} [(CN3)],N3	D = 4, 6, or 9

SUMMARY: C(Y-char_{n1}) -> C(Y-char_{n3}) under C(Y-char₉₂) MOP control
See "Micro Operations for Edit Instructions" later in this section for details of editing under MOP control.

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1, MF2, and MF3

INDICATORS: None affected

NOTES: If data types are dissimilar (TA1 ≠ TA3), each character of C(Y-char_{n1}) is high-order truncated or zero filled, as appropriate, as it is moved. No character conversion takes place.

If the data type of the receiving string is not 9-bit (TA3 ≠ 0), then Insertion Characters are high-order truncated as they are inserted.

The maximum string length is 63. The count fields N1, N2, and N3 are treated as modulo 64 numbers.

The instruction completes normally only if N3 = minimum (N1,N2,N3), that is, if the receiving string is the first to exhaust; otherwise, an Illegal Procedure Fault occurs.

If MF_k.RL = 1, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF_k.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char_{n1}) and C(Y-char_{n3}) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string (C(Y-char_{n1})) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in

unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlaid string (C(Y-char_{n3})) is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Bit-String Combine

CSL

Combine Bit Strings Left

060 (1)

FORMAT:

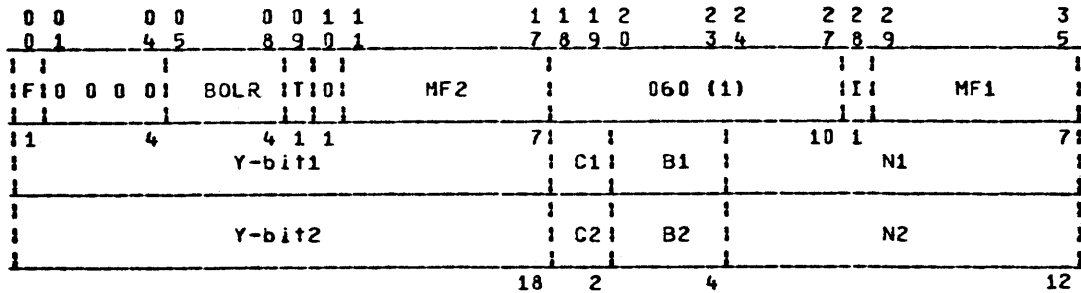


Figure 2-23 Combine Bit Strings Left (CSL) EIS Multi-Word Instruction Format

F	Fill bit for string extension
BOLR	Boolean result control field
T	Truncation Fault enable bit
MF1	Modification Field for Operand Descriptor 1
MF2	Modification Field for Operand Descriptor 2
I	Program Interrupt inhibit bit
Y-bit1	Address of "sending" string
C1	First character position of "sending" string
B1	First bit position of "sending" string
N1	Length of "sending" string
Y-bit2	Address of "receiving" string
C2	First character position of "receiving" string
B2	First bit position of "receiving" string
N2	Length of "receiving" string

ALM Coding Format:

```

cs1      (MF1),(MF2)[.enablefault][,bool(octalexpression)][,fill(01)]
descb   Y-bit1[(BITN01)],N1
descb   Y-bit2[(BITN02)],N2

```

EIS - BIT STRING COMBINE

SUMMARY: For $i = \text{bits } 1, 2, \dots, \text{minimum } (N1, N2)$
 $m = C(Y\text{-bit}1)^{i-1} \text{ !! } C(Y\text{-bit}2)^{i-1}$
 $C(\text{BOLR})_m \rightarrow C(Y\text{-bit}2)^{i-1}$
 If $N1 < N2$, then for $i = N1+1, N1+2, \dots, N2$
 $m = C(F) \text{ !! } C(Y\text{-bit}2)^{i-1}$
 $C(\text{BOLR})_m \rightarrow C(Y\text{-bit}2)^{i-1}$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y\text{-bit}2) = 00\dots 0$, then ON; otherwise OFF

Truncation If $N1 > N2$, then ON; otherwise OFF

NOTES: If $N1 > N2$, the low order $(N1-N2)$ bits of $C(Y\text{-bit}1)$ are not processed and the Truncation indicator is set ON.

The bit pattern in $C(\text{BOLR})$ defines the Boolean operation to be performed. Any of the sixteen possible Boolean operations may be used. Some common Boolean operations and their BOLR fields are shown below.

Operation	C(BOLR)
MOVE	0011
AND	0001
OR	0111
NAND	1110
Exclusive OR	0110
Clear	0000
Invert	1100

If $MF_k.RL = 1$, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MF_k.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-bit}1)$ and $C(Y\text{-bit}2)$ may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string $(C(Y\text{-bit}1))$ data is not

EIS - BIT STRING COMBINE

If $MF_k.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y-bit1)$ and $C(Y-bit2)$ may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string $C(Y-bit1)$ data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlaid string $C(Y-bit2)$ is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Bit String Compare

CMPB

Compare Bit Strings

066 (1)

FORMAT:

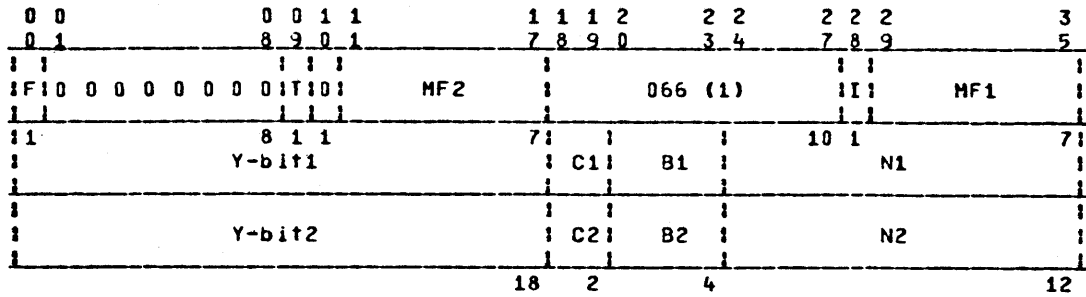


Figure 2-24 Compare Bit Strings (CMPB) EIS Multi-Word Instruction Format

- F Fill bit for string extension
- T Truncation Fault enable bit
- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- I Program Interrupt inhibit bit
- Y-bit1 Address of "left hand" string
- C1 First character position of "left hand" string
- B1 First bit position of "left hand" string
- N1 Length of "left hand" string
- Y-bit2 Address of "right hand" string
- C2 First character position of "right hand" string
- B2 First bit position of "right hand" string
- N2 Length of "right hand" string

ALM Coding Format:

```

cmpb    (MF1),(MF2)[.enablefault][,fill(01)]
descb   Y-bit1[(BITN01)],N1
descb   Y-bit2[(BITN02)],N2
    
```

SUMMARY: For i = 1, 2, ..., minimum (N1,N2)

EIS - BIT STRING COMPARE

$C(Y\text{-bit1})_{i-1} :: C(Y\text{-bit2})_{i-1}$

If $N1 < N2$, then for $i = N1+1, N1+2, \dots, N2$

$C(\text{FILL}) :: C(Y\text{-bit2})_{i-1}$

If $N2 < N1$, then for $i = N2+1, N2+2, \dots, N1$

$C(Y\text{-bit1})_{i-1} :: C(\text{FILL})$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y\text{-bit1})_i = C(Y\text{-bit2})_i$ for all i , then ON;
otherwise, OFF

Carry If $C(Y\text{-bit1})_i < C(Y\text{-bit2})_i$ for any i , then OFF;
otherwise ON

NOTES: Instruction execution proceeds until an inequality is found or the larger string length count is exhausted.

If $MFk.RL = 1$, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFk.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Bit String Set Indicators

SZTL

Set Zero and Truncation Indicators
with Bit Strings Left

064 (1)

FORMAT: Same as Combine Strings Left (CSL) (See Figure 2-23).

SUMMARY: For $i = \text{bits } 1, 2, \dots, \text{minimum } (N1, N2)$

$$m = C(Y\text{-bit}1)i-1 \text{ !! } C(Y\text{-bit}2)i-1$$

If $C(\text{BOLR})_m \neq 0$, then terminate

If $N1 < N2$, then for $i = N1+1, N1+2, \dots, N2$

$$m = C(F) \text{ !! } C(Y\text{-bit}2)i-1$$

If $C(\text{BOLR})_m \neq 0$, then terminate

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If $C(\text{BOLR})_m = 0$ for all i , then ON; otherwise OFF

Truncation If $N1 > N2$, then ON; otherwise OFF

NOTES: If $N1 > N2$, the low order $(N1-N2)$ bits of $C(Y\text{-bit}1)$ are not processed and the Truncation indicator is set ON.

The execution of this instruction is identical to Combine Strings Left (CSL) except that $C(\text{BOLR})_m$ is not placed into $C(Y\text{-bit}2)i-1$.

The bit pattern in $C(\text{BOLR})$ defines the Boolean operation to be performed. Any of the sixteen possible Boolean operations may be used. See NOTES under Combine Strings Left (CSL) instruction for examples of BOLR.

If $\text{MF}_k\text{-RL} = 1$, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $\text{MF}_k\text{-ID} = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

If $T = 1$ and the Truncation indicator is set ON by

execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - BIT STRING SET INDICATORS

SZTR Set Zero and Truncation Indicators with Bit Strings Right 065 (1)

FORMAT: Same as Combine Strings Left (CSL) (See Figure 2-23).

SUMMARY: For $i = \text{bits } 1, 2, \dots, \text{minimum } (N_1, N_2)$
 $m = C(Y\text{-bit}1)N_1 - i \text{ ; } C(Y\text{-bit}2)N_2 - i$
If $C(\text{BOLR})_m \neq 0$, then terminate
If $N_1 < N_2$, then for $i = N_1 + 1, N_1 + 2, \dots, N_2$
 $m = C(F) \text{ ; } C(Y\text{-bit}2)N_2 - i$
If $C(\text{BOLR})_m \neq 0$, then terminate

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If $C(\text{BOLR})_m = 0$ for all i , then ON; otherwise OFF

Truncation If $N_1 > N_2$, then ON; otherwise OFF

NOTES: If $N_1 > N_2$, the low order $(N_1 - N_2)$ bits of $C(Y\text{-bit}1)$ are not processed and the Truncation indicator is set ON.

The execution of this instruction is identical to Combine Strings Right (CSR) except that $C(\text{BOLR})_m$ is not placed into $C(Y\text{-bit}2)N_2 - i$.

The bit pattern in $C(\text{BOLR})$ defines the Boolean operation to be performed. Any of the sixteen possible Boolean operations may be used. See NOTES under Combine Strings Left (CSL) instruction for examples of BOLR.

If $\text{MF}_k.\text{RL} = 1$, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $\text{MF}_k.\text{ID} = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow)

Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Data Conversion

BTD Binary to Decimal Convert 301 (1)

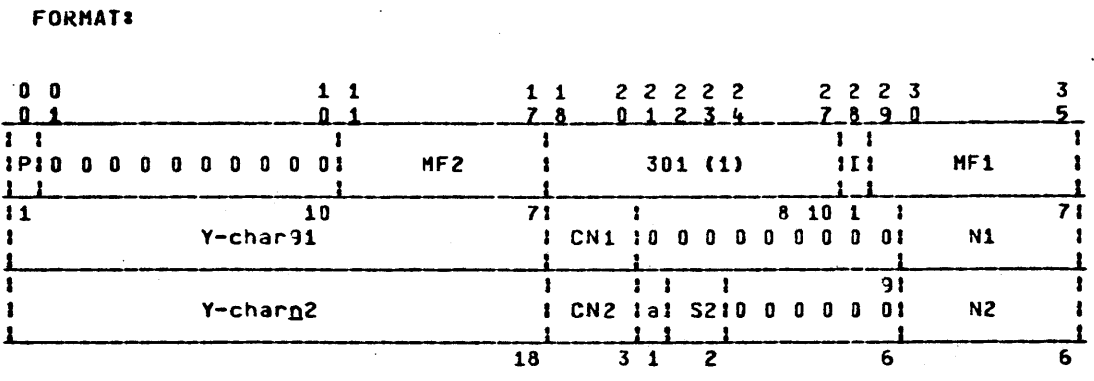


Figure 2-25 Binary to Decimal Convert (BTD) EIS Multi-Word Instruction Format

- key**
- P 4-bit data sign character control
 - MF1 Modification Field for Operand Descriptor 1
 - MF2 Modification Field for Operand Descriptor 2
 - I Program Interrupt inhibit bit
 - Y-char₉₁ Address of binary number
 - CN1 First character position of binary number
 - N1 Length of binary number in characters
 - Y-char_{n2} Address of decimal number
 - CN2 First character position of decimal number
 - a TN2 Data type of decimal number
 - S2 Sign and decimal type of decimal number
 - N2 Length of decimal number

ALM Coding Format:

btd (MF1), (MF2)
 desc9ns Y-char₉₁[(CN1)], N1
 descn[ls, ns, fs] Y-char_{n2}[(CN2)], N2 n = 4 or 9

SUMMARY: C(Y-char₉₁) converted to decimal -> C(Y-char_{n2})

EIS - DATA CONVERSION

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 ad MF2

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y\text{-char}_{\Omega 2}) = \text{decimal } 0$, then ON; otherwise OFF

Negative If a minus sign character is moved to $C(Y\text{-char}_{\Omega 2})$, then ON; otherwise OFF

Overflow If fixed point integer overflow occurs, then ON; otherwise unchanged (See NOTES)

NOTES: $C(Y\text{-char}_{91})$ contains a two's complement binary integer aligned on 9-bit character boundaries with length $0 < N1 \leq 8$.

If $TN2$ and $S2$ specify a 4-bit signed number and $P = 1$, then if $C(Y\text{-char}_{91})$ is positive (bit 0 of $C(Y\text{-char}_{91})_0 = 0$), then the 13 (octal) plus sign character is moved to $C(Y\text{-char}_{\Omega 2})$ as appropriate.

The scaling factor of $C(Y\text{-char}_{\Omega 2})$, $SF2$, must be 0.

If $N2$ is not large enough to hold the digits of $C(Y\text{-char}_{91})$ an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character and a signed fixed point field, 2 characters.

If $MFk.RL = 1$, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFk.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char}_{91})$ and $C(Y\text{-char}_{\Omega 2})$ may be overlapping strings; no check is made.

Attempted conversion to a floating point number ($S2 = 0$) or attempted use of a scaling factor ($SF2 \neq 0$) causes an Illegal Procedure Fault.

If $N1 = 0$ or $N1 > 8$ an Illegal Procedure Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

DTB

Decimal to Binary Convert

305 (1)

FORMAT:

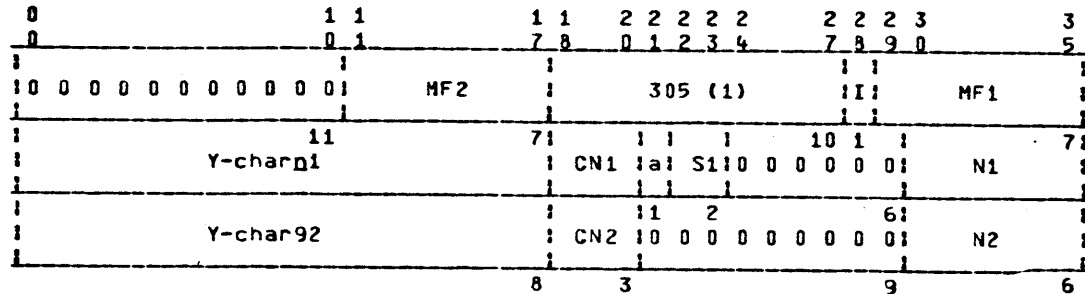


Figure 2-26 Decimal to Binary Convert (DTB) EIS Multi-Word Instruction Format

key

- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- I Program Interrupt inhibit bit
- Y-char_{d1} Address of decimal number
- CN1 First character position of decimal number
- a TN1 Data type of decimal number
- S1 Sign and decimal type of decimal number
- N1 Length of decimal number
- Y-char₉₂ Address of binary number
- CN2 First character position of binary number
- N2 Length of binary number in characters

ALM Coding Format:

dtb	(MF1),(MF2)	
descn[ls,ns,ts]	Y-char _{d1} [(CN1)],N1	d = 4 or 9
desc9ns	Y-char ₉₂ [(CN2)],N2	

SUMMARY: C(Y-char_{d1}) converted to binary -> C(Y-char₉₂)

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 ad MF2

INDICATORS: (Indicators not listed are not affected)

EIS - DATA CONVERSION

Zero If $C(Y\text{-char92}) = 0$, then ON; otherwise OFF

Negative If a minus sign character is found in $C(Y\text{-char91})$, then ON; otherwise OFF

Overflow If fixed point integer overflow occurs, then ON; otherwise unchanged (See NOTES)

NOTES:

$C(Y\text{-char92})$ will contain a two's complement binary integer aligned on 9-bit character boundaries with length $0 < N2 \leq 8$.

The scaling factor of $C(Y\text{-char91})$, SF1, must be 0.

If $N2$ is not large enough to hold the converted value of $C(Y\text{-char91})$ an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs.

If $MFk.RL = 1$, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFk.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char91})$ and $C(Y\text{-char92})$ may be overlapping strings; no check is made.

Attempted conversion of a floating point number ($S1 = 0$) or attempted use of a scaling factor ($SF1 \neq 0$) causes an Illegal Procedure Fault.

If $N2 = 0$ or $N2 > 8$ an Illegal Procedure Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - DECIMAL ADDITION

- CN2 First character position of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)
- b TN2 Data type of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)
- S2 Sign and decimal type of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)
- SF2 Scaling factor of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)
- N2 Length of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)

ALM Coding Format:

```

ad2d (MF1),(MF2)[,enable fault][,round]
descn[f1,ls,ns,ts] Y-charn1[(CN1)],N1,SF1 n = 4 or 9
descn[f1,ls,ns,ts] Y-charn2[(CN2)],N2,SF2 n = 4 or 9
    
```

SUMMARY: C(Y-char_n1) + C(Y-char_n2) -> C(Y-char_n2)

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

- Zero If C(Y-char_n2) = decimal 0, then ON; otherwise OFF
- Negative If C(Y-char_n2) is negative, then ON; otherwise OFF
- Truncation If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)
- Overflow If the overflow condition exists, then ON; otherwise unchanged (See NOTES)
- Exponent Overflow If exponent of floating point result exceeds 127 then ON; otherwise unchanged.
- Exponent If exponent of floating point result is less than -128 then ON; otherwise unchanged

NOTES: If TN2 and S2 specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If N2 is not large enough to hold the integer part of the result as scaled by SF2, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault

occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If N_2 is not large enough to hold all the digits of the result as scaled by SF2 and $R = 0$, then a truncation condition exists; data movement stops when $C(Y\text{-char}_2)$ is filled and the Truncation indicator is set ON. If $R = 1$, then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If $MF_k.RL = 1$, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MF_k.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char}_1)$ and $C(Y\text{-char}_2)$ may be overlapping strings; no check is made.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - DECIMAL ADDITION

AD3D

Add Using 3 Decimal Operands

222 (1)

FORMAT:

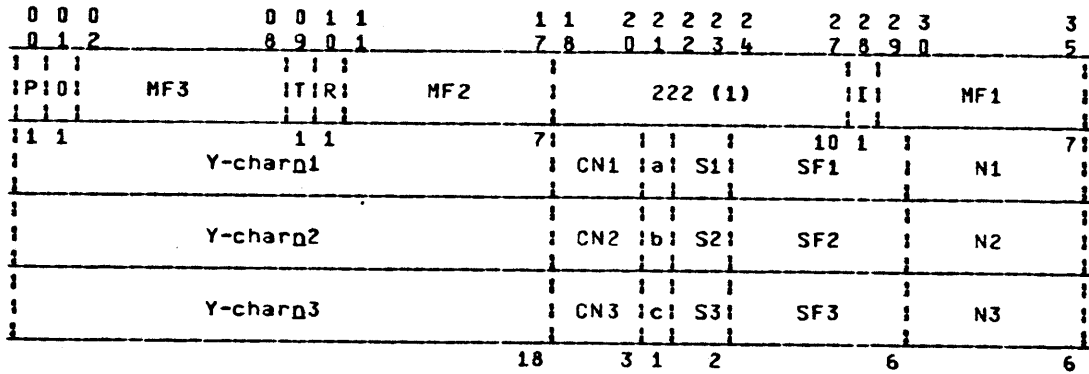


Figure 2-28 Add Using 3 Decimal Operands (AD3D) EIS Multi-Word Instruction Format

key

- P 4-bit data sign character control
- T Truncation Fault enable bit
- R Rounding flag
- MF1 Modification Field for Operand Descriptor 1
- MF2 Modification Field for Operand Descriptor 2
- MF3 Modification Field for Operand Descriptor 3
- I Program Interrupt inhibit bit
- Y-char₀1 Address of augend (AD3D), minuend (SB3D), multiplicand (MP3D), or divisor (DV3D)
- CN1 First character position of augend (AD3D), minuend (SB3D), multiplicand (MP3D), or divisor (DV3D)
- a TN1 Data type of augend (AD3D), minuend (SB3D), multiplicand (MP3D), or divisor (DV3D)
- S1 Sign and decimal type of augend (AD3D), minuend (SB3D), multiplicand (MP3D), or divisor (DV3D)
- SF1 Scaling factor of augend (AD3D), minuend (SB3D), multiplicand (MP3D), or divisor (DV3D)
- N1 Length of augend (AD3D), minuend (SB3D), multiplicand (MP3D), or divisor (DV3D)
- Y-char₂ Address of addend (AD3D), subtrahend (SB3D), multiplier (MP3D), or dividend (DV3D)

CN2	First character position of addend (AD3D), subtrahend (SB3D), multiplier (MP3D), or dividend (DV3D)
b TN2	Data type of addend (AD3D), subtrahend (SB3D), multiplier (MP3D), or dividend (DV3D)
S2	Sign and decimal type of addend (AD3D), subtrahend (SB3D), multiplier (MP3D), or dividend (DV3D)
SF2	Scaling factor of addend (AD3D), subtrahend (SB3D), multiplier (MP3D), or dividend (DV3D)
N2	Length of addend (AD3D), subtrahend (SB3D), multiplier (MP3D), or dividend (DV3D)
Y-char _{D3}	Address of sum (AD3D), difference (SB3D), product (MP3D), or quotient (DV3D)
CN3	First character position of sum (AD3D), difference (SB3D), product (MP3D), or quotient (DV3D)
a TN3	Data type of sum (AD3D), difference (SB3D), product (MP3D), or quotient (DV3D)
S3	Sign and decimal type of sum (AD3D), difference (SB3D), product (MP3D), or quotient (DV3D)
SF3	Scaling factor of sum (AD3D), difference (SB3D), product (MP3D), or quotient (DV3D)
N3	Length of sum (AD3D), difference (SB3D), product (MP3D), or quotient (DV3D)

ALM Coding Format:

```

ad3d          (MF1),(MF2),(MF3)[,enablefault][,round]
descn[f1,ls,ns,ts] Y-charD1[(CN1)],N1,SF1      D = 4 or 9
descn[f1,ls,ns,ts] Y-charD2[(CN2)],N2,SF2      D = 4 or 9
descn[f1,ls,ns,ts] Y-charD3[(CN3)],N3,SF3      D = 4 or 9
    
```

SUMMARY: C(Y-char_{D1}) + C(Y-char_{D2}) -> C(Y-char_{D3})

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If C(Y-char_{D3}) = decimal 0, then ON; otherwise OFF

Negative If C(Y-char_{D3}) is negative, then ON; otherwise OFF

Truncation If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

Overflow If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

Exponent Overflow If exponent of floating point result exceeds 127 then ON; otherwise unchanged.

EIS - DECIMAL ADDITION

Exponent If exponent of floating point result is less than -128
Underflow then ON; otherwise unchanged

NOTES:

If TN3 and S3 specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If S3 specifies fixed point and N3 is not large enough to hold the integer part of the result as scaled by SF3, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If N3 is not large enough to hold all the digits of the result as scaled by SF3 and R = 0, then a truncation condition exists; data movement stops when C(Y-char_{N3}) is filled and the Truncation indicator is set ON. If R = 1, then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If MF_k.RL = 1, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF_k.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char_{N1}), C(Y-char_{N2}), and C(Y-char_{N3}) may be overlapping strings; no check is made.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Decimal Subtraction

SB2D Subtract Using 2 Decimal Operands 203 (1)

FORMAT: Same as Add Using 2 Decimal Operands (AD2D)
(See Figure 2-27).

SUMMARY: $C(Y\text{-char}_{n1}) - C(Y\text{-char}_{n2}) \rightarrow C(Y\text{-char}_{n2})$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y\text{-char}_{n2}) = \text{decimal } 0$, then ON; otherwise OFF

Negative If $C(Y\text{-char}_{n2})$ is negative, then ON; otherwise OFF

Truncation If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

Overflow If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

Exponent Overflow If exponent of floating point result exceeds 127 then ON; otherwise unchanged.

Exponent Underflow If exponent of floating point result is less than -128 then ON; otherwise unchanged

NOTES: If IN2 and S2 specify a 4-bit signed number and $P = 1$, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If N2 is not large enough to hold the integer part of the result as scaled by SF2, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If N2 is not large enough to hold all the digits of the result as scaled by SF2 and $R = 0$, then a truncation condition exists; data movement stops when $C(Y\text{-char}_{n2})$ is filled and the Truncation indicator is set ON. If $R = 1$, then the last digit moved is rounded according to the

absolute value of the remaining digits of the result and the instruction completes normally.

If $MF_k.RL = 1$, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MF_k.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

EIS - DECIMAL SUBTRACTION

C(Y-char₁) and C(Y-char₂) may be overlapping strings; no check is made.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SB3D Subtract Using 3 Decimal Operands 223 (1)

FORMAT: Same as Add Using 3 Decimal Operands (AD3D) (See Figure 2-28).

SUMMARY: C(Y-char₁) - C(Y-char₂) -> C(Y-char₃)

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Y-char ₃) = decimal 0, then ON; otherwise OFF
Negative	If C(Y-char ₃) is negative, then ON; otherwise OFF
Truncation	If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)
Overflow	If the overflow condition exists, then ON; otherwise unchanged (See NOTES)
Exponent Overflow	If exponent of floating point result exceeds 127 then ON; otherwise unchanged.
Exponent Underflow	If exponent of floating point result is less than -128 then ON; otherwise unchanged

NOTES: If TN3 and S3 specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If S3 specifies fixed point and N3 is not large enough to hold the integer part of the result as scaled by SF3, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters;

and a floating point field, 3 characters.

If N_3 is not large enough to hold all the digits of the result as scaled by SF_3 and $R = 0$, then a truncation condition exists; data movement stops when $C(Y\text{-char}_3)$ is filled and the Truncation indicator is set ON. If $R = 1$, then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If $MF_k.RL = 1$, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MF_k.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char}_1)$, $C(Y\text{-char}_2)$, and $C(Y\text{-char}_3)$ may be overlapping strings; no check is made.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - DECIMAL MULTIPLICATION

and a floating point field, 3 characters.

If N3 is not large enough to hold all the digits of the result as scaled by SF3 and R = 0, then a truncation condition exists; data movement stops when C(Y-char_{D3}) is filled and the Truncation indicator is set ON. If R = 1, then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If MF_k.RL = 1, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF_k.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char_{D1}), C(Y-char_{D2}), and C(Y-char_{D3}) may be overlapping strings; no check is made.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EIS - Decimal Division

DV2D Divide Using 2 Decimal Operands 227 (1)

FORMAT: Same as Add Using 2 Decimal Operands (AD2D)
(See Figure 2-27).

SUMMARY: $C(Y\text{-char}_{D2}) / C(Y\text{-char}_{D1}) \rightarrow C(Y\text{-char}_{D2})$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero If $C(Y\text{-char}_{D2}) = \text{decimal } 0$, then ON; otherwise OFF

Negative If $C(Y\text{-char}_{D2})$ is negative, then ON; otherwise OFF

Truncation If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

Overflow If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

Exponent Overflow If exponent of floating point result exceeds 127 then ON; otherwise unchanged.

Exponent Underflow If exponent of floating point result is less than -128 then ON; otherwise unchanged

NOTES: This instruction performs continued long division on the operands until it has produced enough output digits to satisfy the requirements of the quotient field. The number of required quotient digits, NQ, is determined before division begins as follows ...

1) Floating point quotient

$NQ = N2$, but if the divisor is greater than the dividend after operand alignment, the leading zero digit produced is counted and the effective precision of the result is reduced by one.

2) Fixed point quotient

$$NQ = (N2 - LZ2 + 1) - (N1 - LZ1) + (E2 - E1 - SF2)$$

where: N_D = given operand field length
 LZ_D = leading zero count for operand D
 E_D = exponent of operand D
 $SF2$ = scaling factor of quotient

3) Rounding

If rounding is specified ($R = 1$), then one extra quotient digit is produced.

EIS - DECIMAL DIVISION

If $C(Y\text{-char}1) = \text{decimal } 0$ or $NQ > 63$, then division does not take place, $C(Y\text{-char}2)$ are unchanged, and a Divide Check Fault occurs.

If $TN2$ and $S2$ specify a 4-bit signed number and $P = 1$, then the $.13$ (octal) plus sign character is placed appropriately if the result of the operation is positive.

If $N2$ is not large enough to hold the integer part of the result as scaled by $SF2$, an overflow condition exists; the Overflow Indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If $N2$ is not large enough to hold all the digits of the result as scaled by $SF2$ and $R = 0$, then a truncation condition exists; data movement stops when $C(Y\text{-char}2)$ is filled and the Truncation indicator is set ON. If $R = 1$, then the last digit moved is rounded according to the absolute value of the extra quotient digit and the instruction completes normally.

If $MFK\text{-}RL = 1$, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFK\text{-}ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char}1)$ and $C(Y\text{-char}2)$ may be overlapping strings; no check is made.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range $[0,11]$ (octal) in a digit position or a character outside the range $[12,17]$ (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

DV3D

Divide Using 3 Decimal Operands

227 (1)

FORMAT:

Same as Add Using 3 Decimal Operands (AD3D)
(See Figure 2-28).

SUMMARY:

$C(Y\text{-char}2) / C(Y\text{-char}1) \rightarrow C(Y\text{-char}3)$

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Y-char _{D3}) = decimal 0, then ON; otherwise OFF
Negative	If C(Y-char _{D3}) is negative, then ON; otherwise OFF
Truncation	If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)
Overflow	If the overflow condition exists, then ON; otherwise unchanged (See NOTES)
Exponent Overflow	If exponent of floating point result exceeds 127 then ON; otherwise unchanged.
Exponent Underflow	If exponent of floating point result is less than -128 then ON; otherwise unchanged

NOTES: This instruction performs continued long division on the operands until it has produced enough output digits to satisfy the requirements of the quotient field. The number of required quotient digits, NQ, is determined before division begins as follows ...

1) Floating point quotient

NQ = N₃, but if the divisor is greater than the dividend after operand alignment, the leading zero digit produced is counted and the effective precision of the result is reduced by one.

2) Fixed point quotient

$$NQ = (N_2 - LZ_2 + 1) - (N_1 - LZ_1) + (E_2 - E_1 - SF_3)$$

where: N_D = given operand field length
LZ_D = leading zero count for operand D
E_D = exponent of operand D
SF₃ = scaling factor of quotient

3) Rounding

If rounding is specified (R = 1), then one extra quotient digit is produced.

If C(Y-char_{D1}) = decimal 0 or NQ > 63, then division does not take place, C(Y-char_{D3}) are unchanged, and a Divide Check Fault occurs.

If TN₃ and S₃ specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If S₃ specifies fixed point and N₃ is not large enough to hold the integer part of the result as scaled by SF₃, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length

EIS - DECIMAL DIVISION

of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If N_3 is not large enough to hold all the digits of the result as scaled by SF3 and $R = 0$, then a truncation condition exists; data movement stops when $C(Y\text{-char}_3)$ is filled and the Truncation indicator is set ON. If $R = 1$, then the last digit moved is rounded according to the absolute value of the extra quotient digit and the instruction completes normally.

If $MF_k.RL = 1$, then N_k does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MF_k.ID = 1$, then the k th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char}_1)$, $C(Y\text{-char}_2)$, and $C(Y\text{-char}_3)$ may be overlapping strings; no check is made.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

MICRO OPERATIONS FOR EDIT INSTRUCTIONS

The Move Alphanumeric Edited (MVE) and Move Numeric Edited (MVNE) instructions require micro operations to perform the editing functions in an efficient manner. The sequence of micro operation steps to be executed is contained in storage and is referenced by the second operand descriptor of the MVE or MVNE instructions. Some of the micro operations require special characters for insertion into the string of characters being edited. These special characters are shown in the "Edit Insertion Table" discussion below.

Micro Operation Sequence

The micro operation string operand descriptor points to a string of 9-bit characters that specify the micro operations to be performed during an edited move. Each of the 9-bit characters defines a micro operation and has the following format:

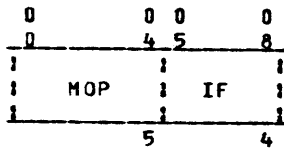


Figure 2-29 Micro Operation (MOP) Character Format

- MOP** 5 bit code specifying Micro Operation to be performed.
- IF** Information Field containing one of the following...
1. A sending string character count. A value of 0 is interpreted as 16.
 2. The index of an entry in the edit insertion table to be used. Permissible values are 1 through 8.
 3. An interpretation of the "blank-when-zero" operation

Edit Insertion Table

While executing an edit instruction, the Processor provides a register of eight 9-bit characters to hold insertion information. This register, called the "Edit Insertion Table", is not maintained after execution of an edit

instruction. At the start of each edit instruction, the Processor hardware initializes the table to the values given in Table 2-8, where each symbol refers to the corresponding standard ASCII character.

Table 2-8 Default Edit Insertion Table Characters

<u>Table Entry Number</u>	<u>Character</u>
1	blank
2	*
3	+
4	-
5	\$
6	,
7	.
8	0 (zero)

One or all of the table entries can be changed by the Load Table Entry or the Change Table micro operations to provide different insertion characters.

Edit Flags

The hardware provides the following four edit flags^a for use by the micro operations.

ES	End Suppression Flag; initially OFF and set ON by a micro operation when zero suppression ends.
SN	Sign Flag; initially set OFF if the sending string is alphanumeric or unsigned numeric. If the sending string is signed numeric, the sending string sign character is tested and SN is set OFF if positive, and ON if negative.
Z	Zero Flag; initially set ON. It is set OFF whenever a sending string character that is not decimal zero is moved into the receiving string.
BZ	Blank-When-Zero Flag; initially set OFF and is set ON by either the ENF or SES micro operation. If, at the completion of a move, both the Z and BZ are ON, the receiving string is filled with character 1 of the Edit Insertion Table.

Terminating Micro Operations

The micro operations are terminated normally when the receive string length becomes exhausted. The micro operations are terminated abnormally (with an Illegal Procedure Fault) if a move from an exhausted sending string or the use

of an exhausted MOP string is attempted.

~MVNE and MVE Differences~

The hardware executes MVNE in a slightly different manner than it executes MVE. This is due to the inherent differences in which numeric and alphanumeric data is handled. The following are brief descriptions of the hardware operations for MVNE and MVE.

~NUMERIC EDIT~

1. Load the entire sending string number (maximum length 63 characters) into the Decimal Unit Input Buffer as 4-bit digits (high-order truncating 9-bit data). Strip the sign and exponent characters (if any), put them aside into special holding registers and decrease the Input Buffer count accordingly.
2. Test sign and, if required, set the SN flag.
3. Execute micro operation string, starting with first (4-bit) digit.
4. If an Edit Insertion Table entry or MOP insertion character is to be stored, "ANDed", or "ORed" into a receiving string of 4- or 6-bit characters, high-order truncate the character accordingly.
5. If the receiving string is 9-bit characters, high-order fill the (4-bit) digits from the Input Buffer with bits 0-4 of character 8 of the Edit Insertion Table. If the receiving string is 6-bit characters, high-order fill the digits with "00"b.

~ALPHANUMERIC EDIT~

1. Load Decimal Unit Input Buffer with sending string characters. Data is read from main store in unaligned units (not 0 modulo 8 boundary) of Y-block8 words. The number of characters read is the minimum of the remaining sending string count, the remaining receiving string count, and 64.
2. Execute micro operation string, starting with the first receiving string character.
3. If an Edit Insertion Table entry or MOP insertion character is to be stored, "ANDed", or "ORed" into a receive string of 4- or 6-bit characters, high-order truncate the character accordingly.

Micro Operators

A description of the 17 micro operations (MOPs) follows. The mnemonic, name, octal value, and the function performed is given for each MOP in a format similar to that for Processor Instructions. These micro operations are included in the alphabetic list of instructions in Appendix D, identified by the code MOP.

Checks for termination are made during and after each micro operation. All MOPs that make a zero test of a sending string character test only the four least significant bits of the character.

The following additional abbreviations and symbols are used in the descriptions of the MOPs.

EIT Edit Insertion Table
pin current position in the sending string
pmop current position in the micro operation string
pout current position in the receiving string

CHT Change Table 21

SUMMARY: For $i = 1, 2, \dots, 8$
C(Y-char₉₂)pmop_i → C(EIT)_i

FLAGS: None affected

NOTES: C(IF) is not interpreted for this operation.

ENF End Floating Suppression 02

SUMMARY: If C(IF)₀ = 0, then
If ES is OFF, then
If SN is OFF, then C(EIT)₃ → C(Y-char₃)pout+1
If SN is ON, then C(EIT)₄ → C(Y-char₃)pout+1
pout = pout + 1
ES set ON
If ES is ON, then no action
If C(IF)₀ = 1, then
If ES is OFF, then
C(EIT)₅ → C(Y-char₃)pout+1
pout = pout + 1

ES set ON
If ES is ON, then no action
If C(IF)₁ = 1, then BZ set ON; otherwise no action

FLAGS: (Flags not listed are not affected)

ES	If OFF, then set ON	
BZ	If C(IF)1 = 1, then set ON; otherwise no change	
IGN	Ignore Source Character	14
SUMMARY:	C(IF) + pin -> pin	
FLAGS:	None affected	
INSA	Insert Asterisk on Suppression	11
SUMMARY:	If ES is OFF, then C(EIT)2 -> C(Y-charge3)pout+1 If C(IF) = 0, then pmop = pmop + 1 If ES is ON, then If C(IF) ≠ 0, then m = C(IF) C(EIT)m -> C(Y-charge3)pout+1 If C(IF) = 0, then C(Y-char92)pmop+1 ->C(Y-charge3)pout+1 pmop = pmop + 1	
FLAGS:	None affected	
NOTES:	If C(IF) > 8 an illegal procedure Fault occurs.	
INSB	Insert Blank on Suppression	10
SUMMARY:	If ES is OFF, then C(EIT)1 -> C(Y-charge3)pout+1 If C(IF) = 0, then pmop = pmop + 1 If ES is ON, then If C(IF) ≠ 0, then m = C(IF) C(EIT)m -> C(Y-charge3)pout+1	

If C(IF) = 0, then
C(Y-char92)pmop+1 ->C(Y-char93)pout+1
pmop = pmop + 1

FLAGS: None affected

NOTES: If C(IF) > 8 an illegal Procedure Fault occurs.

INSM Insert Table Entry 1 Multiple 01

SUMMARY: For i = 1, 2, ..., C(IF)
C(EIT)i -> C(Y-char93)pout+1

FLAGS: None affected

INSN Insert on Negative 12

SUMMARY: If SN is OFF, then
C(EIT)1 -> C(Y-char93)pout+1
If C(IF) = 0, then pmop = pmop + 1
If SN is ON, then
If C(IF) ≠ 0, then
m = C(IF)
C(EIT)m -> C(Y-char93)pout+1
If C(IF) = 0, then
C(Y-char92)pmop+1 ->C(Y-char93)pout+1
pmop = pmop + 1

FLAGS: None affected

NOTES: If C(IF) > 8 an illegal procedure Fault occurs.

INSP

Insert on Positive

13

SUMMARY:

If SN is ON, then

C(EIT)1 -> C(Y-char₃)pout+1

If C(IF) = 0, then pmop = pmop + 1

If SN is OFF, then

If C(IF) ≠ 0, then

m = C(IF)

C(EIT)m -> C(Y-char₃)pout+1

If C(IF) = 0, then

C(Y-char₉₂)pmop+1 ->C(Y-char₃)pout+1

pmop = pmop + 1

FLAGS:

None affected

NOTES:

If C(IF) > 8 an Illegal Procedure Fault occurs.

LTE

Load Table Entry

20

SUMMARY:

m = C(IF)

C(Y-char₉₂)pmop+1 -> C(EIT)m

pmop = pmop + 1

FLAGS:

None affected

NOTES:

If C(IF) = 0 or C(IF) > 8 an Illegal Procedure Fault occurs.

MFLC

Move with Float Currency Symbol Insertion

07

SUMMARY:

For i = 1, 2, ..., C(IF)

If ES is ON, then C(Y-char₁)pin+i -> C(Y-char₃)pout+i

If ES is OFF and C(Y-char₁)pin+i = decimal 0, then

C(EIT)1 -> C(Y-char₃)pout+i

If ES is OFF and C(Y-char₁)pin+i ≠ decimal 0, then

C(EIT)5 -> C(Y-char₃)pout+i

C(Y-char_{n1})pin+i → C(Y-char_{n3})pout+i+1

pout = pout + 1

ES set ON

FLAGS: (Flags not listed are not affected)

ES If OFF and any of C(Y-char_{n1})pin+i ≠ decimal 0, then ON; otherwise unchanged

NOTES: If N1 or N2 exhausts before N3, an Illegal Procedure Fault occurs.

The number of characters moved to the receiving string is data dependent. If the entire C(Y-char_{n1}) is decimal 0's, C(IF) characters are moved to C(Y-char_{n3}). However, if the receiving string contains a non-zero character, then C(IF)+1 characters are moved to C(Y-char_{n3}); the insertion character plus C(Y-char_{n1}). The user is advised that a possible Illegal Procedure Fault due to this condition may be avoided by assuring that the Z and BZ flags are ON.

MFLS Move with Float Sign Insertion

06

*SUMMARY: For i = 1, 2, ..., C(IF)

If ES is ON, then C(Y-char_{n1})pin+i → C(Y-char_{n3})pout+i

If ES is OFF and C(Y-char_{n1})pin+i = decimal 0, then

C(EIT)1 → C(Y-char_{n3})pout+i

If ES is OFF and C(Y-char_{n1})pin+i ≠ decimal 0, then

If SN is OFF, then C(EIT)3 → C(Y-char_{n3})pout+i

If SN is ON, then C(EIT)4 → C(Y-char_{n3})pout+i

C(Y-char_{n1})pin+i → C(Y-char_{n3})pout+i+1

pout = pout + 1

ES set On

FLAGS: (Flags not listed are not affected)

ES If OFF and any of C(Y-char_{n1})pin+i ≠ decimal 0, then ON; otherwise unchanged

NOTES: If N1 or N2 exhausts before N3, an Illegal Procedure Fault occurs.

The number of characters moved to the receiving string is data dependent. If the entire C(Y-char_{n1}) is decimal 0's,

C(IF) characters are moved to C(Y-char₃). However, if the receiving string contains a non-zero character, then C(IF)+1 characters are moved to C(Y-char₃); the insertion character plus C(Y-char₁). The user is advised that a possible Illegal Procedure Fault due to this condition may be avoided by assuring that the Z and 3Z flags are ON.

MORS Move and OR Sign 17

SUMMARY: For i = 1, 2, ..., C(IF)
 If SN is OFF, then
 C(Y-char₁)_{pin+i} | C(EIT)3 -> C(Y-char₃)_{pout+i}
 If SN is ON, then
 C(Y-char₁)_{pin+i} | C(EIT)4 -> C(Y-char₃)_{pout+i}

FLAGS: None affected

MSES Move and Set Sign 16

SUMMARY: EOC MVNE
 For i = 1, 2, ..., C(IF)
 C(Y-char₁)_{pin+i} -> C(Y-char₃)_{pout+i}
 EOC MVE
 For i = 1, 2, ..., C(IF)
 C(Y-char₁)_{pin+i} -> C(Y-char₃)_{pout+i}
 C(Z) = C(Y-char₁)_{pin+i} & C(EIT)3
 If C(Z) ≠ 0, then for j = i+1, i+2, ..., C(IF)
 C(Y-char₁)_{pin+j} -> C(Y-char₃)_{pout+j}
 If C(Z) = 0, then
 C(Z) = C(Y-char₁)_{pin+i} & C(EIT)4
 If C(Z) ≠ 0, then

SN set ON

For j = i+1, i+2, ..., C(IF)
 C(Y-char₁)_{pin+j} -> C(Y-char₃)_{pout+j}

FLAGS: (Flags not listed are not affected)

FLAGS: (Flags not listed are not affected)
ES If OFF and any of C(Y-char)pin+i ≠ decimal 0, then ON; otherwise unchanged
NOTES: If N1 or N2 exhausts before N3, an Illegal Procedure Fault occurs.

SES Set End Suppression 03

SUMMARY:
 If C(IF)0 = 0, then ES set OFF
 If C(IF)0 = 1, then ES set ON
 If C(IF)1 = 1, then BZ set ON; otherwise no action

FLAGS: (Flags not listed are not affected)
ES Set by this micro operation
BZ If C(IF)1 = 1, then ON; otherwise no change

Micro Operation Code Assignment Map

Operation code assignments for the micro operations are shown in Table 2-9 below. (---) indicates an unassigned code. All unassigned codes cause an Illegal Procedure Fault.

Table 2-9 Micro Operation Code Assignment Map

	0	1	2	3	4	5	6	7
00	---	insm	enf	ses	mvzb	mvza	mfis	mfic
10	insb	insal	insn	insp	ign	mvc	mses	mors
20	lte	cht	---	---	---	---	---	---
30	---	---	---	---	---	---	---	---

SECTION III

DATA REPRESENTATION

INFORMATION ORGANIZATION

The Processor, like the rest of the Multics system, is organized to deal with information in basic units of 36-bit "words". Other units of 4-, 6-, 9-bit "characters" or "bytes", 18-bit "half words", and 72-bit "word pairs" can be manipulated within the Processor by use of the instruction set. These bit groupings are used by the hardware and software to represent a variety of forms of coded data. Certain Processor functions appear to manipulate larger units of 144, 288, 576, and 1152 bits, but functions are performed by means of repeated use of 72-bit word pairs. All information is represented as strings of binary bits.

POSITION NUMBERING

The numbering of bit positions, character positions, and words increases in the direction of conventional reading and writing: from the most-significant to the least-significant digit of a number, and from left to right in conventional alphanumeric text.

Graphic presentations in this manual show registers and data with position numbers increasing from left to right.

NUMBER SYSTEM

The arithmetic functions of the Processor are implemented in the two's complement, binary number system. One of the primary properties of this number system is that a field (or register) having width n bits may be interpreted in two different ways; the "logical" case and the "arithmetic" or "algebraic" case.

In the logical case, the number is unsigned, positive, and lies in the range $(0, 2^{**n} - 1)$. The results of arithmetic operations on numbers for this case are interpreted as 0 modulo n numbers. Overflow is not defined for this case since the range of the field or register cannot be exceeded. The numbers "0" and " $2^{**n} - 1$ " are consecutive (not separated) in the set of numbers defined for the field or register.

In the arithmetic case, the number is signed and lies in the range $(-2^{n-1}, 2^{n-1} - 1)$. Overflow is defined for this case since the range can be exceeded in either direction (positive or negative). The left-hand-most bit of the field or register (bit 0) serves as the sign bit and does not contribute to the value of the number.

The main advantage of this implementation is that the hardware arithmetic algorithms for the two cases are identical; the only distinction lying in the interpretation of the results by the user. Instruction set features are provided for performing binary arithmetic with overflow disabled (the so-called logical instructions) and for comparing numbers in either sense.

Subtraction is performed by adding the two's complement of the subtrahend to the minuend. (Note that when the subtrahend is zero the algorithm for forming the two's complement is still carried out, but, since the two's complement of zero is zero, the result is correct.)

Another important feature of the two's complement number system (with respect to comparison of numeric values) is that the "no borrow" condition in true subtraction is identical to the "carry" condition in true addition and vice versa.

A statement on the assumed location of the binary point has significance only for multiplication and division. These two operations are implemented for the arithmetic case in both integer and fraction modes. "Integer" means that the position of the binary point is assumed to the right of the least-significant bit position (that is, to the right of the right-hand-most bit of the field or register) and "fraction" means that the position of the binary point is assumed to the left of the most-significant bit position (that is, between bit 0 and bit 1 of the field or register; recall that bit 0 is the sign bit).

INFORMATION FORMATS

The Figures below show the unstructured formats (templates) for the various information units defined for the Processor. Data transfer between the Processor and main store is word oriented; a 36-bit machine word is transferred for single-precision operands and sub-fields of machine words, and a 72-bit word pair is transferred for all other cases (multi-word operands, instruction fetches, bit- and character-strings). The information unit to be used and the data transfer mode is determined by the Processor according to the function to be performed.

The 36-bit unstructured machine word shown in Figure 3-1 below is the minimum addressable information unit in main store. Its location is uniquely determined by its main store address, Y. All other information units are defined relative to the 36-bit machine word.

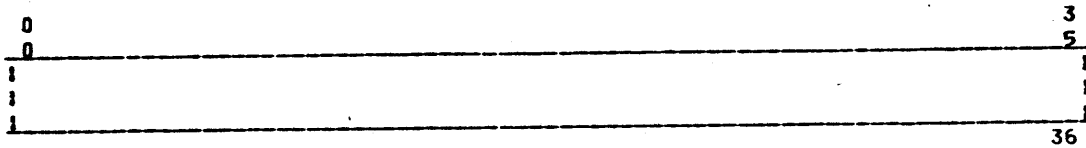


Figure 3-1 Unstructured Machine Word Format

Two consecutive machine words as shown in Figure 3-2 below, the first having an even main store address, form a 72-bit word pair. In 72-bit word pair data transfer mode, the word pair is uniquely located by the main store address of either of its constituent 36-bit machine words. Thus, if Y is even, the word pair at (Y,Y+1) is selected. If Y is odd, the word pair at (Y-1,Y) is selected. The term "Y-pair" is used for such a word pair address.

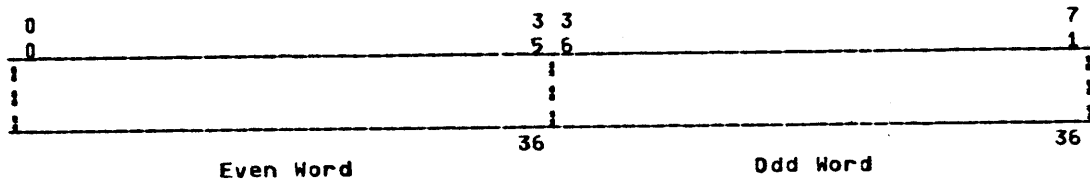


Figure 3-2 Unstructured Word Pair Format

4-bit characters are mapped onto 36-bit machine words as shown in Figure 3-3 below. The "0" bits at bit positions 0, 9, 18, and 27 are forced to be 0 by the Processor on data transfers to main store and are ignored on data transfers from main store.

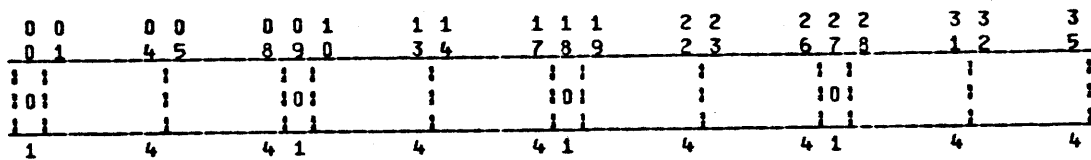


Figure 3-3 Unstructured 4-bit Character Format

6-bit characters are mapped onto 36-bit machine words as shown in Figure

3-4 below.

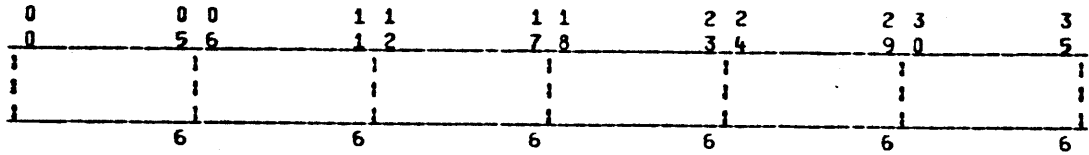


Figure 3-4 Unstructured 6-bit Character Format

9-bit characters are mapped onto 36-bit machine words as shown in Figure 3-5 below.

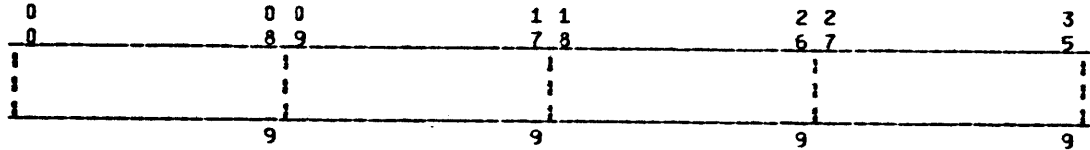


Figure 3-5 Unstructured 9-bit Character Format

18-bit half words are mapped onto 38-bit machine words as shown in Figure 3-6 below.

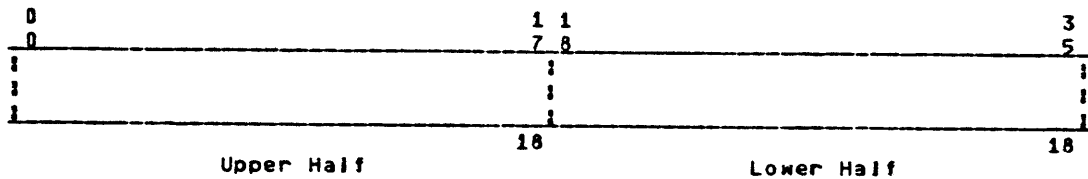


Figure 3-6 Unstructured 18-bit Half Word Format

DATA PARITY

Odd parity on each 36-bit machine word transferred to main store is generated as it leaves the Processor, is verified at several points along the transmission path, and is held in main store as an "extra" bit. If an incorrect parity is detected at any of the various parity "check points", the main store

returns an Illegal Action signal and a code appropriate to the check point.

On data transfers from main store, the parity bit is retrieved and transmitted with the data bits. The same verification checks are made and Illegal Action signalled for errors. The Processor makes a final parity check as the data enters the Processor.

Any detected parity error causes the Processor Parity indicator to set ON and (if enabled) a Parity Fault.

REPRESENTATION OF DATA

Data is defined by imposing an operand structure on the information units described above. Data is represented in two forms: numeric or alphanumeric. The form is determined by the Processor according to function to be performed.

Numeric Data

Numeric data is represented in three modes: fixed point binary, floating point binary, and decimal. The mode is determined by the Processor according to the function being performed and any Address Modification invoked for the instruction being executed.

FIXED POINT BINARY DATA

Fixed Point Binary Integers

Fixed point binary integer data is defined by imposing either of the bit position value structures shown below on an information unit of n bits.

Logical values:

$$a(0)x2^{*n} + a(1)x2^{*(n-1)} + \dots + a(n-1) \quad \uparrow$$

Arithmetic values:

$$["-" \{ a(0) \} (a(1) \ominus a(0)) x2^{*(n-1)} + (a(2) \ominus a(0)) x2^{*(n-2)} + \dots + (a(n-1) \ominus a(0)) \quad \uparrow$$

where:

$a(i)$ is the value of the bit in the i th bit position

\ominus indicates the Boolean Exclusive OR function

" \uparrow " indicates the position of the binary point

["-" { $a(0)$ } selects the proper sign according the value of $a(0)$

The following fixed point binary integer data items are defined:

n	Name
6	6-bit Byte Operand
9	9-bit Byte Operand
18	Half Word Operand
36	Single Precision Operand
72	Double Precision Operand

Note that 4-bit Byte Operands are not defined. This data item is defined only for Decimal Data. (See Decimal Data below.)

The proper operand and its position within a 36-bit machine word is determined by the Processor during preparation of the main store address for the operand. If the data width of the operand selected is smaller than the register involved, the operand is high-order and/or low-order zero filled as necessary.

Table 3-1 Fixed Point Binary Integer Values

Operand	6-bit Byte	9-bit Byte	18-bit Half Word	36-bit Single Precision	72-bit Double Precision
Logical range					
Minimums	0	0	0	0	0
Maximums	$(2^{**6})-1$	$(2^{**9})-1$	$(2^{**18})-1$	$(2^{**36})-1$	$(2^{**72})-1$
Resolutions	1	1	1	1	1
Arithmetic range					
Minimums	0	0	0	0	0
Maximums					
Neg.	$-(2^{**5})$	$-(2^{**8})$	$-(2^{**17})$	$-(2^{**35})$	$-(2^{**71})$
Pos.	$(2^{**5})-1$	$(2^{**8})-1$	$(2^{**17})-1$	$(2^{**35})-1$	$(2^{**71})-1$
Resolutions	1	1	1	1	1

Fixed Point Binary Fractions

Fixed point binary fraction data is defined by imposing the bit position value structure below on an information unit of n bits.

Arithmetic values:

$$["-" \text{ or } \text{a}(0)] \text{ a}(1) \text{ or } \text{a}(0) \times 2^{** -1} + \text{a}(2) \text{ or } \text{a}(0) \times 2^{** -2} + \dots + \text{a}(n-1) \text{ or } \text{a}(0) \times 2^{** -(n-1)}$$

Note that logical values are not defined for fixed point binary fraction

data.

The following fixed point binary fraction data items are defined:

n	Name
6	6-bit Byte Operand
9	9-bit Byte Operand
18	Half Word Operand
36	Single Precision Operand

Note that 4-bit Byte Operands and 72-bit Double Precision Operands are not defined. 4-bit Byte Operands are defined only for Decimal Data. (See Decimal Data below.) If the instruction being executed is Divide Fraction (DVF), the contents of the combined Accumulator and Quotient Registers are treated as a 72-bit fixed point binary fraction value but are not addressable as an operand.

The proper operand and its position within a 36-bit machine word is determined by the Processor during preparation of the main store address for the operand. If the data width of the operand selected is smaller than the register involved, the operand is high-order or low-order zero filled as necessary.

Table 3-2 Fixed Point Binary Fraction Values

Operand	6-bit Byte	9-bit Byte	Lower 18-bit Half Word
Arithmetic range			
Minimum:	0	0	0
Maximum:	---	---	---
Neg.	---(1)	---	---
Pos.	$((2^{**5})-1) \times 2^{**-35}$	$((2^{**8})-1) \times 2^{**-35}$	$((2^{**17})-1) \times 2^{**-35}$
Resolution:	2^{**-35}	2^{**-35}	2^{**-35}

Operand	Upper 18-bit Half Word	36-bit Single Precision
Arithmetic range		
Minimum:	0	0
Maximum:		
Neg.	-1.0	-1.0
Pos.	$1.0 - 2^{**-17}$	$1.0 - 2^{**-35}$
Resolution:	2^{**-17}	2^{**-35}

(1) No Negative maximum is shown for 6-bit Byte, 9-bit Byte, and Lower 18-bit Half Word operands since the high-order zero fill during operand alignment forces the sign bit to zero.

All operands are legal for the Divide Fraction (DVF) instruction but only the 18-bit Half Word and 36-bit Single Precision operands are legal for the

Multiply Fraction (MPF) instruction.

Fixed point binary fraction operands are illegal for all other instructions.

FLOATING POINT BINARY DATA

A floating point binary number is expressed as

$$Z = M \times 2^{**} E$$

where:

M is an arithmetic fixed point binary fraction; the mantissa

E is an arithmetic fixed point integer; the exponent

A floating point binary number is defined by imposing the bit position value structure below on an information unit of n bits.

Exponent values:

$$[\text{---} \&a(0)] (a(1)\&a(0))x2^{**6} + (a(2)\&a(0))x2^{**5} + \dots + (a(7)\&a(0)) \uparrow$$

Mantissa value:

$$[\text{---} \&a(8)] (a(9)\&a(8))x2^{**(-1)} + (a(10)\&a(8))x2^{**(-2)} + \dots + (a(n-1)\&a(8))x2^{**(7-n)} \uparrow$$

where the symbols and notation are the same as for fixed point binary data above.

The following floating point data items are defined.

D	Name
18	Half Word Operand
36	Single Precision Operand
72	Double Precision Operand

For clarity, the formats of these operands are shown in Figures 3-7 through 3-10 below.

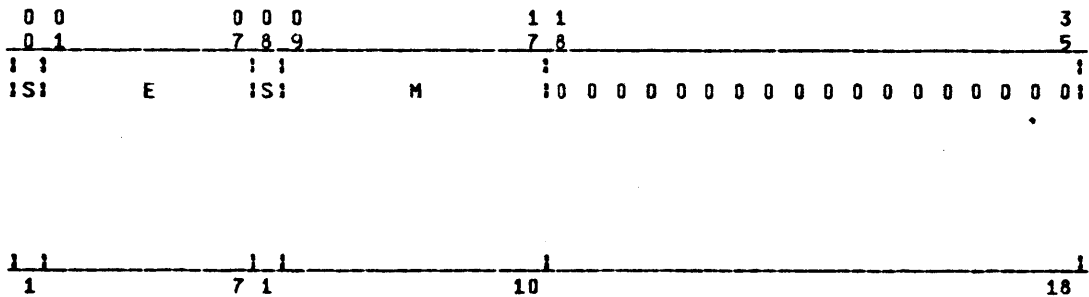


Figure 3-7 Upper 18-bit Half Word Floating Point Binary Operand Format

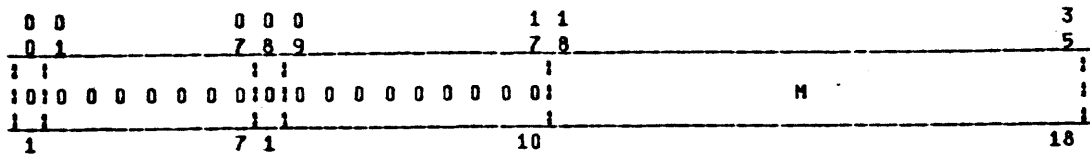


Figure 3-8 Lower 18-bit Half Word Floating Point Binary Operand Format

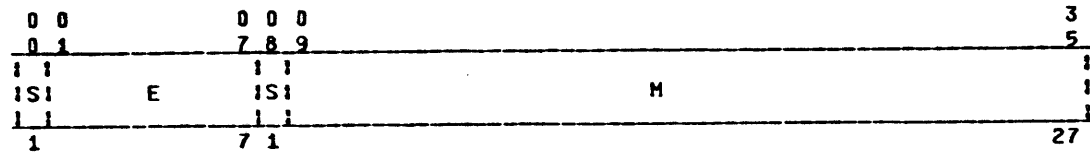


Figure 3-9 Single Precision Floating Point Binary Operand Format

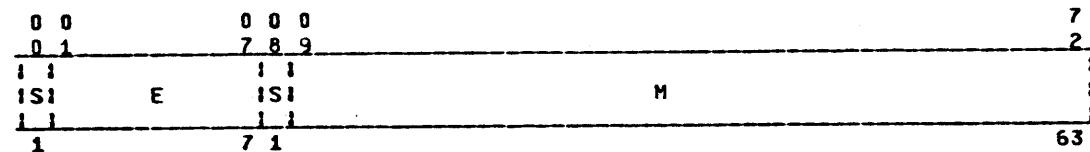


Figure 3-10 Double Precision Floating Point Binary Operand Format

The proper operand is selected by the Processor during preparation of the main store address for the operand. If the data width of the operand is smaller than the register involved, the operand is high-order or low-order zero filled as necessary.

Overlength Registers

The combined AQ register is used to hold the mantissa of all floating point binary numbers. The AQ register is said to be overlength with respect to the operands since it has more bits than are provided by the operands. Operands are low-order zero filled when loaded and low-order truncated (or rounded, depending on the instruction) when stored. Thus, the result of all floating point instructions has more bits of precision in the AQ than may be stored.

Users are cautioned that algorithms involving floating point operands may

suffer from propagation of truncation errors unless the algorithms are designed to hold mantissas in the AQ register as long as possible. It is possible to retain full AQ precision of results if they are saved with the Store AQ (STAQ) and Store Exponent (STE) instructions but such saved data are not usable as a floating point operand.

Normalized Numbers

A floating point number is said to be normalized if the relation

$$(0.5 \leq |M| < 1.0)$$

is satisfied. The presence of unnormalized numbers in any finite mantissa arithmetic can only degrade the accuracy of results. For example, in an arithmetic allowing only two digits in the mantissa, the number $0.005 \times 10^{+2}$ has the value zero instead the value one half.

Normalization is a process of shifting the mantissa and adjusting the exponent until the relation above is satisfied. Normalization may be used to recover some or all of the extra bits of the overlength AQ register after a floating point operation.

There are cases where the limits of the registers force the use of unnormalized numbers. For example, in an arithmetic allowing three digits of mantissa and one digit of exponent, the calculation $0.3 \times 10^{+10} - 0.1 \times 10^{+11}$ (the normalized case) may not be made, but $0.03 \times 10^{+9} - 0.001 \times 10^{+9} = 0.029 \times 10^{+9}$ (the unnormalized case) is a valid result.

Some examples of normalized and unnormalized numbers are:

Unnormalized positive binary	$0.00011010 \times (2^{+7})$
Same number normalized	$0.11010000 \times (2^{+4})$
Unnormalized negative binary	$1.11010111 \times (2^{+4})$
Same number normalized	$1.01011100 \times (2^{+6})$

The minimum normalized non-zero floating point binary number is 2^{+128} in all cases.

Table 3-3 Floating Point Binary Operand Values

Operand	Lower 18-bit Half Word	Upper 18-bit Half Word	36-bit Single Precision
Unnormalized range			
Minimum:	0(1)	0	0
Maximum:			
Neg.	---(2)	$-1.0 \times 2^{+127}$	$-1.0 \times 2^{+127}$
Pos.	$(2^{+9} - 1) \times 2^{+155}$	$(1 - 2^{+9}) \times 2^{+127}$	$(1 - 2^{+27}) \times 2^{+127}$
Resolution:	2^{+155}	1:9(3)	1:27
Operand	72-bit Double Precision		
Unnormalized range			
Minimum:	0		
Maximum:			
Neg.	$-1.0 \times 2^{+127}$		
Pos.	$(1 - 2^{+63}) \times 2^{+127}$		
Resolution:	1:63		

- (1) There is no unique representation for the value zero in floating point binary numbers; any number with mantissa zero has the value zero. However, the Processor treats a zero mantissa as a special case in order to preserve precision in later calculations with a zero intermediate result. Whenever the Processor detects a zero mantissa as the result of a floating binary operation, the AQ register is cleared to zeros and the E register is set to -128. This representation is known as a floating normalized zero. The unnormalized zero (any zero mantissa) will be handled correctly if encountered in an operand but precision may be lost. For example, $A \times 10^{*-14} + 0 \times 10^{*85}$ will not produce desired results since all the precision of A will be lost when it is aligned to match the 10^{*85} exponent of the 0.
- (2) No Negative maximum is shown for Lower 18-bit Half Word operands since the high-order zero fill during operand alignment forces the sign bit to zero.
- (3) A value cannot be given for Resolution in these cases since such a value depends on the value of the exponent, E. The notation used (1:m) indicates resolution to 1 bit in a field of m. Thus, the following general statement on resolution may be made:

The resolution of a floating point binary operand with mantissa length m and exponent value E is $2^{*(E-m)}$.

DECIMAL DATA

Decimal numbers are expressed in one of the following forms:

Fixed point, no sign	MMMMMM.
Fixed point, leading sign	±MMMMMM.
Fixed point, trailing sign	MMMMMM.±
Floating point	±MMMMMM. x 10^{*E}

The form is specified by control information in the Operand Descriptor for the operand as used by the Extended Instruction Set (EIS). (See Section II, Machine Instructions.)

A decimal number is defined by imposing any of the character position value structures below on a 4-bit Character or 9-bit Character information unit of length n characters.

Fixed point, no sign:

$$c(0) \times 10^{*(n-1)} + c(1) \times 10^{*(n-2)} + \dots + c(n-1)$$

†

Fixed point, leading sign:

$$[\text{sign}=c(0)] c(1) \times 10^{*(n-2)} + c(2) \times 10^{*(n-3)} + \dots + c(n-1)$$

†

Fixed point, trailing sign:

$$c(0) \times 10^{*(n-2)} + c(1) \times 10^{*(n-3)} + \dots + c(n-1) \text{ [sign=c(n)]}$$

Floating point:

$$\text{[sign=c(0)] } c(1) \times 10^{*(n-3)} + c(2) \times 10^{*(n-4)} + \dots + c(n-2) \text{ [exponent=8 bits]}$$

where:

c(i) is the decimal value of the character in the i-th character position.

"f" indicates the position of the decimal point.

[sign=c(i)] indicates that c(i) is interpreted as a sign character.

[exponent=8 bits] indicates that the exponent value is taken from the last 8 bits of the character string. If the data is in 9-bit Characters, the exponent is bits 1-8 of c(n). If the data is in 4-bit Characters, the exponent is the concatenated value of c(n-1) and c(n).

The decimal number as described above is the only decimal data item defined. It may begin on any legal character boundary (without regard to word boundaries) and has a maximum extent of 63 characters.

The Processor handles decimal data as 4-bit bytes internally. Thus, 9-bit characters are high-order truncated as they are transferred from main store and high-order filled as they are transferred to main store. The fill pattern is "00110"b for digit characters and "00100"b for sign characters. The floating point exponent is a special case and is treated as a two's complement binary integer.

The Processor performs validity checking on decimal data. Only the byte values (0,11) octal are legal in digit positions and only the byte values (12,17) octal are legal in sign positions. Detection of an illegal byte value causes an Illegal Procedure Fault. The interpretation of decimal sign characters is shown in Table 3-4 below.

Table 3-4 Decimal Sign Character Interpretation

9-bit Character	4-bit Character	Interpretation
--------------------	--------------------	----------------

52	12	+
53(1)	13(2)	+
54	14(1)	+
55(1)	15(1)	-
56	16	+
57	17	+

- (1) This character is used as the default sign character for storage of results. The presence of other characters will yield correct results according to the interpretation.
- (2) An optional control bit in the EIS Decimal Arithmetic instructions (See Section II, Machine Instructions) allows the selection of (13) octal for the plus sign character for storage of results in 4-bit data mode.

Decimal Data Values

The Operand Descriptors for decimal data operands have a 6-bit two's complement binary field for invocation of a Scaling Factor (SF). This Scaling Factor has the same effect as the value of E in floating point decimal operands; a negative value moves the assumed decimal point to the left; a positive value, to the right. The use of the Scaling Factor extended the range and resolution of decimal data operands. The range of the Scaling Factor is (-32,31).

Table 3-5 Decimal Data Values

Operand	Fixed Point No Sign	Fixed Point Leading or Trailing Sign
Arithmetic range		
Minimum:	0(1)	0
Maximum:	$(10^{**}64 - 1) \times 10^{**}31$	$\pm(10^{**}63 - 1) \times 10^{**}31$
Resolution:	1:SF(2)	1:SF
Operand	9-bit Floating Point	4-bit Floating Point
Arithmetic range		
Minimum:	0	0
Maximum:	$\pm(10^{**}62 - 1) \times 10^{**}158$	$\pm(10^{**}61 - 1) \times 10^{**}158$
Resolution:	1:SF+E	1:SF+E

(1) See Decimal Zero below.

(2) A value cannot be given for Resolution in these cases since such a value depends on the value of the Scaling Factor, SF, and/or the exponent, E. The notation used (1:SF+E) indicates resolution to 1 part in $10^{**}(SF+E)$. Thus, the following general statement on resolution may be made:

The resolution of a fixed point decimal operand with Scaling Factor SF is $10^{**}SF$ and the resolution of a floating point decimal operand with Scaling Factor SF and exponent E is $10^{**}(SF+E)$.

Decimal Zero

As in floating point binary arithmetic, there is no unique representation of the value zero except in the case of fixed point, no sign data. Therefore, the Processor detects a zero result and forces a value of +0. for fixed point, leading or trailing sign and +0. x 10**127 for floating point data. Again, as in floating binary arithmetic, other representations of the value zero will be handled correctly except for possible loss of precision during operand alignment.

Alphanumeric Data

Alphanumeric data is represented in two modes; character string and bit string. The mode is determined by the Processor according to the function being performed.

CHARACTER STRING DATA

Character string data is defined by imposing the character position structure below on a 4-bit, 6-bit, or 9-bit Character information unit of length n characters.

$$c(0) \text{ || } c(1) \text{ || } \dots \text{ || } c(n-1)$$

where:

$c(i)$ is the character in the i th character position.

|| indicates the concatenation operation.

The character string described above is the only character string data item defined. It may begin on any legal character boundary (without regard to word boundaries) and has a maximum extent as shown in Table 3-6 below.

Table 3-6 Character String Data Length Limits

<u>Character Size</u>	<u>Length Limit</u>
9-bit	1048576
6-bit	1572864
4-bit	2097152

No interpretation of the characters is made except as specified for the instruction being executed. (See Section II, Machine Instructions.)

BIT STRING DATA

Bit string data is defined by imposing the bit position structure below on a machine word information unit of length n bits.

$$b(0) || b(1) || \dots || b(n-1)$$

where:

$b(i)$ is the value of the bit in the i th position.

$||$ indicates the concatenation operation.

The bit string described above is the only bit string data item defined. It may begin at any bit position (without regard to character or word boundaries) and has a maximum extent of 9437184000 bits.

SECTION IV

PROGRAM ACCESSIBLE REGISTERS

A Processor register is a hardware assembly that holds information for use in some specified way. An accessible register is a register whose contents are available to the user for his purposes. Some accessible registers are explicitly referenced by particular instructions, some are implicitly referenced during the course of execution of instructions, and some are used in both ways. The accessible registers are listed in the table below. See Section II, Machine Instructions, for a discussion of each instruction to determine the way in which the registers are used.

Table 4-1 Processor Registers

Name	Mnemonic	Bit Length	Quantity
Accumulator Register	A	36	1
Quotient Register	Q	36	1
Accumulator-Quotient Register(1)	AQ	72	1
Exponent Register	E	8	1
Exponent-Accumulator-Quotient Register(1)	EAQ	80	1
Index Registers	Xn	18	8
Indicator Register	IR	14	1
Base Address Register	BAR	18	1
Timer Register	TR	27	1
Ring Alarm Register	RALR	3	1
Pointer Registers	PRn	42	8
Procedure Pointer Register	PPR	37	1
Temporary Pointer Register	TPR	42	1
Descriptor Segment Base Register	DSBR, (DBR)	51	1
Segment Descriptor Word Associative Memory	SDWAM	85	16
Page Table Word Associative Memory	PTWAM	51	16
Fault Register		35	1
Mode Register	MR	33	1
Cache Mode Register	CMR	28	1
Control Unit (CU) History Register		72	16
Operations Unit (OU) History Register		72	16
Decimal Unit (DU) History Register		72	16
Appending Unit (AU) History Register		72	16
Configuration Switch Data		36	5
Control Unit Data		576	1
Decimal unit data		288	1

(1) These registers are not separate physical assemblies but are logical combinations of their constituent registers.

In the descriptions that follow, the diagrams given for register formats do not imply that a physical assembly possessing the pictured bit pattern exists.

The diagram is a graphic representation of the form of the register data as it appears in main store when the register contents are stored or how data bits must be assembled for loading into the register.

ACCUMULATOR REGISTER (A)

Format: - 36 bits

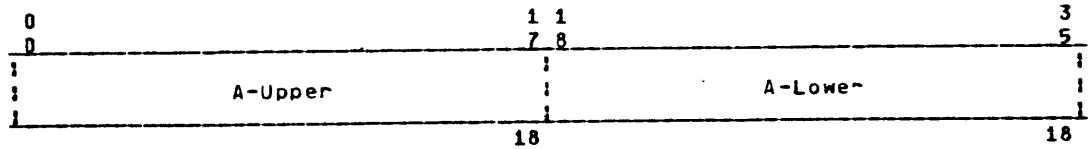


Figure 4-1 Accumulator Register (A) Format

Description:

A 36 bit physical register located in the Operations Unit.

Functions:

In fixed point binary operations, holds operands and results.

In floating point binary operations, holds the most significant part of the mantissa.

In shifting operations, holds original data and shifted results.

In address preparation, may hold two logically independent word offsets, A-Upper and A-Lower, or an extended range bit or character offset.

QUOTIENT REGISTER (Q)

Format: - 36 bits

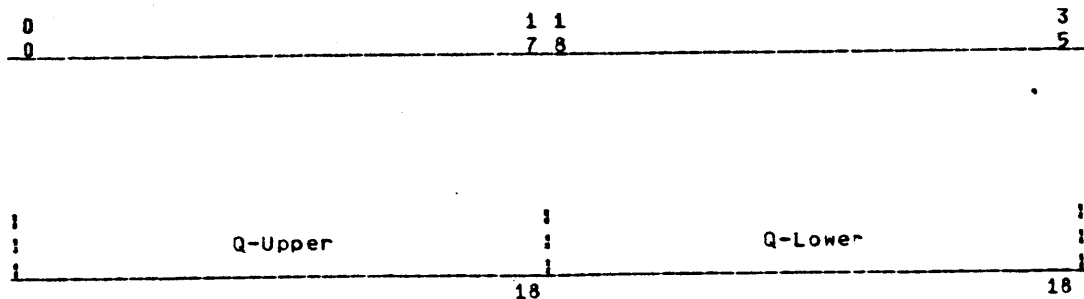


Figure 4-2 Quotient Register (Q) Format

Description:

A 36 bit physical register located in the Operations Unit.

Function:

In fixed point binary operations, holds operands and results.

In floating point binary operations, holds the least significant part of the mantissa.

In shifting operations, holds original data and shifted results.

In address preparation, may hold two logically independent word offsets, Q-Upper and Q-Lower, or an extended range bit or character offset.

ACCUMULATOR-QUOTIENT REGISTER (AQ)

Format: - 72 bits

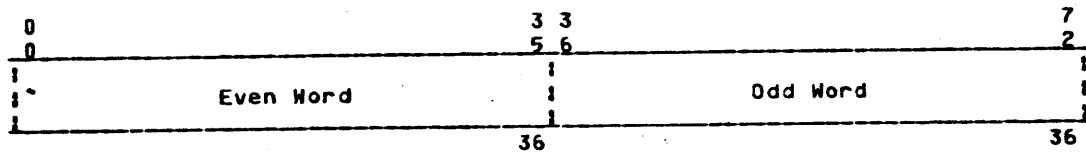


Figure 4-3 Accumulator-Quotient Register (AQ) Format

Description:

A logical combination of the Accumulator (A) and Quotient (Q) registers.

Function:

In fixed point binary operations, holds double precision operands and results.

In floating point binary operations, holds the mantissa.

In shifting operations, holds original data and shifted results.

EXPONENT REGISTER (E)

Format: - 8 bits

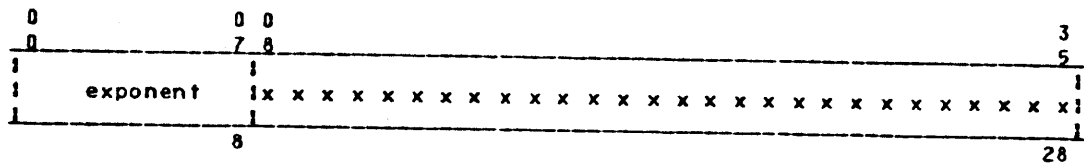


Figure 4-4 Exponent Register (E) Format

Description:

An 8 bit physical register located in the Operations Unit. Bits pictured as "x" are "don't care" bits, that is, are irrelevant to the register or its use.

Function:

In floating point binary operations, holds the exponent.

EXPONENT-ACCUMULATOR-QUOTIENT REGISTER (EAQ)

Format: - 80 bits

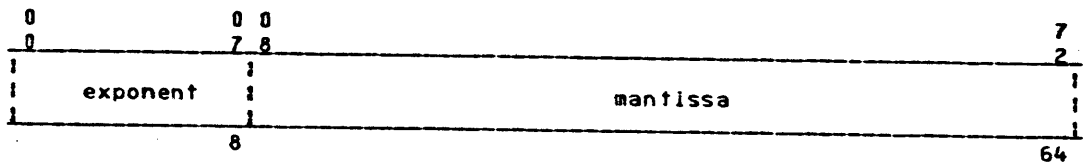


Figure 4-5 Exponent-Accumulator-Quotient Register (EAQ) Format

Description:

A logical combination of the Exponent (E), Accumulator (A), and Quotient (Q) registers. Although the register has a total of 80 bits, only 72 are

involved in transfers to and from main store. The low order 8 bits are truncated on store and zero filled on load.

Function:

In floating point binary operations, holds operands and results.

INDEX REGISTERS (Xn)

Format: - 18 bits each

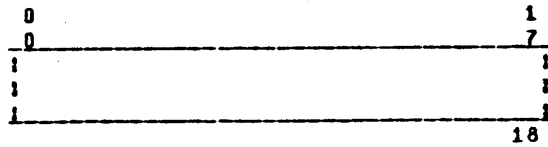


Figure 4-6 Index Register (Xn) Format

Description:

Eight 18 bit physical registers in the Operations Unit numbered 0 through 7. Index Register data may occupy the position of either an Upper or Lower 18-bit Half Word operand in a main store machine word.

Function:

In fixed point binary operations, hold half word operands and results.

In address preparation, hold word offsets or extended range bit or character offsets.

INDICATOR REGISTER (IR)

Format: - 14 bits

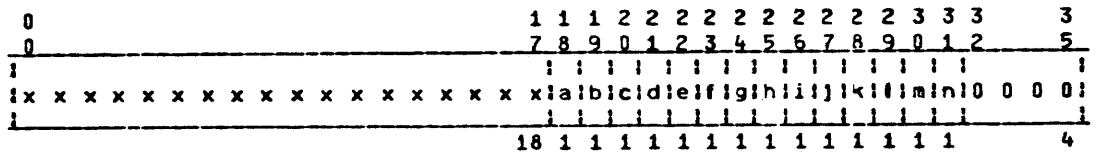


Figure 4-7 Indicator Register (IR) Format

Description:

A logical assemblage of 14 indicator flags from various Units of the Processor. The data occupies the position of a Lower 18-bit Half Word operand. Bits pictured as "x" are "don't care" bits and are irrelevant to the register or its use. Bits pictured as "0" are reserved and must have value 0. When interpreted as data, a bit value of 1 corresponds to the ON state of the indicator, a bit value of 0 corresponds to the OFF state.

Function:

The functions of the individual indicator bits are given below. An "x" in the column headed "L" indicates that the state of the indicator is not affected by instructions that load the IR.

key	L	Indicator_Name	Action
a		Zero	This indicator is set ON whenever the output of the main binary adder consists entirely of zero bits for binary or shifting operations or the output of the decimal adder consists of zero digits for decimal operations; otherwise, it is set OFF.
b		Negative	This indicator is set ON whenever the output of bit 0 of the main binary adder has value 1 for binary or shifting operations or the sign character of the result of a decimal operation is the negative sign character; otherwise, it is set OFF.
c		Carry	This indicator is set ON for any of the following conditions; otherwise, it is set OFF. (1) If a bit propagates leftward out of bit 0 of the main binary adder for any binary or shifting operation. (2) If !value1! =< !value2! for a decimal numeric comparison operation. (3) If char1 =< char2 for a decimal alphanumeric compare operation.
d		Overflow	This indicator is set ON if the arithmetic range of a register is exceeded in a fixed point binary operation or if the target string of a decimal numeric operation is too small to hold the integer part of the result. It remains ON until reset by the Transfer on Overflow (TOV) instruction or is reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an Overflow Fault. (See Overflow Mask indicator below.)
e		Exponent Overflow	This indicator is set ON if the exponent of the result of a floating point binary or decimal numeric operation is greater than +127. It remains ON until reset by the Transfer on Exponent Overflow (TEO) instruction or is reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an Overflow Fault. (See Overflow Mask indicator below.)
f		Exponent Underflow	This indicator is set ON if the exponent of the result of a floating point binary or decimal numeric operation is less than -128.

key L Indicator Name

Action

- It remains ON until reset by the Transfer on Exponent Underflow (TEU) instruction or is reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an Overflow Fault. (See Overflow Mask indicator below.)
- g Overflow Mask
- This indicator is set ON or OFF only by the instructions that load the IR. When set ON, it inhibits the generation of the fault for those events that normally cause an Overflow Fault. If the Overflow Mask indicator is set OFF after occurrence of an Overflow event, an Overflow Fault will not occur even though the indicator for that event is still set ON. The state of the Overflow Mask indicator does not affect the setting, testing, or storing of any other indicator.
- h Tally Runout
- This indicator is set OFF at initialization of any tallying operation, that is, any repeat instruction or any Indirect Then Tally Address Modification. It is then set ON for any of the following conditions:
- (1) If a repeat instruction terminates because of tally exhaust.
 - (2) If a Repeat Link (RPL) instruction terminates because of a zero link address.
 - (3) If a tally exhaust is detected for an Indirect Then Tally modifier. The instruction will be executed whether or not tally exhaust occurs.
- i Parity Error
- This indicator is set ON whenever the main store signals Illegal Actor with a parity error code or the Processor detects an internal parity error condition. The indicator is set OFF only by instructions that load the IR.
- j Parity Mask
- This indicator is set ON or OFF only by the instructions that load the IR. When it is set ON, it inhibits the generation of the Parity Fault for all events that set the Parity Error indicator. If the Parity Mask indicator is set OFF after the occurrence of a Parity Error event, a Parity Fault will not occur even though the Parity Error indicator
- may still be set ON. The state of the Parity Mask indicator does not affect the loading, testing, or storing of any other indicator, generated from previously set parity error indicators. The status of the parity mask indicator does not affect the setting, testing, or storing of the parity error indicator.

Key I Indicator Name

Action

k x Not BAR Mode

This indicator is set OFF only by execution of the Transfer and Set Slave (TSS) instruction that places the Processor in BAR Mode. It is set ON (taking the Processor out of BAR Mode) by the execution of any transfer class instruction other than ISS during a Fault or Interrupt Trap. However, if the Fault or Interrupt Trap occurs while in BAR Mode, and the transfer class instruction is Return (RET), Return Control Double (RTCD), or Restore Control Unit (RCU) and bit 28 of the saved IR data is 0, the Processor will remain in BAR Mode.

l Truncation

This indicator is set ON whenever the target string of a decimal numeric operation is too small to hold all the fraction digits of the result or the target string of an alphanumeric operation is too small to hold all the bits or characters to be stored. Also see the Overflow indicator condition for decimal numeric operations. The event that sets this indicator ON may also cause an Overflow Fault. (See Overflow Mask indicator above.)

m Mid Instruction Interrupt Fault

This indicator is set ON whenever the current instruction is interrupted by an external event. The indicator has meaning only when determining the proper restart resquence for the interrupted instruction. The indicator is set OFF at normal termination of every instruction. The events that set this indicator are:

- (1) An Access Violation Fault during Address Preparation for any operand.
- (2) Detection of the arrival of a Program Interrupt signal during execution of those EIS instructions that allow very long operand strings.

n x Absolute Mode

This indicator is set ON only by execution an absolute (non-appended) transfer class instruction during a Fault or Interrupt Trap and is set OFF by any execution of an appended transfer class instruction. However, if the Processor is not in Absolute Mode when the Fault or Interrupt occurs and the transfer class instruction is Return (RET), Return Control Double (RTCD), or

Restore Control Unit (RCU) and the appropriate mode bit is properly set in the IR data, the Processor will remain in its current mode.

Description:

A 27 bit settable, free running clock in the Control Unit. The value decrements at a rate of 512 kHz. Its range is 1.953125 microseconds to approximately 4.37 minutes. Bits pictured as "x" are "don't care" bits and are irrelevant to the register and its use.

Function:

The TR may be loaded with any convenient value with the privileged Load Timer (LDT) instruction. When the value next passes through zero, a Timer Runout Fault will be signalled. If the Processor is in Normal or BAR Mode with Program Interrupts not inhibited, the Fault will occur immediately. If the Processor is in Absolute or Privileged Mode or has Program Interrupts inhibited, the Fault will be delayed until the Processor returns to uninhibited Normal or BAR Mode.

RING ALARM REGISTER (RALR)

Format: - 3 bits

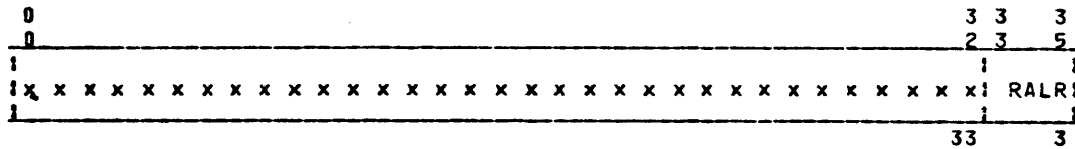


Figure 4-10 Ring Alarm Register (RALR) Format

Description:

A 3 bit physical register in the Appending Unit. The bits pictured as "x" are "don't care" bits and are irrelevant to the register or its use. The bits may have meaning with regard to other data structures.

Function:

If the Effective Ring Number (See TPR.TRR below) is greater than the contents of RALR an Access Violation, Ring Alarm, Fault will occur. The Multics supervisor uses this mechanism to assure the proper handling of User Ring events (such as QUITs) that occur while executing in the supervisor.

Key Register**Function**

greater (less privileged) than the current validation level of the process (as contained in PPR.PRR described below) then the Effective Ring Number for the access is PRn.RNR. The value of PRn.RNR is determined from directory entry information for the segment when the pointer data is constructed.

PRn.SNR

The Segment Number Register contains the segment number of the segment containing the data item described by the Pointer Register. The segment number is determined when the Segment Descriptor Word (SDW) is constructed from directory entry information for the segment.

PRn.WORDNO

The Word Number register contains the offset in machine words from the base or origin of the segment to the data item. The value is determined when the pointer data is constructed from the data item description in the procedure.

a PRn.CHAR

The Character register contains the number of the 9-bit character within the machine word at PRn.WORDNO containing the data item. The value is determined when the pointer data is constructed from the data item description in the procedure. Word boundary aligned data items will always have the value 0. Unaligned data items may have any value.

NOTE: The reader's attention is directed to the double definition of bits 18-26 of the Odd Word in the format above. Because the Multics Processor was implemented as an enhancement to an existing design, certain apparent anomalies appear. One of these is the difference in the handling of unaligned data items by the Appending Unit and Decimal Unit. The preexisting Decimal Unit handles all unaligned data items with a 9-bit character number plus bit offset with conversion from the description given in the EIS Operand Descriptor done automatically by the hardware. The Appending Unit maintains compatibility with the earlier generation Multics Processor by handling all unaligned data items with a bit offset from the prior word boundary; again with any necessary conversion done automatically by the hardware. Thus, a Pointer Register may be loaded from an ITS Pointer Pair having a pure bit offset and modified by one of the EIS Address Register instructions (A4BD, S9BD, etc.) using character displacement counts. When the results of such a modification are stored as an ITS Pointer Pair with SPRIn (or as a Packed Pointer with SPRPn), the BITNO field as indicated in the upper line of the format (bits 21-26) will contain a pure bit count. When the results are stored as an Address Register with

SARn the CHAR and BITNO fields as indicated in the lower line of the format (bits 18-23) will contain the character number plus bit offset.

WARNING: The Decimal Unit has builtin hardware checks for illegal bit offset values but the Appending Unit does not except for a single case for packed pointers. See NOTES for Load Packed Pointers (LPRPn) in Section II, Machine Instructions.

Key Register

Function

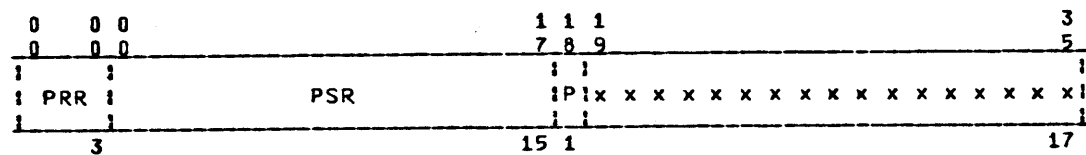
PRn.BITNO

The Bit Number register contains the number of the bit within PRn.CHAR of the word at PRn.WORDNO containing the data item. The value is determined when the pointer data is constructed from the data item description in the procedure. Word and character boundary aligned data items will always have the value 0. Unaligned data items may have any value in the range (0,10) octal. See NOTE under PRn.CHAR above.

PROCEDURE POINTER REGISTER (PPR)

Format: - 37 bits

Word 0 of Control Unit Data



Word 4 of Control Unit Data

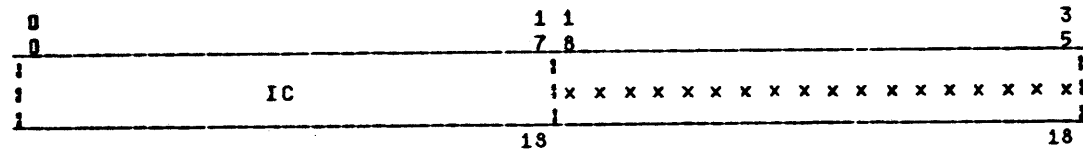


Figure 4-12 Procedure Pointer Register (PPR) Format

Description:

A logical combination of physical registers from the Appending Unit and the Control Unit. PPR.PRR, PPR.PSR, and PPR.P are located in the Appending Unit and PPR.IC is located in the Control Unit. The data is pictured as it appears in main store in Words 0 and 4 of Control Unit Data. Bits pictured as "x" are "don't care" bits and are irrelevant to the register or its use. The bits do have meaning with regard to Control Unit Data. (See Control Unit Data below.)

Function:

The Procedure Pointer Register holds information relative to the location in main store of the procedure segment in execution and the location of the current instruction within that segment. The functions of the individual constituent registers are:

<u>Register</u>	<u>Function</u>
PPR.PRR	The <u>Procedure Ring Register</u> contains the number of the ring (validation level) in which the process is executing. It is set to the Effective Ring Number of the procedure segment when control is transferred to the procedure.
PPR.PSR	The <u>Procedure Segment Register</u> contains the segment number of the procedure being executed. Its value changes every time control is transferred to a new procedure.
PPR.P	The <u>Privileged bit register</u> is a flag controlling execution of privileged instructions. Its value is "1"b (permitting privileged instructions) if PPR.PRR is 0 and the privileged bit in the Segment Descriptor word (SDW.P) for the procedure is "1"b. Its value is "0"b if SDW.P is 0 or PPR.PRR is greater than 0. Its value is set every time a new procedure is entered.
PPR.IC	The <u>Instruction Counter register</u> contains the word offset from the origin of the procedure segment to the current instruction.

"0" are reserved and must have the value 0.

Function

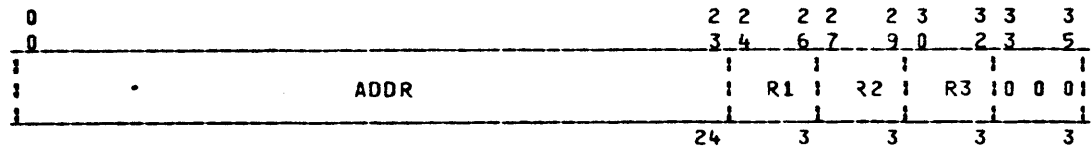
The Descriptor Segment Base Register contains information concerning the Descriptor Segment for a process. The Descriptor Segment holds the Segment Descriptor Words (SDWs) for all segments accessible by the process. The functions of its individual constituent registers are:

<u>Register</u>	<u>Function</u>
DSBR.ADDR	The interpretation of the <u>ADDRESS</u> register depends on the value of DSBR.U. For <u>DSBR.ADDR</u> contains U=0 The 24-bit main store address of the Page Table for the Descriptor Segment. U=1 The 24-bit main store address of the Descriptor Segment.
DSBR.BND	The <u>BOUND</u> register contains 14 most significant bits of the highest 16 word block of the Descriptor Segment that can be addressed without causing an Access Violation.
DSBR.U	The <u>U</u> register is a flag specifying whether the descriptor segment is unpagged (U = 1) or pagged (U = 0).
DSBR.STACK	The <u>STACK</u> register contains the upper 12 bits of the 15-bit stack base segment number. It is used only during the execution of the CALL6 instruction. (The Segment Number of the Stack Segment for a running process is given by $8 * \text{DSBR.STACK} + \text{PPR.PRR}$.)

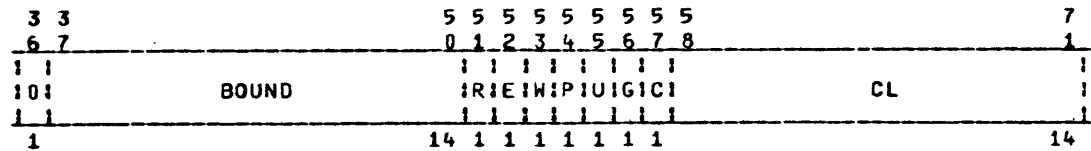
SEGMENT DESCRIPTOR WORD ASSOCIATIVE MEMORY (SDWAM)

Format: - 85 bits each

Even Word of Y-pairs as stored by Store Segment Descriptor Registers (SSDR)



Odd Word of Y-pairs as stored by Store Segment Descriptor Registers (SSDR)



Data as stored by Store Segment Descriptor Pointers (SSDP)

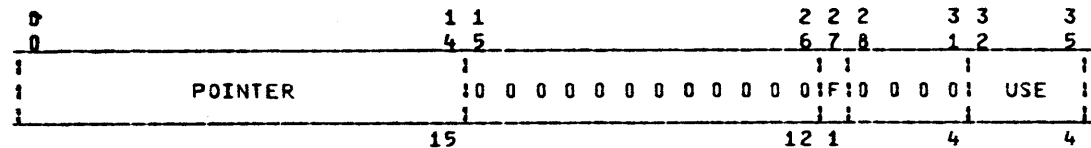


Figure 4-15 Segment Descriptor Word Associative Memory (SDWAM) Format

Description:

Sixteen logical combinations of registers and flags from the Appending Unit comprising the Segment Descriptor Word Associative Memory Assembly. The registers are numbered from 0 through 15 but are not directly addressable by number. Bits pictured as "0" are reserved and must have the value 0.

Function:

Hardware segmentation in the Multics Processor is implemented by the Appending Unit (See Section V, Address -- Segmentation and Paging for details). In order to permit addressing by Segment Number and offset as prepared in the Temporary Pointer Register (described above), a table containing the location and status of each accessible segment must be kept. This table is the Descriptor Segment and is unique to the process. The Descriptor Segment for a running process is located by information held in the Descriptor Segment Base Register (DSBR) described above.

Every time an Effective Segment Number (TPR.TSR) is prepared, it is used as an index into the Descriptor Segment to retrieve the Segment Descriptor Word (SDW) for the target segment. To reduce the number of main store references required for segment addressing, the SDWAM provides a content addressable store to hold the sixteen most recently referenced SDWs.

Whenever a reference to the SDW for a segment is required, the Effective Segment Number (TPR.TSR) is matched associatively against all 16 SDWAM.POINTER registers (described below). If the SDWAM match logic circuitry indicates a "hit", all usage counts (SDWAM.USE) greater than the usage count of the "hit" register are decremented by one, the usage count of the "hit" register is set to 15, and the contents of the "hit" register are read out into the address preparation circuitry as necessary. If the SDWAM match logic does not indicate a "hit", the SDW is fetched from main store and loaded into the SDWAM register with usage count 0 (the "oldest"), all usage counts are decremented by one with the newly loaded register rolling over from 0 to 15, and the newly loaded register is read out into the address preparation circuitry as necessary. Faulted SDWs are loaded into the SDWAM.

The functions of the constituent registers and flags of each SDWAM register are:

<u>Register</u>	<u>Function</u>
SDWAM.ADDR	The interpretation of the <u>ADDR</u> ess register depends on the value of SDWAM.U. <u>For SDWAM.ADDR contains</u> U=0 The 24-bit main store address of the Page Table for the target segment. U=1 The 24-bit main store address of the target segment.
SDWAM.R1	Upper limit of read/write Ring Bracket. (See Section VIII, Hardware Ring Implementation)
SDWAM.R2	Upper limit of read/execute Ring Bracket. (See Section VIII, Hardware Ring Implementation)
SDWAM.R3	Upper limit of call Ring Bracket. (See Section VIII, Hardware Ring Implementation)
SDWAM.BOUND	The upper limit of segment addresses stated as a number of 16 word blocks. A segment address (TPR.CA) with a block address larger than this value will cause an Access Violation, Out of Segment Bounds, Fault.
SDWAM.R	<u>R</u> ead permission bit. If this bit is set ON, read access requests may be honored.
SDWAM.E	<u>E</u> xecute permission bit. If this bit is set ON, the SDW may be loaded into the Procedure Pointer Register (PPR) and control transferred to the segment for execution.
SDWAM.W	<u>W</u> rite permission bit. If this bit is set ON, write access requests may be honored.
SDWAM.P	<u>P</u> rivileged flag bit. If this bit is set ON, privileged instructions from the segment may be executed if PPR.PRR is 0.

<u>Register</u>	<u>Function</u>
SDWAM.U	Unpaged flag bit. If this bit is set ON, the segment is unpaged and SDWAM.ADDR is the 24-bit main store address of the base of the segment. If this bit is set OFF, the segment is paged and SDWAM.ADDR is the 24-bit address the array of Page Table Words (PTWs) for the segment.
SDWAM.G	Gate control bit. If this bit is set ON, calls into the segment must be to an offset no greater than the value of SDWAM.CL as described below.
SDWAM.C	Cache control bit. If this bit is set ON, data from the segment may be placed in the cache store.
SDWAM.CL	Call Limiter value. If the segment is gated (SDWAM.G set ON), transfers of control into the segment must be to segment addresses no greater than this value.
SDWAM.POINTER	The Effective Segment Number used to fetch this SDW from main store.
SDWAM.F	Full/empty bit. If this bit is set ON, the SDW in the register is valid. If this bit is set OFF, a "hit" is not possible. All SDWAM.F bits are set OFF by the instructions that clear the SDWAM.
SDWAM.USE	Usage count for the register. The SDWAM.USE field is used to maintain a strict FIFO queue order among the SDWs. When an SDW is matched its USE value is set to 15 ("newest") and the queue is reordered. SDWs newly fetched from main store replace the SDW with USE value 0 ("oldest") and the queue is reordered. SDWAM.USE is set the internal (and invisible) SDWAM register number by instructions that clear the SDWAM.

TPR.TSR is matched against PTWAM.POINTER (described below). If the PTWAM match logic circuitry indicates a "hit", all usage counts (PTWAM.USE) greater than the usage count of the "hit" register are decremented by one, the usage count of the "hit" register is set to 15, and the contents of the "hit" register are read out into the address preparation circuitry as necessary. If the PTWAM match logic does not indicate a "hit", the PTW is fetched from main store and loaded into the PTWAM register with usage count 0 (the "oldest"), all usage counts are decremented by one with the newly loaded register rolling over from 0 to 15, and the newly loaded register is read out into the address preparation circuitry as necessary. Faulted PTWs are not loaded into the PTWAM.

The functions of the constituent registers and flags of each PTWAM register are:

<u>Register</u>	<u>Function</u>																
PTWAM.ADDR	The ADDRESS register holds the 18 most significant bits of the 24-bit main store address of the page. The hardware ignores low order bits of the page address according to page size based on the following ...																
	<table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">Page Size in words</th> <th style="text-align: left;">ADDR Bits ignored</th> </tr> </thead> <tbody> <tr><td>64</td><td>none</td></tr> <tr><td>128</td><td>17</td></tr> <tr><td>256</td><td>16-17</td></tr> <tr><td>512</td><td>15-17</td></tr> <tr><td>1024</td><td>14-17</td></tr> <tr><td>2048</td><td>13-17</td></tr> <tr><td>4096</td><td>12-17</td></tr> </tbody> </table>	Page Size in words	ADDR Bits ignored	64	none	128	17	256	16-17	512	15-17	1024	14-17	2048	13-17	4096	12-17
Page Size in words	ADDR Bits ignored																
64	none																
128	17																
256	16-17																
512	15-17																
1024	14-17																
2048	13-17																
4096	12-17																
PTWAM.M	Page Modified flag bit. This bit is set ON whenever the PTW is used for a store type instruction. When the bit changes value from 0 to 1, a special extra cycle is generated to write it back into the PTW in the PTWA.																
PTWAM.POINTER	The Effective Segment Number used to fetch this PTW from main store.																
PTWAM.PAGENO	The 12 most significant bits of the 18-bit Effective Address (TPR.CA) used to fetch this PTW from main store. Low order bits are forced to zero by the hardware and not used as part of the PTWA index according to page size based on the following ...																
	<table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">Page Size in words</th> <th style="text-align: left;">PAGENO bits forced</th> </tr> </thead> <tbody> <tr><td>64</td><td>none</td></tr> <tr><td>128</td><td>11</td></tr> <tr><td>256</td><td>10-11</td></tr> <tr><td>512</td><td>09-11</td></tr> <tr><td>1024</td><td>08-11</td></tr> <tr><td>2048</td><td>07-11</td></tr> <tr><td>4096</td><td>06-11</td></tr> </tbody> </table>	Page Size in words	PAGENO bits forced	64	none	128	11	256	10-11	512	09-11	1024	08-11	2048	07-11	4096	06-11
Page Size in words	PAGENO bits forced																
64	none																
128	11																
256	10-11																
512	09-11																
1024	08-11																
2048	07-11																
4096	06-11																
PTWAM.F	Full/empty bit. If this bit is set ON, the PTW in the register is valid. If this bit is set OFF, a "hit" is not possible. All PTWAM.F bits are set OFF by the instructions that clear the PTWAM.																
PTWAM.USE	USAgE count for the register. The PTWAM.USE field is used to maintain a strict FIFO queue order among the																

<u>key Register</u>	<u>Function</u>
a ILL OP	An illegal operation code has been detected.
b ILL MOD	An illegal Address Modifier has been detected.
c ILL SLV	An illegal BAR Mode procedure has been encountered.
d ILL PROC	An illegal procedure other than BAR Mode has been encountered.
e NEM	A nonexistent main store address has been requested.
f OOB	A BAR Mode boundary violation has occurred.
g WRT INH	An illegal decimal digit has been detected by the Decimal Unit. (Flag name is obsolete)
h PROC PARU	A parity error has been detected in the upper 36 bits of data.
i PROC PARL	A parity error has detected in the lower 36 bits of data.
j \$CON A	A \$CONNECT signal has been received through port A.
k \$CON B	A \$CONNECT signal has been received through port B.
l \$CON C	A \$CONNECT signal has been received through port C.
m \$CON D	A \$CONNECT signal has been received through port D.
n DA ERR1	CPU/SCU interface sequence error 1 has been detected. (\$DATA-AVAIL received with no prior \$INTERRUPT sent.)
o DA ERR2	CPU/SCU interface sequence error 2 has been detected. (Multiple \$DATA-AVAIL received or \$DATA-AVAIL received out of order.)
IAA	Coded Illegal Action, Port A. (See Table 4-2 below)
IAB	Coded Illegal Action, Port B. (See Table 4-2 below)
IAC	Coded Illegal Action, Port C. (See Table 4-2 below)
IAD	Coded Illegal Action, Port D. (See Table 4-2 below)
p CPAR DIR	A parity error has been detected in the cache store directory.
q CPAR STR	A data parity error has been detected in the cache store.
r CPAR IA	An Illegal Action has been received from an SCU during a store operation.
s CPAR BLK	A cache parity error has occurred during a cache store data block load.

Key Register

Function

FFV

A "floating fault vector" address. The 15 most significant bits of the Y-block8 address of four word pairs constituting a "floating fault vector". Traps to these floating faults are generated by other conditions setable by the mode register.

a OC TRAP

Trap on OPCODE match. If this bit is set ON and OPCODE matches the operation code of the instruction for which an address is being prepared (including indirect cycles), generate the second floating fault (XED FFV+2). (See NOTE below)

b ADR TRAP

Trap on ADDRESS match. If this bit is set ON and the Computed Address (TPR.CA) matches the setting of the Address Switches on the Processor Maintenance panel, generate the fourth floating fault (XED FFV+6). (See NOTE below)

OPCODE

The operation code on which to trap if OC TRAP (bit 16, key a) is set ON or for which to strobe all CU cycles into the CU History Registers if O.CSt (bit 29, key j) is set ON

OC

Processor conditions codes as follows if OC TRAP (bit 16, key a) and O.CSt (bit 29, key j) are set OFF and t VOLTAGE (bit 32, key m) is set ON.

Key Condition

c Set Control Unit Overlap Inhibit if set ON. The Control Unit shall wait for the operations Unit to complete execution of the even instruction of the current instruction pair before it begins address preparation for the associated odd instruction. The Control Unit shall also wait for the Operations Unit to complete execution of the odd instruction before it fetches the next instruction pair.

d Set Store Overlap Inhibit if set ON. The Control Unit shall wait for completion of a current main store fetch (read cycles only) before requesting a main store access for another fetch.

e Set Store Incorrect Data Parity if set ON. The Control Unit shall cause incorrect data parity to be sent to the SCU for the next data store instruction and then shall reset bit 20.

f Set Store Incorrect ZAC Parity if Set ON. The

Control Unit shall cause incorrect Zone-Address-Command (ZAC) parity to be sent to the SCU for each main store cycle of the next data store instruction and shall reset bit 21 at the end of the instruction.

g Set Timing Margins if set ON. If t VOLT (bit 32, key m) is set ON and the Margin Control switch on the Processor Maintenance panel is in PROG position, set Processor timing margins as follows.

key Register

Function

key Condition

<u>22,33</u>	<u>margin</u>
0,0	normal
0,1	slow
1,0	normal
1,1	fast

h Set +5 Voltage Margins if set ON. If & VOLT (bit 32, key m) is set ON and the Margin Control switch on the Processor Maintenance panel is in the PROG position, set +5 voltage margins as follows.

<u>24,25</u>	<u>margin</u>
0,0	normal
0,1	low
1,0	high
1,1	normal

i Trap on Control Unit History Register count overflow if set ON. If this bit and STROBE & (bit 30, key k) are set ON and the Control Unit History Register counter overflows, generate the third floating fault (XED FFV+4). Further, if FAULT RESET (bit 31, key l) is set, reset STROBE & (bit 30, key k), locking the history registers. An LCPR, TAG = 04, instruction setting bit 28 ON will reset the Control Unit History Register counter to zero. (See NOTE below)

j O.C& Strobe Control Unit History Registers on OPCODE match. If this bit and STROBE & (bit 30, key k) are set ON and the operation code of the current instruction matches OPCODE, strobe the Control Unit History Registers on all Control Unit cycles (including indirect cycles).

k STROBE & Enable history registers. If this bit is set ON, all history registers are strobed at appropriate points in the various Processor cycles. If this bit is set OFF or MR ENABLE (bit 35, key n) is set OFF, all history registers are locked. This bit is set OFF with an LCPR, TAG = 04, instruction providing a "zero" bit, by an Op Not Complete Fault, and, conditionally, by other faults (See FAULT RESET (bit 31, key l) below). Once set OFF, this bit must be set ON with an LCPR, TAG = 04, instruction providing a "one" bit before the history registers again become active.

l FAULT RESET History register lock control. If this bit is set ON, set STROBE & (bit 30, key k) OFF, locking the history registers, for all faults including the floating faults. (See NOTE below)

m & VOLT Test mode indicator. This bit is set ON whenever the Test/Normal switch on the Processor Maintenance panel is in Test position and is set OFF otherwise. It serves to enable the program control of Voltage and Timing Margins.

n MR ENABLE Enable mode register. When this bit is set ON, all other bits and controls of the mode register are active. When this bit is set OFF, the mode register controls are disabled.

NOTE: The traps described above (Address match, OPCODE match, Control Unit History Register counter overflow) occur after completion of the next odd instruction following their detection. They are handled as Group VII faults in regard to servicing and inhibition. The complete Group VII priority sequence is...

- 1 - con
- 2 - tro
- 3 - sdf
- 4 - OPCODE trap
- 5 - Control Unit History Register counter overflow
- 6 - Address match trap
- 7 - External interrupts

CACHE MODE REGISTER (CMR)

Format: - 28 bits

Odd word of Y-pair as stored by Store Central Processor Register (SCPR), TAG = 06, instruction

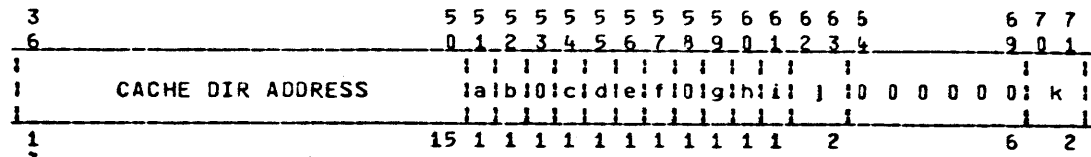


Figure 4-19 Cache Mode Register (CMR) Format

Descriptions:

A logical assemblage of flags and registers from the control unit. The Mode Register and Cache Mode Register are both stored into the Y-pair by the SCPR, TAG = 06, instruction.

The Cache Mode Register data stored is address dependent. The algorithm used to map main store into the cache store (See Section XX, Cache Store) is effective for the SCPR instruction. In general, the user may read out data from the directory entry for any cache block by proper selection of certain subfields of the final 24-bit main store address. In particular, the user may read out the directory entry for the cache block involved in a suspected cache error by assuring that the required 24-bit final address subfields are the same as those for the access which produced the suspected error.

WARNING: The user is warned that the fault handling procedure(s) should be unencachable (SDW.C = 0) and that the History Registers and cache should be disabled as quickly as possible in order that vital information concerning the suspected error not be lost.

The Cache Mode Register is loaded with the Load Central Processor Register (LCPR), TAG = 02, instruction. Those items with an "x" in the column headed L are not loaded with the LCPR instruction. Bits pictured as "0" are reserved and must have the value 0.

The functions of the constituent flags and registers are:

<u>Key</u>	<u>L Register</u>	<u>Function</u>
	x CACHE DIR ADDRESS	15 most significant bits of the block address from the cache directory
a	x PAR BIT	Cache directory parity error on this read out
b	x LEV FUL	The selected column and level is loaded with active data
c	CSH1 ON	Enable the upper 1024 words of the cache
d	CSH2 ON	Enable the lower 1024 words of the cache
e	OPND ON	Enable the cache for operands
f	INST ON	Enable the cache for instructions
g	CSH REG	Enable cache-to-register (dump) mode When this bit is set ON, double precision Operations Unit operands (e.g., LDAQ operands) are read from the cache according to the mapping algorithm and without regard to matching of the full final address. All other operands address main store as though the cache were disabled. This bit is reset automatically by the hardware for any Fault or Program Interrupt.
h	x STR ASD	Enable store aside When this bit is set ON, the Processor does not wait for main store cycle completion after a store operation but proceeds after the cache cycle is complete.
i	x COL FUL	Selected cache column is full
j	x RRD A,B	Cache round robin counter
k	LUF MSB,LSB	Lockup timer setting The Lockup Timer may set to four different values according to the setting of this field.

<u>LUF Value</u>	<u>Lockup Time</u>
0	2 ms.
1	4 ms.
2	8 ms.
3	16 ms.

The Lockup Timer is set to 16 ms. when the Processor is initialized.

CONTROL UNIT (CU) HISTORY REGISTERS

Format: - 72 bits each

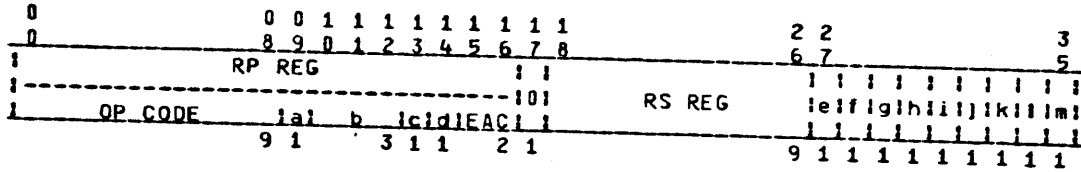
Even word as stored by Store Central Processor Register (SCPR), Tag = 20, instruction

<u>Key Flag Name</u>	<u>Meaning</u>
i TRGO	transfer GO (conditions met)
j XDE	execute XED from even IC
k XOD	execute XED from odd IC
l IC	execute odd instruction of the current pair
m RPTS	execute a repeat operation
n WI	wait for instruction fetch
o AR F/E	1 = Computed Address (TPR.CA) has valid data
p <u>XIP</u>	NOT prepare Program Interrupt address
q <u>FLT</u>	NOT prepare Fault address
r <u>BASE</u>	NOT BAR mode
OPCODE	current operation code
I	Program Interrupt inhibit bit
P	Pointer register flag bit
TAG	Current address modifier This modifier is replaced by the contents of the TAG fields of indirect words as they are fetched during indirect chains.
ADDRESS	Current Computed Address (TPR.CA)
CMD	SCU command
SEL selected)	Port select bits. (Valid only if Port A through D is selected)
s XEC-INT	A Program Interrupt is present
t INS-FETCH	Perform an instruction fetch
u CU-STORE	Control Unit store cycle
v OU-STORE	Operations Unit store cycle
w CU-LOAD	Control Unit load cycle
x OU-LOAD	Operations Unit load cycle
y <u>DIRECT</u>	direct cycle
z PC-BUSY	Port control logic not busy
* BUSY	Port interface busy

OPERATIONS UNIT (OU) HISTORY REGISTERS

Format: - 72 bits each

Even word as stored by Store central Processor Register (SCPR), TAG = 40, instruction



Odd word as stored by Store Central Processor Register (SCPR), TAG = 40, instruction

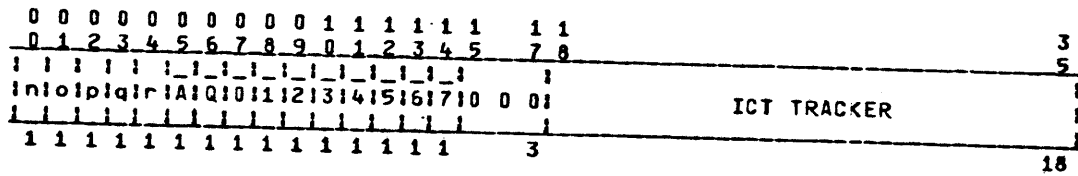


Figure 4-21 Operations Unit (OU) History Register Format

Description:

Sixteen logical combinations of flags and registers from the Operations Unit and Control Unit. The sixteen registers are handled as a rotating queue controlled by the Operations Unit History Register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or SCPR).

Function:

An Operations Unit History Register entry shows the conditions at the end of the Operations Unit cycle to which it applies. The sixteen registers will hold the conditions for the last sixteen Operations Unit cycles. Entries are made according to controls set in the Mode Register. (See Mode Register above)

The meanings of the constituent flags and registers are:

<u>Key Flag Name</u>	<u>Meaning</u>
RP REG	Primary Operations Unit operation register RP REG receives the instruction operation code and other data from the Control Unit during the Control Unit instruction cycle while the Operations Unit may be busy with a prior operation. RP REG is further sub-structured as ...

<u>key</u>	<u>Flag Name</u>	<u>Meaning</u>
	OP CODE	The 9 most significant bits of the operation code for the instruction. Note that basic (non EIS) operations do not involve bit 27 hence the 9 bit field is sufficient to define the operation code.
a	9 CHAR	Character size for Indirect Then Tally modifiers 0 = 6-bit 1 = 9-bit
b	TAG1,2,3	The 3 least significant bits of the modifier of the instruction. This field may contain a character position for an Indirect Then Tally character modifier.
c	CR FLG	Character operation flag
d	DK FLG	Direct operation flag
	EAC	Effective address counter for LREG/SREG instructions
	RS REG	Secondary Operations Unit operation register OP CODE is moved from RP REG to RS REG during the operand fetch and is held until completion of the instruction.
e	RB1 FULL	OP CODE buffer full
f	RP FULL	RP REG full
g	RS FULL	RS REG full
h	GIN	First cycle for all Operations Unit operations
i	GOS	Second cycle for Operations Unit multi-ops
j	GD1	First divide cycle
k	GD2	Second divide cycle
l	GOE	Exponent compare cycle
m	GOA	Mantissa alignment cycle
n	GOM	General Operations Unit cycle
o	GON	Normalize cycle
p	GOF	Final Operations Unit cycle
q	STR OP	Operations Unit store data available
t	DA-AV	Data not available
-	-----	
A	A-REG	A register not in use
Q	Q-REG	Q register not in use
0	X0-RG	X0 not in use
1	X1-RG	X1 not in use
2	X2-RG	X2 not in use
-	-----	

<u>key</u>	<u>Flag Name</u>	<u>Meaning</u>
3	X3-RG	X3 not in use
4	X4-RG	X4 not in use
5	X5-RG	X5 not in use
6	X6-RG	X6 not in use
7	X7-RG	X7 not in use

ICT TRACKER The current value of the Instruction Counter (PPR.IC). Since the Control Unit and Operations Unit run asynchronously and overlap is usually enabled, the value of ICT TRACKER may not be the address of the Operations Unit instruction currently being executed.

DECIMAL UNIT (DU) HISTORY REGISTERS

Format: - 72 bits each

Decimal Unit History Register data is stored with the Store Central Processor Register (SCPR), TAG = 60, instruction. No Format diagram is given because the data is defined as individual bits.

Description:

Sixteen logical combinations of flags from the Decimal Unit. The sixteen registers are handled as a rotating queue controlled by the Decimal Unit History Register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or SCPR).

The Decimal Unit and the Control Unit run synchronously. There is a Control Unit History Register entry for every Decimal Unit History Register entry and vice versa. If the Processor is not executing a Decimal operation, the Decimal Unit History Register entry will show an idle condition.

Function:

A Decimal Unit History Register entry shows the conditions in the Decimal Unit at the end of the Control Unit cycle to which it applies. The sixteen registers will hold the conditions for the last sixteen Control Unit cycles. Entries are made according to controls set in the Mode Register.

(See Mode Register above)

A minus (-) sign preceding the flag name indicates that the complement of the flag is shown. Unused bits are set ON.

The meanings of the constituent flags are:

<u>bit</u>	<u>Flag Name</u>	<u>Meaning</u>
0	-FPOL	Prepare operand length
1	-FPOP	Prepare operand pointer
2	-NEED-DESC	Need descriptor
3	-SEL-ADR	Select address register
4	-DLEN=DIRECT	Length equals direct
5	-DFRST	Descriptor processed for first time
6	-FEXR	Extended register modification
7	-DLAST-FRST	Last cycle of DFRST
8	-DDU-LDEA	Decimal Unit load
9	-DDU-STAE	Decimal Unit store
10	-DREDO	Redo operation without pointer and length update
11	-DLVL<WD-SZ	Load with count less than word size
12	-EXH	Exhaust
13	DEND-SEQ	End of sequence
14	-DEND	End of instruction
15	-DU=RD+WRT	Decimal Unit write-back
16	-PTRAO0	PR address bit 0
17	-PTRAO1	PR address bit 1
18	FA/I1	Descriptor 1 active
19	FA/I2	Descriptor 2 active
20	FA/I3	Descriptor 3 active
21	-WRD	Word operation
22	-NINE	9-bit character operation
23	-SIX	6-bit character operation
24	-FOUR	4-bit character operation
25	-BIT	Bit operation
26		Unused
27		Unused
28		Unused
29		Unused
30	FSAMPL	Sample for mid-instruction interrupt

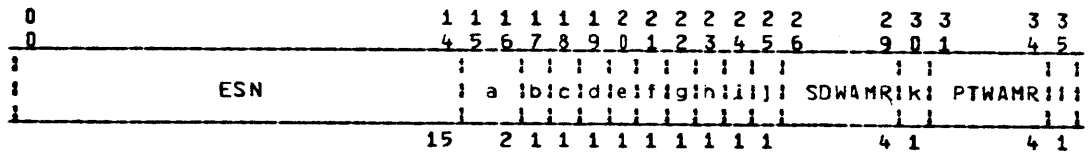
<u>bit</u>	<u>Flag Name</u>	<u>Meaning</u>
31	-DFRST-CT	Specified first count of a sequence
32	-ADJ-LENGTH	Adjust length
33	-INTRPTD	Mid-instruction interrupt
34	-INHIB	Inhibit STC1 (force "STC0")
35		Unused
36	DUD	Decimal Unit idle
37	-GDLDA	Descriptor load gate A
38	-GDLDB	Descriptor load gate B
39	-GDLDC	Descriptor load gate C
40	NLD1	Prepare alignment count for first numeric operand load
41	GLDP1	Numeric operand one load gate
42	NLD2	Prepare alignment count for second numeric operand load
43	GLDP2	Numeric operand two load gate
44	ANLD1	Alphanumeric operand one load gate
45	ANLD2	Alphanumeric operand two load gate
46	LDWRT1	Load rewrite register one gate
47	LDWRT2	Load rewrite register two gate
48	-DATA-AVLDU	Decimal Unit data available
49	WRT1	Rewrite register one loaded
50	GSTR	Numeric store gate
51	ANSTR	Alphanumeric store gate
52	FSTR-OP-AV	Operand available to be stored
53	-FEND-SEQ	End sequence flag
54	-FLEN<128	Length less than 128
55	FGCH	Character operation gate
56	FANPK	Alphanumeric packing cycle gate
57	FEXMOP	Execute MOP gate
58	FBLNK	Blanking gate
59		Unused
60	DGBD	Binary to decimal execution gate
61	DGDB	Decimal to binary execution gate

bit	Flag Name	Meaning
62	DGSP	Shift procedure gate
63	FFLTG	Floating result flag
64	FRND	Rounding flag
65	DADD-GATE	Add/subtract execute gate
66	DMP+DV-GATE	Multiply/divide execution gate
67	DXPN-GATE	Exponent network execution gate
68		Unused
69		Unused
70		Unused
71		Unused

APPENDING UNIT (AU) HISTORY REGISTERS

Format: - 72 bits each

Even word as stored by Store Central Processor Register (SCPR), TAG = 00, instruction



Odd word as stored by Store Central Processor Register (SCPR), TAG = 00, instruction

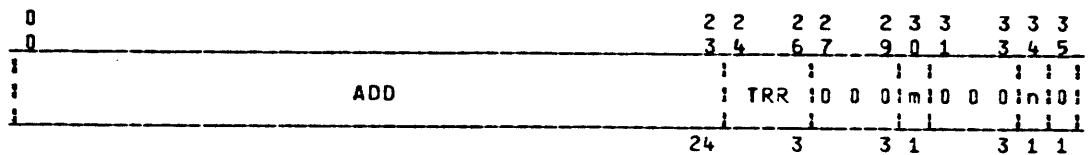


Figure 4-22 Appending Unit (AU) History Register Format

Description:

Sixteen logical combinations of flags and registers from the Appending Unit. The sixteen registers are handled as a rotating queue controlled by the Appending Unit History Register counter. The counter is always set to the number of the oldest entry and advances by one for each history

register reference (data entry or SCPR).

Function

An Appending Unit History Register entry shows the conditions in the Appending Unit at the end of an address preparation cycle in Appending Mode. The sixteen registers will hold the conditions for the last sixteen such address preparation cycles. Entries are made according to controls set in the Mode Register. (See Mode Register above)

The meanings of the constituent flags and registers are:

<u>bit</u>	<u>Flag Name</u>	<u>Meaning</u>
	ESN	Effective segment number (TPR.TSR)
a	BSY	Data source for ESN 00 = from PPR.PSR 01 = from PRn.SNR 10 = from TPR.TSR 11 = not used
b	FDSTPW	Descriptor segment PTW fetch
c	MDSPTW	Descriptor segment PTW modification
d	FSDWP	SDW fetch from paged descriptor segment
e	FPTW	PTW fetch
f	FPTW2	PTW+1 fetch
g	MPTW	PTW modification
h	FANP	Final address fetch from non-paged segment
i	FAP	Final address fetch from paged segment
j	SDWAMM	SDWAM match occurred
	SDWAMR	SDWAM register number for SDWAMM=1
k	PTWAMM	PTWAM match occurred
	PTWAMR	PTWAM register number for PTWAMM=1
l	FLT	ACV or DFT _n fault on this cycle
	ADD	24 bit final address from this cycle
	TRR	Ring number from this cycle (TPR.TRR)
m	CA	Segment is encacheable
n	FHLD	An ACV or DFT _n is waiting

Read Switches (RSW), Y = xxxxx1 reads data for Ports A, B, C, and D. Read Switches (RSW), Y = xxxxx3 reads data for Ports E, F, G, and H.

Function:

The meanings of the constituent fields are:

<u>key</u>	<u>Field Name</u>	<u>Meaning</u>
	FLT BASE	Seven most significant bits of the 12 bit Fault Base Address
a		Cache option 0 = enabled 1 = disabled
b		Main store speed option 0 = slow 1 = fast
	CPU	Processor number
	PORT A/E, etc.	Port data fields further substructured as...
	ADR	Address Assignment Switch setting for port
c		Port enabled flag
d		System Initialize enabled flag
e		Interlace enabled flag
	MEM	Coded memory size ...
		000 32K
		001 64K
		010 96K or 160K
		011 128K
		100 512K
		101 1024K
		110 2048K
		111 256K
	A, B, etc.	Port data fields further substructured as...
f		Interlace mode 0 = 4 word if interlace enabled for port 1 = 2 word if interlace enabled for port
g		Main store size 0 = full, all of MEM is configured

1 = half, half of MEM is configured

CONTROL UNIT DATA

Format: - 288 bits, 8 machine words

Data as stored by Store Control Unit (SCU) instruction

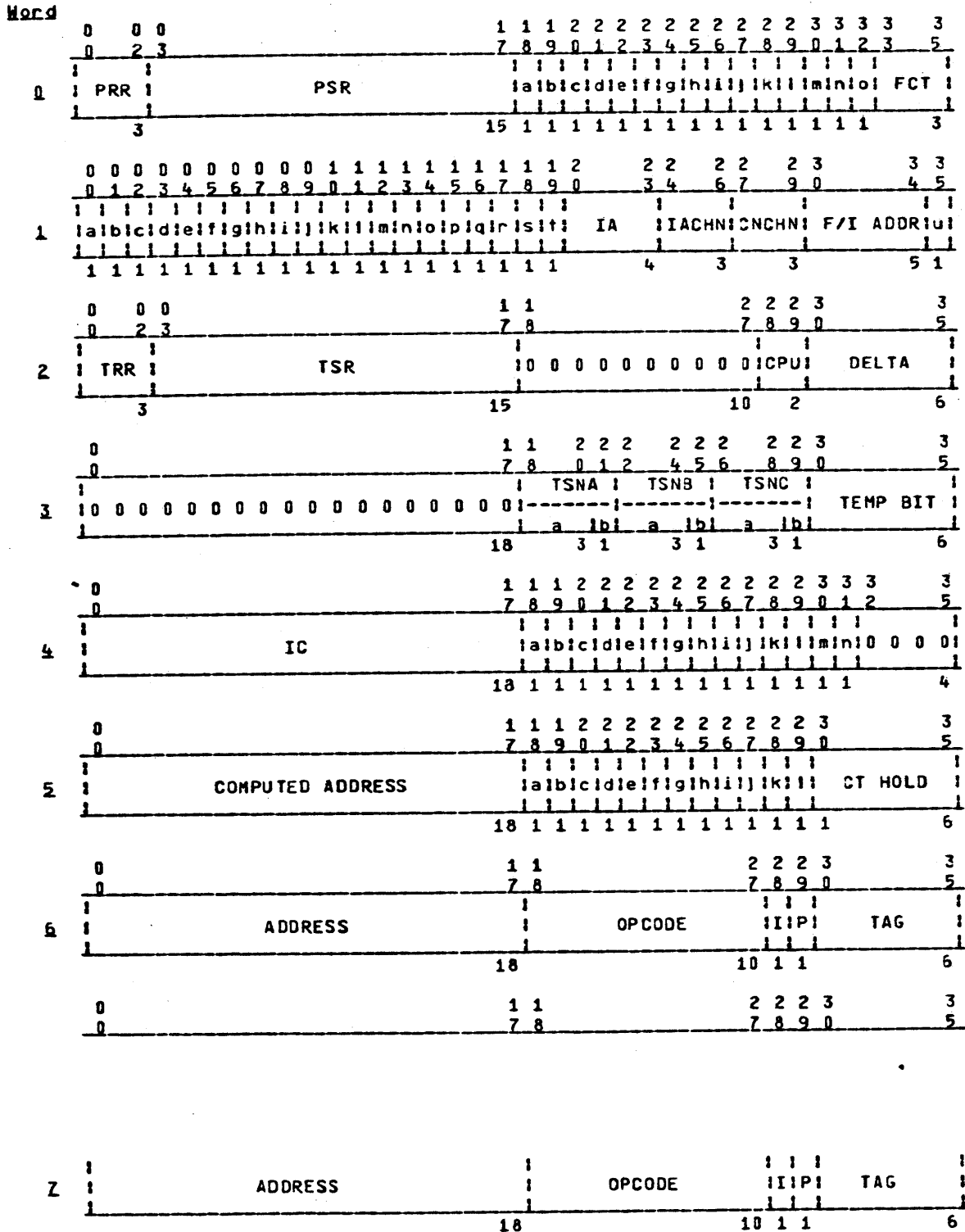


Figure 4-24 Control Unit Data Format

Description:

REVIEW DRAFT
 SUBJECT TO CHANGE
 October, 1975

A logical collection of flags and registers from the Appending Unit and the Control Unit. In general, the data has valid meaning only when stored with the Store Control Unit (SCU) instruction as the first instruction of a Fault Trap pair. Bits pictured as "0" are reserved and must have the value 0.

Functions

The Control Unit Data allows the Processor to restart an instruction at the point of interruption when it is interrupted by an Access Violation Fault, a Directed Fault, or (for certain EIS instructions) a Program Interrupt. Directed Faults are intentional, and most Access Violation Faults and Program Interrupts are recoverable. If the interruption is not recoverable, the Control Unit Data provides enough information to determine the exact nature of the error.

Instruction execution restarts immediately upon execution of a Restore Control Unit (RCU) instruction referencing the Y-block8 area into which the Control Unit Data was stored.

Fields having an "x" in the column headed L are not restored by the Restore Control Unit (RCU) instruction.

Word key	L	Field Name	Meaning
0		PPR.PRR	Procedure ring register
0		PPR.PSR	Procedure segment register
0	a	PPR.P	Privileged bit
0	b	XSF	External segment flag
0	c	x SDWAM.SDWAMM	Match on SDWAM
0	d	x SD-ON	SDWAM enabled
0	e	x PTWAM.PTWAMM	Match on PTWAM
0	f	x PT-ON	PTWAM enabled
0	g	x PI-AP	Instruction fetch append cycle
0	h	x DSPTW	Fetch Descriptor Segment PTW
0	i	x SDWNP	Fetch SDW - nonpaged
0	j	x SDWP	Fetch SDW - paged
0	k	x PTW	Fetch PTW
0	l	x PTW2	Fetch prepage PTW
0	m	x FAP	Fetch final address - paged
0	n	x FANP	Fetch final address - nonpaged
0	o	x FABS	Fetch final address - absolute
0		FCT	Fault counter - counts instruction retries

<u>Word</u>	<u>key</u>	<u>L</u>	<u>Field Name</u>	<u>Meaning</u>
1	a	x	IRO x ISN	For ACV - illegal ring order For STR - illegal segment number
1	b	x	OEB x IOC	For ACV - out of execute bracket For IPR - illegal op code
1	c	x	E-OFF x IA+IM	For ACV - execute bit is off. For IPR - illegal address or modifier
1	d	x	ORB x ISP	For ACV - out of read bracket For IPR - illegal slave procedure
1	e	x	R-OFF x IPR	For ACV - read bit is off For IPR - illegal EIS digit
1	f	x	OWB x NEA	For ACV - out of write bracket For STR - nonexistent address
1	g	x	W-OFF x OOB	For ACV - write bit is off For STR - out of bounds
1	h	x	NO GA	For ACV - not a gate
1	i	x	OCB	For ACV - out of call bracket
1	j	x	OCALL	For ACV - outward call
1	k	x	BOC	For ACV - bad outward call
1	l	x	INRET	For ACV - inward return
1	m	x	CRT	For ACV - cross ring transfer
1	n	x	RALR	For ACV - ring alarm
1	o	x	AM-ER	For ACV - associative memory error
1	p	x	OOSB	For ACV - out of segment bounds
1	q	x	PARU	For PAR - processor parity upper
1	r	x	PARL	For PAR - processor parity lower
1	s	x	ONC1	For ONC - CPU/SCU sequence error #1
1	t	x	ONC2	For ONC - CPU/SCU sequence error #2
1			x IA	SCU illegal action lines (See Table 4-2)
1			x IACHN	Illegal action CPU port.
1			x CNCHN	For CON - connect (CIOC) CPU port
1			x F/I ADDR	Modulo 2 fault/interrupt vector address
1	u	x	F/I	Fault/interrupt bit flag bit 0 = interrupt 1 = fault
2			TPR.TRR	Temporary ring register

<u>Word key</u>	<u>L</u>	<u>Field Name</u>	<u>Meaning</u>
2		TPR.TSR	Temporary segment register
2		CPU	CPU number
2		DELTA	Address increment for repeats
3		TSNA	Pointer Register number for non-EIS operands or for EIS operand #1 further substructured as...
3	a	PRNO	Pointer register number
3	b	----	1 = PRNO is valid
3		TSNB	Pointer Register number for EIS operand #2 further substructured as for TSNA above
3		TSNC	Pointer Register number for EIS operand #3 further substructured as for TSNA above
3		TEMP BIT	BITNO field of Temporary Pointer Register (TPR.TBR)
4		PPR.IC	Instruction counter
4	a	ZERO	Zero indicator
4	b	NEG	Negative indicator
4	c	CARY	Carry indicator
4	d	OVFL	Overflow indicator
4	e	EOVF	Exponent overflow indicator
4	f	EUFL	Exponent underflow indicator
4	g	OFLM	Overflow mask indicator
4	h	TR0	Tally runout indicator
4	i	PAR	Parity error indicator
4	j	PARM	Parity mask indicator
4	k	BM	Not BAR Mode indicator
4	l	TRU	EIS truncation indicator
4	m	MIF	Mid-instruction interrupt
4	n	ABS	Absolute mode
5	x	TPR.CA	Current Effective Address
5	a	RF	First cycle of a repeat operation
5	b	RPT	Executing a repeat
5	c	RD	Executing a repeat double

<u>Word</u>	<u>key</u>	<u>L</u>	<u>Field Name</u>	<u>Meaning</u>
5	d		RL	Executing a repeat link
5	e		POT	Prepare operand tally This flag is up until the indirect word of an IT indirect cycle is successfully fetched.
5	f		PON	Prepare operand notally This flag is up until the indirect word of a "return" type instruction is successfully fetched. It indicates that there is no indirect chain even though an indirect fetch is being done.
5	g		XDE	Execute double from even IC
5	h		XDO	Execute double from odd IC
5	i		ITP	ITP cycle
5	j		RST	Restart this instruction
5	k		ITS	Executing ITS indirect cycle
5	l		FIF	Fault occurred during instruction fetch
5			CT HOLD	Contents of the "remember modifier" register
6				Word 6 is the contents of the "working instruction register" and reflects conditions at the exact point of address preparation when the fault/interrupt occurred. The ADDRESS and TAG fields are replaced with data from pointer registers, indirect pointers, and/or indirect words during each indirect cycle. Each instruction of the current pair is moved to this register before actual address preparation begins.
7				Word 7 is the contents of the "instruction holding register". It contains the odd word of the last instruction pair fetched from main store. Note that, primarily because of store overlap, this instruction is not necessarily paired with the instruction in Word 6.

DECIMAL UNIT DATA

Format - 288 bits, 8 machine words

Data as stored by Store Pointers and Lengths (SPL) instruction Word

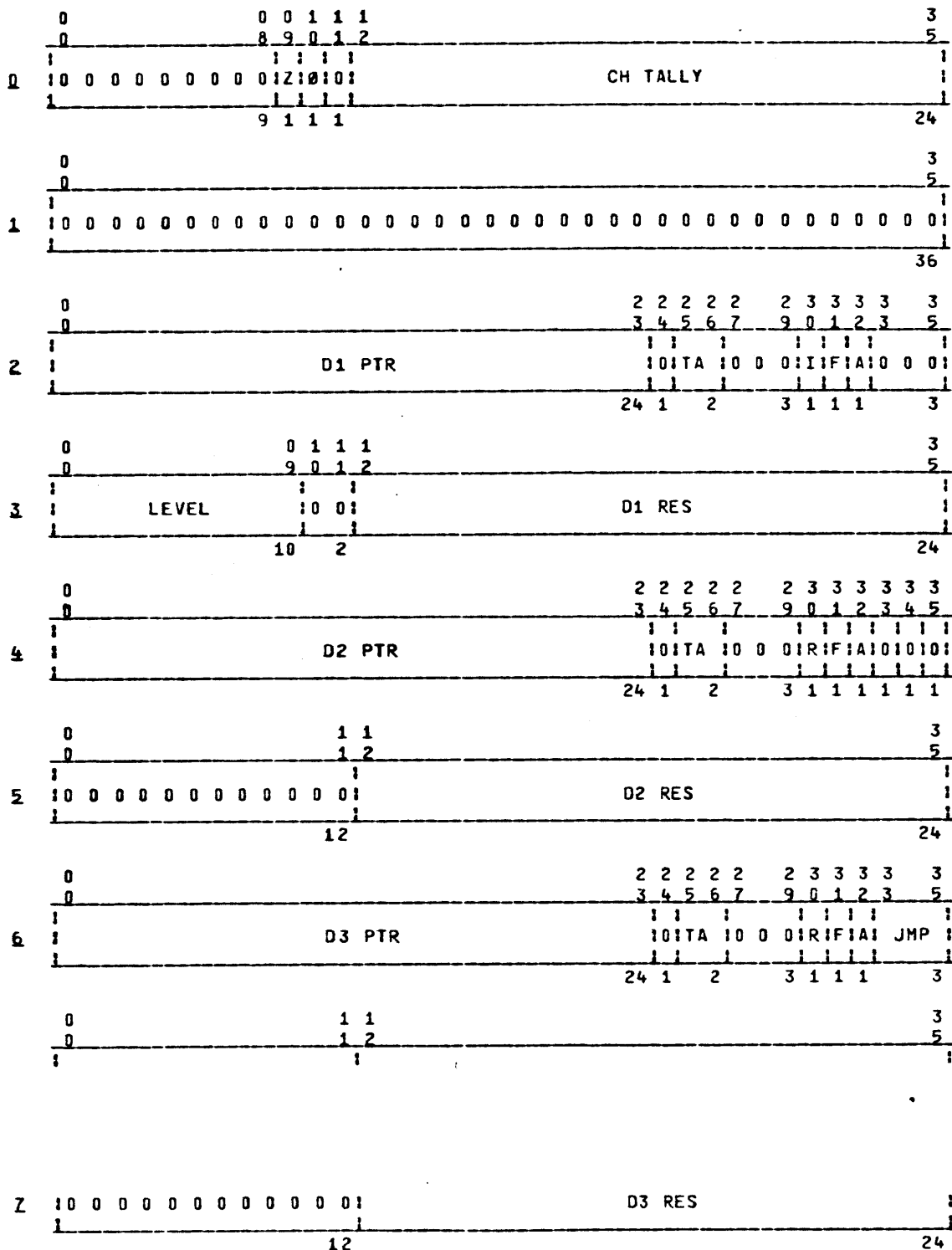


Figure 4-25 Decimal Unit Data Format

Description:

A logical collection of flags and registers from the Decimal Unit. Bits pictured as "0" are reserved and must have the value 0.

Function:

The Decimal Unit Data allows the Processor to restart an EIS instruction at the point of interruption when it is interrupted by an Access Violation Fault, a Directed Fault, or (for certain EIS instructions) a Program Interrupt. Directed Faults are intentional, and most Access Violation Faults and Program Interrupts are recoverable.

The data are restored with the Load Pointers and Lengths (LPL) instruction. Fields having an "x" in the column headed L are not restored. When starting execution of an EIS instruction, the decimal unit registers and flags are not initialized from the Operand Descriptors if the Mid-instruction Interrupt Fault (MIF) indicator is set 0N.

The meanings of the constituent flags and registers are:

<u>Word</u>	<u>L</u>	<u>Field Name</u>	<u>Meaning</u>
0	Z		All bit string instruction results are zero
0	0		Negative overpunch found in 6-4 expanded move
0	CHTALLY		The number of characters examined by the SCAN, TCT, or TCTR instruction (up to the interrupt or match)
2	D1 PTR		Address of last double word accessed by Operand Descriptor 1; bits 17-23 (bit address) valid only for initial access
2,4,6	TA		Alphanumeric type of Operand Descriptor 1,2,3
2	x I		Decimal Unit interrupted flag; a copy of the Mid-Instruction Interrupt Fault indicator
2,4,6	F		First time; data in Operand Descriptor 1,2,3 is valid
2,4,6	A		Operand Descriptor 1,2,3 is active
3	LEVEL		Difference in the count of characters loaded into the CPU and characters stored back to main store
3	D1 RES		Count of characters remaining in Operand Descriptor 1
4	D2 PTR		Address of last double word accessed by Operand Descriptor 2; bits 17-23 (bit address) valid only for initial access
4,6	x R		Last cycle performed must be repeated
5	D2 RES		Count of characters remaining in Operand Descriptor 2
6	D3 PTR		Address of the last double word accessed by Operand Descriptor 3; bits 17-23 (bit address) valid only for initial access

<u>Word</u>	<u>L</u>	<u>Field Name</u>	<u>Meaning</u>
6		JMP	Descriptor count; number of words to skip to find the next instruction following this multiword instruction
7		D3 RES	Count of characters remaining in Operand Descriptor 3

SECTION V

ADDRESSING -- SEGMENTATION AND PAGING

ADDRESSING MODES

The Multics Processor is able to access the main store in either of two modes; Absolute Mode or Append Mode.

The Processor prepares an Effective Address for each main store reference for instructions or operands. An Effective Address consists of a 12-bit segment number and an 18-bit offset within that segment. An offset is defined as the number of machine words from the segment base or origin to the referent. The Processor uses the Effective Address to generate a 24-bit final address. The final address is used either as a direct operand or as an address for a main store access. The various means of Effective Address formation are explained in Section VI, Effective Address Formation. The generation of the final address is different in the two Addressing Modes.

Absolute Mode

In Absolute Mode, the segment number is null, that is, undefined, and the segment base is the origin of main store. The final address is generated by high-order zero filling the offset with six binary 0's. Absolute Mode addressing is limited to the first 262,144 words of main store.

In Absolute Mode, all instruction fetches are made from Absolute addresses. Instruction operands may be located anywhere in main store and may be accessed by specifying ITS Address Modification for the instruction or by loading a Pointer Register with an appropriate value and specifying ITP Address Modification or using bit 29 of the instruction word. The use of ITS or ITP Address Modification in an Indirect Word will have the same effect.

WARNING: The use any of the above constructs in Absolute Mode places the Processor in Append Mode for one or more Address Preparation cycles. All necessary registers must be properly loaded and all Fault

conditions must be considered (See Append Mode below).

If a transfer of control is made with any of the above constructs, the processor remains in Append Mode after the transfer and subsequent instruction fetches are made in Append Mode.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

Although no segment is defined for Absolute Mode, it may be helpful to understanding to visualize a virtual, unpagged segment overlaying the first 262,144 words of main store.

Append Mode

In Append mode, the appending mechanism is employed for all main store references. The appending mechanism is described in "Segmentation" and "Paging" following in this section.

SEGMENTATION

A Multics segment is defined as an array of machine words of arbitrary (but limited) size containing arbitrary data. A segment is identified within the Processor by a segment number (segno), unique to the segment for the process, that is assigned by the operating system when the segment is first referenced by the process.

To simplify this discussion, the operation of the hardware ring mechanism is not described although it is an integral part of Address Preparation. See Section VIII, Hardware Ring Implementation, for a discussion of the ring mechanism hardware.

An Effective Address in the Processor consists of a pair of integers (segno, offset). The range of segno is $(0, 2^{*12}-1)$, the range of offset is $(0, 2^{*18}-1)$. The description of the segment identified by segno value n is kept in the nth word-pair (offset = $2 * n$) in a table known as the descriptor segment (dseg). The descriptor segment always has segno value 0 and contains descriptions of all segments accessible by the process including its own description in Y-pair 0. The location of the descriptor segment for a running process is held by the Processor in the Descriptor Segment Base Register (DSBR). (See Section IV, Program Accessible Registers) Each word-pair of a descriptor segment is known as a Segment Descriptor Word (SDW) and is 72 bits long. (See Figure 5-5, Segment Descriptor Word (SDW) Format, later in this section.)

A bit in the SDW for a segment (SDW.U) specifies whether the segment is paged or unpagged. The following is a simplified description of the appending process for unpagged segments. (Refer to Figures 4-14 and 5-5)

1. If $2 * \text{segno} \geq 16 * (\text{DSBR.BND} + 1)$, then generate an Access Violation, Out of Segment Bounds Fault.
2. Fetch the SDW from $\text{DSBR.ADDR} + 2 * \text{segno}$.
3. If $\text{SDW.F} = "0"$, then generate Directed Fault n where n is given in SDW.FC . The value of n used here is the value assigned to define a missing segment fault or segment fault.
4. If $\text{offset} \geq 16 * (\text{SDW.BOUND} + 1)$, then generate an Access Violation, Out of Segment Bounds Fault.
5. If the access bits (SDW.R , SDW.E , etc.) of the segment are incompatible with the reference, generate the appropriate Access Violation Fault.

- o Avoid using common keywords as names. When names such as

goto declare dcl if then

and so on are used as names, a superficial but irritating confusion is introduced. On the other hand, do use uncommon keywords as names where that is convenient. There is certainly no harm in using 'dft' to name a variable for the "debit final total" (or something of the sort) even though 'dft' is a keyword.

- o Where possible, avoid using troublesome letters in identifiers. For example, the digits zero and one are troublesome because some output devices do not clearly distinguish between zero and the letter '0', or between one and the letter '1'.

Literal Constants

There is a literal constant lexeme for each type of arithmetic and string value. The full syntax and interpretation of these lexemes are given later, in the section on "Expressions". The following is a representative set of examples of arithmetic literal constants:

Arithmetic_Constant	Data_Type
304	fixed dec(3)
3.04	fixed dec(3,2)
3.04e-5	float dec(3)
3.04e-5i	complex float dec(3)
0110001b	fixed(7)
011.0001b	fixed(7,4)
011.0001e-2b	float(7)
011.0001e-2bi	complex float(7)

Observe that an arithmetic constant does not begin with a sign. When a negative constant is required, it is written as two lexemes, a sign followed by an arithmetic constant.

The following is a representative set of examples of string literal constants:

String_Constant	Data_Type	Remark
"abcd"	char(4)	
(3)"abcd"	char(12)	means "abcdabcdabcd"
""	char(0)	means the null string
"" "Hello," "" he said. ""	char(17)	"" counts as " in value
"11101"b	bit(5)	
(4)"01"b	bit(8)	means "C1010101"b
""b	bit(0)	means the null string

Any ASCII character can be used in a 'character' string constant, including such non-printing characters as tab, newline, and so on. A string constant is a single lexeme, and is not considered to contain smaller lexemes.

Punctuators

There are six punctuator_lexemes; each is given, together with its purpose, in the following table:

Punctuator	Purpose
. (period)	indicates the decimal or binary point; also, separates names in a qualified reference
, (comma)	separates items in a list of arguments, parameters, subscripts, declarations, options, and so on
: (colon)	terminates a condition prefix or a label prefix; also, separates the bounds of an array
; (semicolon)	terminates a statement
((left parenthesis)	indicates the beginning of a list, an expression, an iteration factor, and so on
) (right parenthesis)	indicates the end of a list, an expression, an iteration factor, and so on

These lexemes are used in most of the features of PL/I.

Operators

There are five kinds of operator_lexemes; they are defined as follows:

Classification	Operators
arithmetic	+ - * / **
relational	= ^= < ^< > ^> <= >=
logical	^ .
string	
qualifier	->

Most of the operators are defined in the section on "Operators". The only exception is the qualifier operator, which is defined in the section on "Expressions".

ADDRESS APPENDING

At the completion of the formation of the Effective Address (See Section VI, Effective Address Formation) an Effective Segment Number (segno) is in the Segment Number Register of the Temporary Pointer Register (TPR.SNR) and a Computed Address (offset) is in the Computed Address register of the Temporary Pointer Register (TPR.CA) (See Section IV, Program Accessible Registers, for a discussion of the Temporary Pointer Register).

Address Appending Sequences

Once segno and offset are formed in TPR.SNR and TPR.CA, respectively, the process of generating the final address can involve a number of different and distinct Appending Unit cycles.

The operation of the Appending Unit is shown the flowchart in Figure 5-4. This flowchart assumes that Directed Faults Store Faults, or Parity Faults do not occur.

A segment boundary check is made in every cycle except PSDW. If a boundary violation is detected, an Access Violation, Out of Segment Bounds Fault will be generated and the execution of the instruction aborted. The occurrence of any Fault will abort the sequence at the point of occurrence. The operating system will save store the Control Unit Data for possible retry and will attempt to resolve the Fault condition.

The value of the Associative Memories may be seen in the flowchart by observing the number of cycles bypassed if an SDW or PTW is found in the Associative Memory.

There are nine different Appending Unit cycles that involve accesses to main store. Two of these (FANP, FAP) generate the final address and initiate a main store access for the operand or instruction pair; five (NSDW, PSDW, PTW, PTW2 and DSPTW) generate a main store access to fetch an SDW or PTW; and two (MDSPTW and MPTW) generate a main store access to update page status bits (PTW.U and PTW.M) in a PTW. The cycles are defined in Table 5-1 below.

Table 5-1 Appending Unit Cycle Definitions

<u>Cycle Name</u>	<u>Function</u>
FANP	<u>F</u> inal <u>A</u> ddress <u>N</u> on <u>P</u> aged Generates the final address and initiates an main store access to an unpagged segment for operands or instructions.
FAP	<u>F</u> inal <u>A</u> ddress <u>P</u> aged Generates the final address and initiates a main store access to a pagged segment for operands or instructions.
NSDW	<u>N</u> onpagged <u>S</u> DW <u>F</u> etch Fetches an SDW from an unpagged <u>d</u> seq.
PSDW	<u>P</u> aged <u>S</u> DW <u>F</u> etch Fetches an SDW from a pagged descriptor segment.
PTW	<u>P</u> IW <u>F</u> etch Fetches a PTW from a page table other than a <u>d</u> seq page table.
PTW2	<u>S</u> econd <u>P</u> IW <u>F</u> etch (Same as PTW above) Fetches the next PTW from a page table other than a <u>d</u> seq page table during hardware prepaging for certain uninterruptable EIS instructions. This cycle does <u>not</u> load the next PTW into the Appending Unit. It merely assures that the PTW is not faulted (PTW.F = "1") and that the target page will be in main store when and if needed by the instruction.
DSPTW	<u>D</u> escriptor <u>S</u> egment <u>P</u> IW <u>F</u> etch Fetches a PTW from a <u>d</u> seq page table.
MDSPTW	<u>M</u> odify <u>D</u> SPIW Sets the page accessed bit (PTW.U) in the PTW for a page in a <u>d</u> seq page table. This cycle always immediately follows a DSPTW cycle.
MPTW	<u>M</u> odify <u>P</u> IW Sets the page modified bit (PTW.M) in the PTW for a page in other than a <u>d</u> seq page table.

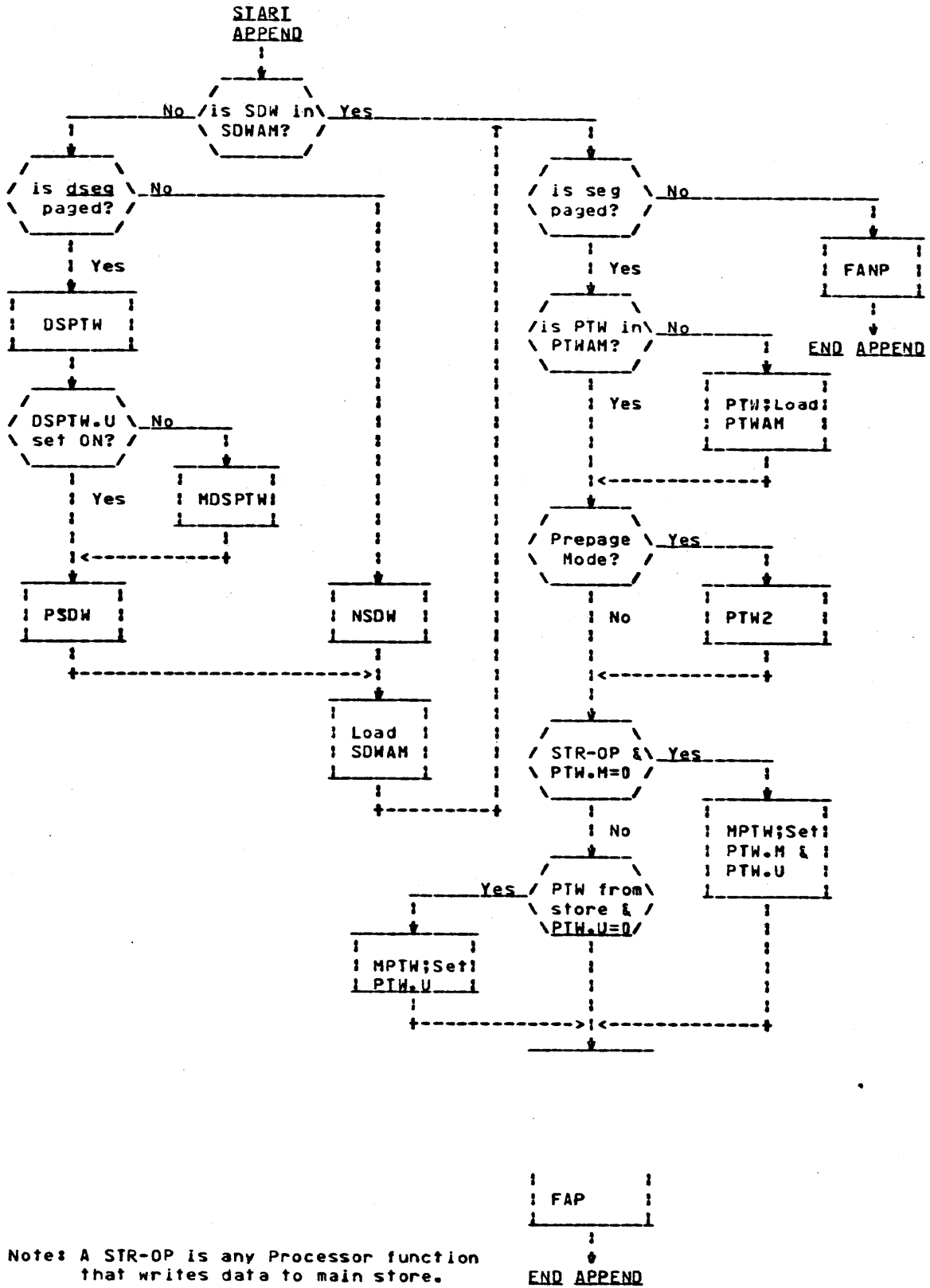


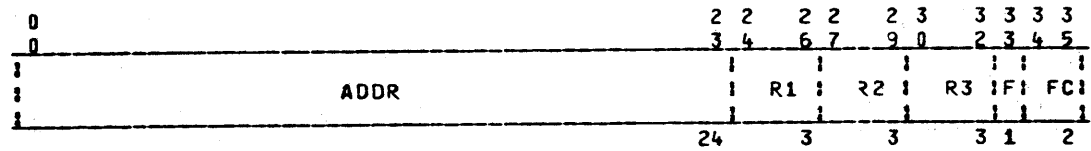
Figure 5-4 Appending Unit Operation Flowchart

APPENDING UNIT DATA WORD FORMATS

Segment Descriptor Word (SDW) Format

The Segment Descriptor Word (SDW) pair contains information necessary to control the access to a segment by a process. The SDW for a segment is constructed from data in the directory entry for the segment and in the System Segment Table (SST) when the segment is initiated by the process. The SDW for segment Δ (unique within the process) is placed at offset 2Δ in the Descriptor Segment ($dseg$) of the process.

Even Word



Odd Word

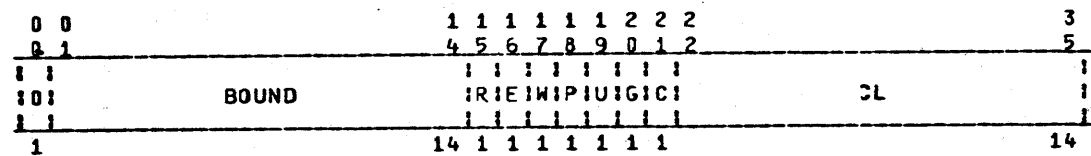


Figure 5-5 Segment Descriptor Word (SDW) Format

Field Name Description

- ADDR 24 bit base address of segment (U=1) or segment page table (U=0).
- R1 highest effective read/write ring.
- R2 highest effective read/execute ring.
- R3 highest effective call ring.
- F directed fault indicator.
1 = the necessary unpagged segment or segment page table is in memory.

- FC 0 = execute the directed fault specified in FC.
the number of the directed fault (DF0-DF3) to be executed if F=0.
- BOUND largest 16-word block number that may be accessed without causing an Access Violation, Out of Segment Bounds Fault.
- R read permission bit.

Field Name	Description
E	eXECute permission bit. (XEC & XED excluded)
W	write permission bit.
P	privileged mode bit. 0 = privileged instructions cannot be executed. 1 = privileged instructions may be executed if in ring 0.
U	paged/unpaged bit. 0 = segment is paged and ADDR is the address of the page table. 1 = segment is unpaged and ADDR is the base address of the segment.
G	gate indicator bit. 0 = any call from an external segment must be to an offset less than the value of CL. 1 = any legal segment offset may be called.
C	cache control bit. 0 = words (operands or instructions) from this segment may not be placed in the cache. 1 = words from this segment may be placed in the cache.
CL	call limiter. Any external call to this segment must be to an offset less than CL if G=0.

Page Table Word (PTW) Format

The Page Table Word (PTW) contains location and status information for a page of a paged segment. The PTWs for a paged segment are copied from the directory entry file map for the segment into the Page Table Word Array (PTWA) of a free area in the Active Segment Table (AST) area of the SST when the segment is first initiated by a process. Subsequent initiations by other processes reference the existing PTWA.

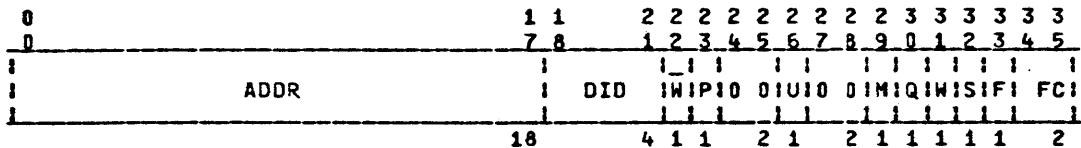


Figure 5-6 Page Table Word (PTW) Format

Field Name	Description
ADDR	18 bit modulo 64 page address if page is in store, QC 18 bit record number of page if page is not in store.
	The hardware ignores low order bits of the in-store page address according to page size based on the following ...

Field Name Description

Page Size in words	ADDR Bits ignored
64	none
128	17
256	16-17
512	15-17
1024	14-17
2048	13-17
4096	12-17

- DID device id for device containing the page.
- \bar{W} 1 = page has not yet been written out.
- P temporary bit used in post_processing.
- U 1 = page has been used (touched).
- M 1 = page has been modified.
- Q 1 = page has been used during the quantum.
- W 1 = page is wired.
- S 1 = page is out of service (i/o in progress).
- F 1 = page is in store.
0 = page not in store. Execute directed fault FC.
- FC directed fault number for page fault.

SECTION VI

EFFECTIVE ADDRESS FORMATION

DEFINITION OF EFFECTIVE ADDRESS

The Effective Address in the Multics Processor is the user's specification of the location of a data item in the Multics Virtual Memory. Each reference to the Virtual Memory for operands, indirect words, indirect pointers, Operand Descriptors, or instructions must provide an Effective Address. The hardware and the operating system translate the Effective Address into the true location of the data item and assure that the data item is in main store for the reference.

The Effective Address consists of two parts, a segment number and an offset. The value of each part is the result of the evaluation of a hardware algorithm (expression) of one or more terms. The selection of the algorithm is made by the use of control bits in the Instruction Word; namely, bit 29 for segment number modification and the Address Modification (or TAG) field for offset modification. If the TAG field of the Instruction Word specifies certain "indirect" modifications, the TAG field of the Indirect Word is also treated as an Address Modifier, thus establishing a continuing "indirect chain". Bit 29 of an Indirect Word has no meaning in the context of Address Modification.

The results of evaluation of the Address Modification algorithm are stored in temporary registers used as working registers by the Processor. The segment number is stored in the Temporary Segment Register (TPR.TSR). The offset is stored in the Computed Address Register (TPR.CA). When each Effective Address computation has been completed, the C(TPR.TSR) and the C(TPR.CA) are presented to the Appending Unit for translation to a 24-bit final Address (See Section V, Addressing -- Segmentation and Paging).

TYPES OF EFFECTIVE ADDRESS FORMATION

There are two types of Effective Address formation. The first type does not make explicit use of segment numbers. The algorithm selected produces a value for C(TPR.CA) only. The segment number in C(TPR.TSR) does not change and is the

segment number used to fetch the instruction. In this case, all references are said to be "local" to the procedure segment as held in C(PPR.PSR).

The second type makes use of a segment number stored either in an Indirect Word-pair in main store or in a Pointer Register (PRn). The algorithm selected produces values for both C(TPR.TSR) and C(TPR.CA). The segment number in C(TPR.TSR) may change and, if it changes, references are said to be "external" to the procedure segment as held in C(PPR.PSR).

The two types of Effective Address formation can be intermixed. In cases where Effective Address calculations are chained together through Pointer Registers or Indirect Words, each Effective Address is translated to a 24-bit final address to fetch the next item in the chain.

EFFECTIVE ADDRESS FORMATION DESCRIPTION

This description of Effective Address formation is divided into two parts corresponding to the two types. The first part describes the type that involves only the offset value C(TPR.CA). The segment number C(TPR.TSR) is assumed constant and equal to C(PPR.PSR).

The second part describes the type that involves both the segment number C(TPR.TSR) and the offset C(TPR.CA).

EFFECTIVE ADDRESS FORMATION INVOLVING OFFSET ONLY

The Address Modifications described here produce values for C(TPR.CA) only. The segment number C(TPR.TSR) is assumed constant and equal to C(PPR.PSR).

The Address Modifier (TAG) Field

Bits 30-35 of an Instruction Word or Indirect Word constitute the Address Modifier or TAG field. The format of the TAG field is:

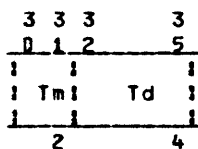


Figure 6-1 Address Modifier (TAG) Field Format

Field Name	Function
Tm	The "modifier" field; specifies one of four general types of offset modification.
Td	The "designator" field; specifies a register number or an Indirect Then Tally variation.

General Types of Offset Modification

There are four general types of offset modification: Register, Register Then Indirect, Indirect Then Register, and Indirect Then Tally. The general types are described in Table 6-1 below.

Each Effective Address formation for an operand begins with a preliminary step of loading TPR.CA with the ADDRESS field of the Instruction Word. This preliminary step takes place during instruction decode. The value loaded into TPR.CA is symbolized by "y" in the descriptions following.

Table 6-1 General Offset Modification Types

<u>Tm Value</u>	<u>Modifier Type</u>	<u>Description</u>
0	Register (R)	The contents of the designated register, Td, are added to the current Computed Address to form the modified Computed Address. Addition is two's complement, modulo 2^{*18} and overflow is not possible.
1	Register Then Indirect (RI)	The contents of the designated register, Td, are added to the current Computed Address to form the modified Computed Address as for Register modification. The word at C(TPR.CA) is then fetched and interpreted as an Indirect Word. The TAG field of the Indirect Word specifies the next step in Effective Address formation. The use of du or dl as the designator in this modification type will cause an Illegal Procedure, Illegal Modifier Fault.
2	Indirect Then Tally (IT)	The Indirect Word at C(TPR.CA) is fetched and the modification performed according to the variation specified in Td and the contents of the Indirect Word. This modification type allows automatic incrementing and decrementing of addresses and tally counting.
3	Indirect Then Register (IR)	The register designator, Td, is safe-stored in a special holding register (CT-HOLD). The word at the current C(TPR.CA) is fetched and interpreted as an Indirect Word. The TAG field of the Indirect Word specifies the next step in Effective Address formation as follows:

If Indirect

TAG is: then:

R or IT Perform Register modification using Td from CT-HOLD.

RI Perform the Register Then Indirect modification immediately and fetch the next Indirect Word from the result of that modification.

IR Replace the safe-stored Td value in CT-HOLD with the Td value of the Indirect Word TAG field and fetch the next Indirect Word from the ADDRESS given in the Indirect Word.

Effective Address Formation Flowcharts

The algorithmic flowcharts depicting the Effective Address formation process are scattered throughout this section and are linked together with "Go to" labels. The flowchart starts with Figure 6-2 below.

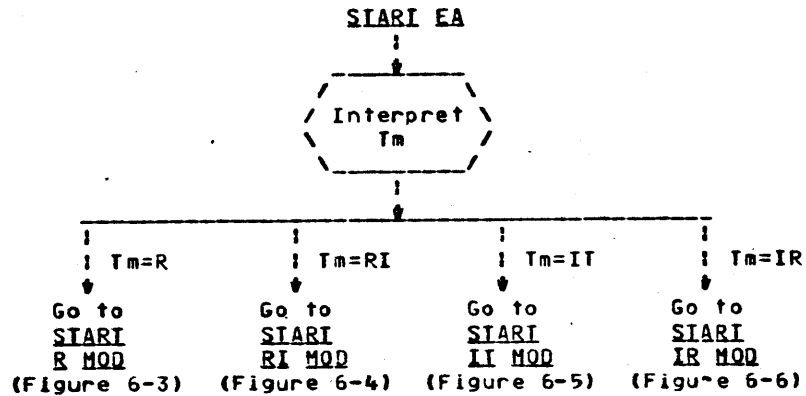


Figure 6-2 Common Effective Address Formation Flowchart

Register (R) Modification

In Register modification ($T_m = 0$) the value of T_d designates a register whose contents are to be added to $C(TPR.CA)$ to form a modified $C(TPR.CA)$. This modified $C(TPR.CA)$ becomes the Effective Address of the operand. See Table 6-2 and Figure 6-3 below for details.

EXAMPLES:

	<u>Label</u>	<u>Instruction</u>	<u>Effective Address</u>
1.	a	lda y	y
2.	a	sta y,n	y
3.	a	ldaq y,au	y + C(A)0,17
4.	a	tra 3,ic	a + 3
5.	a	ldq y,du	y; operand has the form zero y,0
6.	a	lxl4 y,d1	y; operand has the form zero 0,y
7.	a	mpy y,1	y + C(X1)
8.	a	stx4 y,7	y + C(X7)

Table 6-2 Register Modification Decode

(NOTE: All examples start with the preliminary step, $y \rightarrow C(TPR.CA)$)

Td Value	Register Selected	Coding Mnemonic	Effective Address
0	none	n or null	y
1	A0,17	au	$y + C(A)0,17$
2	Q0,17	qu	$y + C(Q)0,17$
3	none	du	y; y becomes the upper 18 bits of the 36-bit zero filled operand
4	PPR.IC	ic	$y + C(PPR.IC)$
5	A18,35	al	$y + C(A)18,35$
6	Q18,35	ql	$y + C(Q)18,35$
7	none	dl	y; y becomes the lower 18 bits of the 36-bit zero filled operand
10	X0	0 or x0	$y + C(X0)$
11	X1	1 or x1	$y + C(X1)$
12	X2	2 or x2	$y + C(X2)$
13	X3	3 or x3	$y + C(X3)$
14	X4	4 or x4	$y + C(X4)$
15	X5	5 or x5	$y + C(X5)$
16	X6	6 or x6	$y + C(X6)$
17	X7	7 or x7	$y + C(X7)$

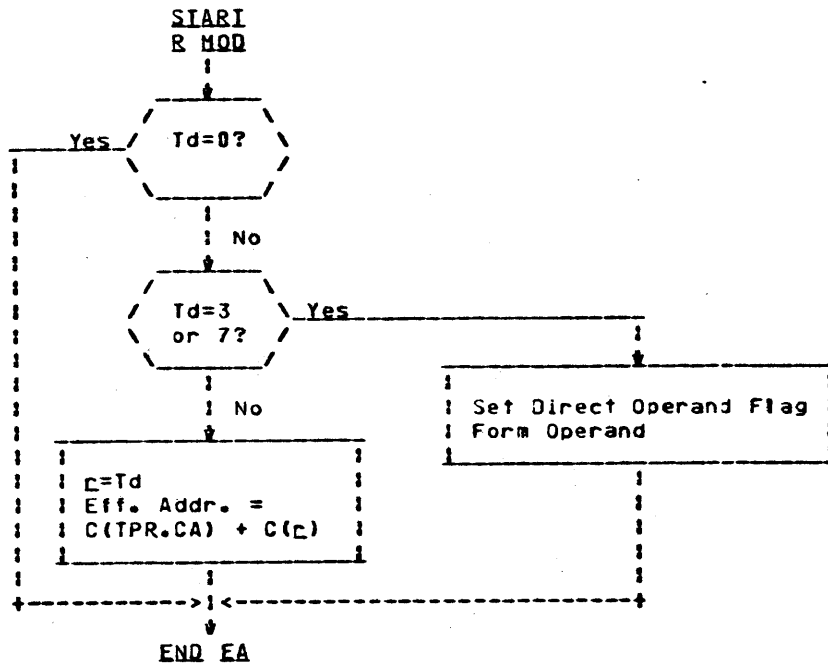


Figure 6-3 Register Modification Flowchart

Register Then Indirect (RI) Modification

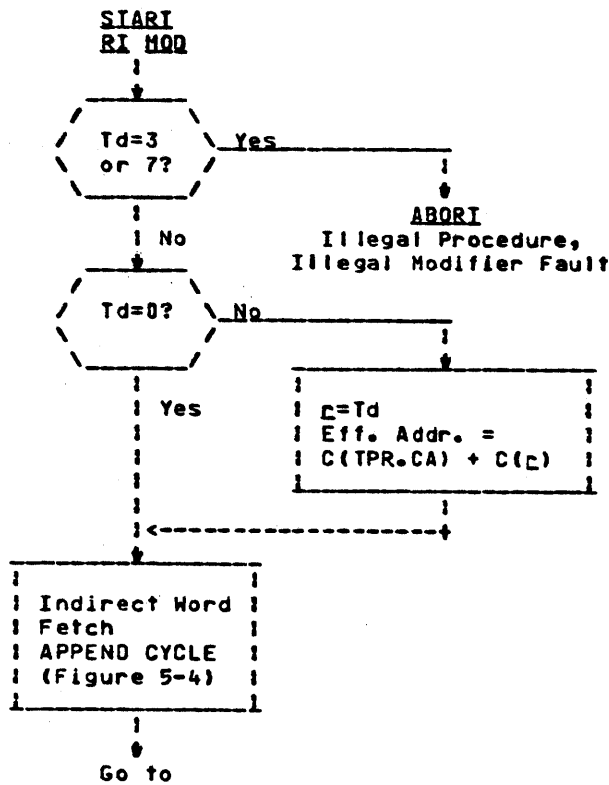
In Register Then Indirect modification ($T_m = 1$) the value of T_d designates a register whose contents are to be added to $C(TPR.CA)$ to form a modified $C(TPR.CA)$. This modified $C(TPR.CA)$ is used as an Effective Address to fetch an Indirect Word. The ADDRESS field of the Indirect Word is loaded into $TPR.CA$ and the TAG field of the Indirect Word is interpreted in the next step of an indirect chain. The TALLY field of the Indirect Word is ignored.

The indirect chain continues until an Indirect Word TAG field specifies a modification without indirection, namely, a Register modification.

The coding mnemonic for Register Then Indirect modification is \underline{r}^* where \underline{r} is any of the coding mnemonics for Register modification as given in Table 6-2 above except du and dl . The du and dl register codes are illegal and will cause an Illegal Procedure, Illegal Modifier fault. See flowchart in Figure 6-4 below.

EXAMPLES:

	Label	Instruction	Effective Address
1.	a b	lda b,* arg y	y
2.	a b+C(X1)	ldq b,1* arg y,au	y + C(A)0,17
3.	a a+4 c	fra 4,ic* arg c,* arg y	y
4.	a b+C(X0) c+C(X1)	lxl4 b,0* arg c,1* arg y,d1	y; operand has the form zero 0,y



START EA
(Figure 6-2)

Figure 6-4 Register Then Indirect Modification Flowchart

Indirect Then Register (IR) Modification

In Indirect Then Register modification ($Tm = 3$) the value of Td designates a register whose contents are to be added to $C(TPR.CA)$ to form the final modified $C(TPR.CA)$ during the last step in the indirect chain. The value of Td is safe-stored in a special holding register, CT-HOLD. The initial $C(TPR.CA)$ is used as an Effective Address to fetch an Indirect Word. The ADDRESS field of the Indirect Word is loaded into $TPR.CA$ and the TAG field of the Indirect Word is interpreted in the next step of an indirect chain. The TALLY field of the Indirect Word is ignored.

If the Indirect Word TAG field specifies a Register Then Indirect modification, that modification is performed and the indirect chain continues.

If the Indirect Word TAG field specifies Indirect Then Register modification, the Td value from that TAG field replaces the safe-stored Td value in CT-HOLD and the indirect chain continues.

If the Indirect Word TAG specifies Register or Indirect Then Tally modification, that modification is replaced with a Register modification using the Td value safe-stored in CT-HOLD and the indirect chain ends.

The coding mnemonic for Indirect Then Register modification is $*r$ where r is any of the coding mnemonics for Register modification as given in Table 6-2 above except null.

EXAMPLES:

	<u>Label</u>	<u>Instruction</u>	<u>Effective Address</u>
1.	a b	lda b,*n arg y,2	(CT-HOLD = n) y
2.	a b	lx12 b,*dl sta y,au	(CT-HOLD = dl) y; operand has the form zero 0,y
3.	a b c d	lda b,*1 arg c,n* arg d,*4 arg y,d1	(CT-HOLD = x1) (CT-HOLD = x4) y + C(X4)
4.	a b+C(X1) c	ldx0 b,1* arg c,*ic arg 5,d1	(CT-HOLD = ic) a + 5

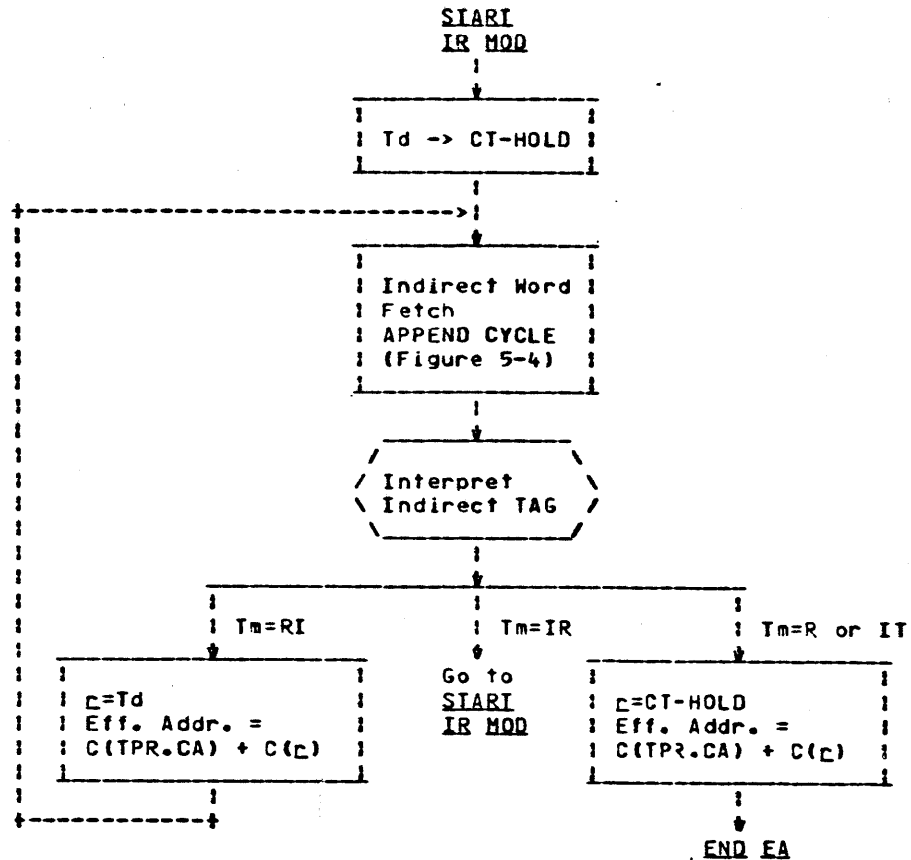


Figure 6-5 Indirect Then Register Modification Flowchart

Indirect Then Tally (IT) Modification

In Indirect Then Tally modification ($T_m = 2$) the value of T_d specifies a variation. The initial $C(TPR.CA)$ is used as an Effective Address to fetch an Indirect Word. The Indirect Word is interpreted and possibly altered as the modification is performed.

The TALLY field of the Indirect Word is used to count references made to the Indirect Word. It has a maximum range of 4096. If the TALLY field has the value 0 after a reference to the Indirect Word, the Tally Runout indicator will be set ON, otherwise the Tally Runout indicator will be set OFF. The value of

the TALLY field and the state of the Tally Runout indicator have no effect on Effective Address formation.

WARNING: If there is more than one Indirect Word in an indirect chain that is referenced by a tally counting modification, only the state of the TALLY field of the last such word will be reflected in the Tally Runout indicator.

The variations of the Indirect Then Tally modification are given in Table 6-3 below and explained in detail in the paragraphs following. See flowchart in figure 6-6. Those entries given as "Undefined" cause an Illegal Procedure, Illegal Modifier Fault. (See "Effective Address Formation Involving Both Segment Number and Offset" later in this section for certain special cases.)

Table 6-3 Variations of Indirect Then Tally Modification

<u>Td</u> <u>Value</u>	<u>Coding</u> <u>Mnemonic</u>	<u>Variation Name</u>
0	f1	Fault Tag 1
1		Undefined
2		Undefined
3		Undefined
4	sd	Subtract Delta
5	scr	Sequence Character Reverse
6	f2	Fault Tag 2
7	f3	Fault Tag 3
10	ci	Character Indirect
11	i	Indirect
12	sc	Sequence Character
13	ad	Add Delta
14	di	Decrement Address, Increment Tally
15	dic	Decrement Address, Increment Tally, and Continue
16	id	Increment Address, Decrement Tally
17	idc	Increment Address, Decrement Tally, and Continue

Fault Tag 1 (Td = 0)

Effective Address formation is terminated immediately and a Fault Tag 1 Fault is generated. A Fault Tag 1 Fault executes the Fault Trap pair at C + 6 where the value of C is obtained from the FAULT BASE

switches on the Processor Configuration panel.

This variation may be used in Indirect Word or program control transfer vectors or tree structures to signal invalid entries or entries that require special handling. C(TPR.CA) at the time of the fault contains the Effective Address of the word containing the Fault Tag 1 modification. Thus, the ADDRESS and TALLY fields of that word may contain information relative to recovery from the fault.

Subtract Delta (Td = 4)

The TAG field of the Indirect Word is interpreted as a 6-bit, unsigned, positive address increment value, delta. For each reference to the Indirect Word, the ADDRESS field is reduced by delta and the TALLY field is increased by 1 before the Effective Address is formed. ADDRESS arithmetic is modulo $2^{*}18$. TALLY arithmetic is modulo 4096. If the TALLY field overflows to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The Effective Address is the value of the modified ADDRESS field.

EXAMPLE:

Label	Instruction	Reference Count	Effective Address	Tally Value
a	lda b,ad	1	c-d	t+1
b	vfd 18/c,12/t,6/d	2	c-2d	t+2
		3	c-3d	t+3
		...		
		n	c-nd	t+n

Sequence Character Reverse (Td = 5)

Bit 30 of the TAG field of the Indirect Word is interpreted as a character size flag, fb, with the value 0 indicating 6-bit characters and the value 1 indicating 9-bit characters. Bits 33-35 of the TAG field are interpreted as a 3-bit character position counter, cf. Bits 31-32 of the TAG field must be zero.

For each reference to the Indirect Word, the character counter, cf, is reduced by 1 and the TALLY field is increased by 1 before the Effective Address is formed. Character count arithmetic is modulo 6 for 6-bit characters and modulo 4 for 9-bit characters. If the character count, cf, underflows to -1, it is reset to 5 for 6-bit characters or to 3 for 9-bit characters and ADDRESS is reduced by 1. ADDRESS arithmetic is modulo $2^{*}18$. TALLY arithmetic is modulo 4096. If the TALLY field overflows to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The Effective Address is the modified value of the ADDRESS field.

A 36-bit operand is formed by high-order zero filling the value of character cf of ADDRESS with an appropriate number of bits.

EXAMPLES:

Label	Instruction	Reference Count	cf	Effective Address	Tally Value	Operand
a	lda b,scr	1	2	c+1	t+1	00...0"I"
b	vfd 18/c+1,12/t,1/0,5/2	2	1	c+1	t+2	00...0"H"
c	bci "ABCDEFGHijkl"	3	0	c+1	t+3	00...0"G"
		4	5	c	t+4	00...0"F"
		5	4	c	t+5	00...0"E"
		...				
a	lda b,scr	1	2	c+1	t+1	00...0"g"
b	vfd 18/c+1,12/t,1/1,5/2	2	1	c+1	t+2	00...0"f"
c	aci "abcdefgh"	3	0	c+1	t+3	00...0"e"
		4	3	c	t+4	00...0"d"
		5	2	c	t+5	00...0"c"
		...				

Fault Tag 2 (Td = 6)

The action for this variation is identical to that for Fault Tag 1 except that the Trap Pair at C + 60 (octal) is executed.

WARNING: Fault Tag 2 is reserved to the Multics operating system for use in the Dynamic Linking feature. Its attempted use for other purposes could cause serious system inconsistencies and/or system crashes.

Fault Tag 3 (Td = 7)

The action for this variation is identical to that for Fault Tag 1 except that the Trap Pair at C + 62 (octal) is executed.

Character Indirect (Td = 10)

Bit 30 of the TAG field of the Indirect Word is interpreted as a character size flag, *ib*, with the value 0 indicating 6-bit characters and the value 1 indicating 9-bit characters. Bits 33-35 of the TAG field are interpreted as a 3-bit character position value, *ci*. Bits 31-32 of the TAG field must be zero.

If the character position value is greater than 5 for 6-bit characters or greater than 3 for 9-bit characters, an Illegal Procedure, Illegal Modifier Fault will occur. The TALLY field is ignored. The Effective Address is the value of the ADDRESS field.

A 36-bit operand is formed by high-order zero filling the value of character *ci* of ADDRESS with an appropriate number of bits.

EXAMPLES:

<u>Label</u>	<u>Instruction</u>	<u>Operand</u>
a	lda b,ci	
b	vfd 18/c+1,12/0,1/0,5/2	00...0"I"
c	bci "ABCDEFGHijkl"	
d	lda d,ci	
	vfd 18/c,12/0,1/0,5/1	00...0"B"
e	lda e,ci	
f	vfd 18/f,12/0,1/1,5/3	00...0"d"
	aci "abcdefgh"	
g	lda g,ci	
	vfd 18/f+1,12/0,1/1,5/0	00...0"e"

Indirect (Td = 11)

The Effective Address is the value of the ADDRESS field. The TALLY

and TAG fields are ignored.

Sequence Character (Td = 12)

Bit 30 of the TAG field of the Indirect Word is interpreted as a character size flag, *ib*, with the value 0 indicating 6-bit characters and the value 1 indicating 9-bit characters. Bits 33-35 of the TAG field are interpreted as a 3-bit character position counter, *ci*. Bits 31-32 of the TAG field must be zero.

For each reference to the Indirect Word, the character counter, *ci*, is increased by 1 and the TALLY field is reduced by 1 after the Effective Address is formed. Character count arithmetic is modulo 6 for 6-bit characters and modulo 4 for 9-bit characters. If the character count, *ci*, overflows to 6 for 6-bit characters or to 4 for 9-bit characters, it is reset to 0 and ADDRESS is increased by 1. ADDRESS arithmetic is modulo $2^{*}18$. TALLY arithmetic is modulo 4096. If the TALLY field is reduced to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The Effective Address is the original unmodified value of the ADDRESS field.

A 36-bit operand is formed by high-order zero filling the value of character *ci* of ADDRESS with an appropriate number of bits.

EXAMPLES:

Label	Instruction	Reference Count	<i>ci</i>	Effective Address	Tally Value	Operand
a	lda b,sc	1	4	c	t-1	00...0"E"
b	vfd 18/c,12/t,1/0,5/4	2	5	c	t-2	00...0"F"
c	bci "ABCDEFGHJKLM"	3	0	c+1	t-3	00...0"G"
		4	1	c+1	t-4	00...0"H"
		5	2	c+1	t-5	00...0"I"
		...				
a	lda b,sc	1	2	c	t-1	00...0"c"
b	vfd 18/c,12/t,1/1,5/2	2	3	c	t-2	00...0"d"
c	aci "abcdefgh"	3	0	c+1	t-3	00...0"e"
		4	1	c+1	t-4	00...0"f"
		5	2	g+1	t-5	00...0"g"
		...				

Add Delta (Td = 13)

The TAG field of the Indirect Word is interpreted as a 6-bit, unsigned, positive address increment value, *delta*. For each reference to the Indirect Word, the ADDRESS field is increased by *delta* and the TALLY field is reduced by 1 after the Effective Address is formed. ADDRESS arithmetic is modulo $2^{*}18$. TALLY arithmetic is modulo 4096. If the TALLY field is reduced to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The Effective Address is the value of the original unmodified ADDRESS field.

EXAMPLE:

Label	Instruction	Reference Count	Effective Address	Tally Value
a	lda b,ad	1	c	t-1
b	vfd 18/c,1/t,6/d	2	c-d	t-2
		3	c-2d	t-3
		...		
		n	c-(n-1)d	t-n

Decrement Address, Increment Tally (Td = 14)

For each reference to the Indirect Word, the ADDRESS field is reduced by 1 and the TALLY field is increased by 1 before the Effective Address is formed. ADDRESS arithmetic is modulo $2^{*}18$. TALLY arithmetic is modulo 4096. If the TALLY field overflows to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The TAG

field of the Indirect Word is ignored. The Effective Address is the value of the modified ADDRESS field.

EXAMPLES

<u>Label</u>	<u>Instruction</u>	<u>Reference Count</u>	<u>Effective Address</u>	<u>Tally Value</u>
a	lda b,di	1	c-1	t+1
b	vfd 18/c,12/t	2	c-2	t+2
		3	c-3	t+3
		...		
		n	c-n	t+n

Decrement Address, Increment Tally, and Continue (Td = 15)

The action for this variation is identical to that for the Decrement Address, Increment Tally variation except that the TAG field of the Indirect Word is interpreted and continuation of the indirect chain is possible. If the TAG of the Indirect Word invokes a register, that is, specifies R, RI, or IR modification, the effective Td value for the register is forced to "null" before the next Effective Address is formed.

Increment Address, Decrement Tally (Td = 16)

For each reference to the Indirect Word, the ADDRESS field is increased by 1 and the TALLY field is reduced by 1 after the Effective Address is formed. ADDRESS arithmetic is modulo $2^{*}18$. TALLY arithmetic is modulo 4096. If the TALLY field is reduced to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The TAG field of the Indirect Word is ignored. The Effective Address is the value of the original unmodified ADDRESS field.

EXAMPLES

<u>Label</u>	<u>Instruction</u>	<u>Reference Count</u>	<u>Effective Address</u>	<u>Tally Value</u>
a	lda b, id	1	c	t-1
b	vfd 18/c,1/t	2	c-1	t-2
		3	c-2	t-3
		...		
		n	c-(n-1)	t-n

Increment Address, Decrement Tally, and Continue (Td = 17)

The action for this variation is identical to that for the Increment Address, Decrement Tally variation except that the TAG field of the Indirect Word is interpreted and continuation of the indirect chain is possible. If the TAG of the Indirect Word invokes a register, that

is, specifies R, RI, or IR modification, the effective Td value for the register is forced to "null" before the next Effective Address is formed.

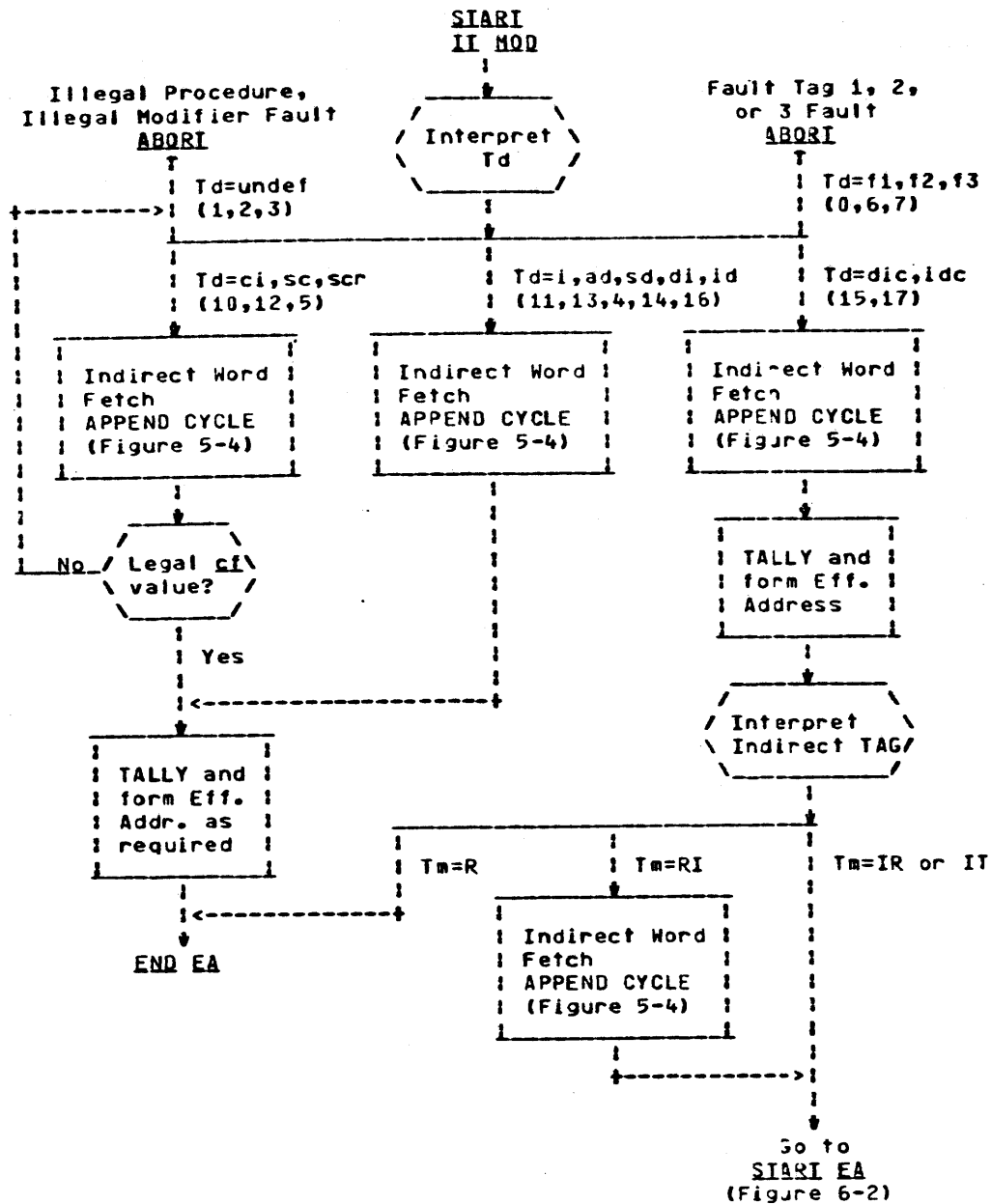


Figure 6-6 Indirect Then Tally Modification Flowchart

EFFECTIVE ADDRESS FORMATION INVOLVING BOTH SEGMENT NUMBER AND OFFSET

The second type of Address Formation allows formation of a modified Segment Number and a modified Offset simultaneously. See Figure 6-10, Effective Segment Number Generation Flowchart, for details.

If any of these conditions is violated, the special Address Modifier will be interpreted as a normal Address Modifier and will cause an Illegal Procedure, Illegal Modifier Fault.

Table 6-4 Special Append Mode Address Modifiers

TAG Value	Coding Mnemonic	Modification Name
41	itp	Indirect to Pointer
43	its	Indirect to Segment

INDIRECT TO POINTER (ITP) MODIFICATION

If the conditions above are satisfied, the Processor examines the TAG field of the Indirect Word for the value 41 (octal). If that value is found, the Indirect Word-pair is interpreted as an ITP Pointer Pair (See Figure 6-8 below for format) and the following actions take place:

For $\Omega = C(\text{ITP.PRUNUM})$:

$C(\text{PR}_{\Omega}.\text{SNR}) \rightarrow C(\text{TPR.TSR})$

$\text{maximum}(C(\text{PR}_{\Omega}.\text{RNR}), C(\text{SDW.R1}), C(\text{TPR.TRR})) \rightarrow C(\text{TPR.TRR})$

$C(\text{ITP.BITNO}) \rightarrow C(\text{TPR.TBR})$

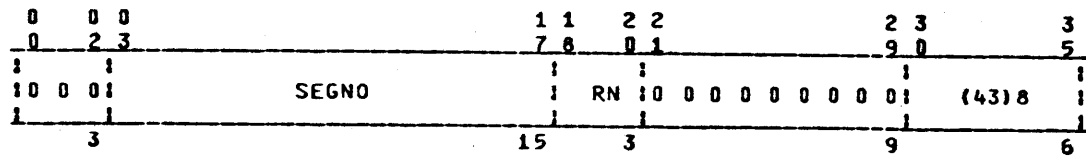
$C(\text{PR}_{\Omega}.\text{WORDNO}) + C(\text{ITP.WORDNO}) + C(\underline{r}) \rightarrow C(\text{TPR.CA})$

where:

- $\underline{r} = C(\text{CT-HOLD})$ if the TAG field of the Instruction Word or preceding Indirect Word specified Indirect Then Register modification, or
- $\underline{r} = C(\text{ITP.MOD.Id})$ if the TAG field of the Instruction Word or preceding Indirect Word specified Register Then Indirect modification and ITP.MOD specifies either Register or Register Then Indirect modification.
- SDW.R1 is the upper limit of the read/write Ring Bracket for the segment $C(\text{PR}_{\Omega}.\text{SNR})$. (See Section VIII, Hardware Ring Implementation.)

3. SDW.R1 is the upper limit of the read/write Ring Bracket for the segment C(ITS.SEGNO). (See Section VIII, Hardware Ring Implementation.)

Even Word



Odd Word

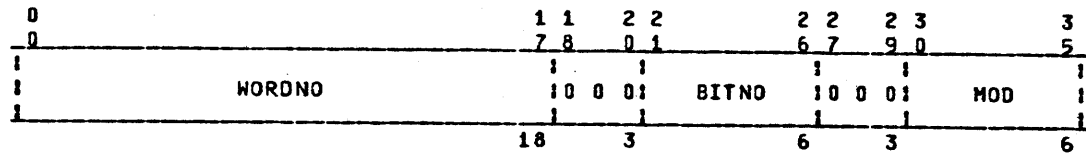


Figure 6-9 ITS Pointer Pair Format

Field Name	Meaning
SEGNO	The number of the segment to be referenced.
WORDNO	Word offset to be used in the effective address formation.
BITNO	The bit offset for the data item.
MOD	Any valid normal Address Modifier.

Effective Segment Number Generation

The details of Effective Segment Number generation are shown in the flowchart in Figure 6-10 below.

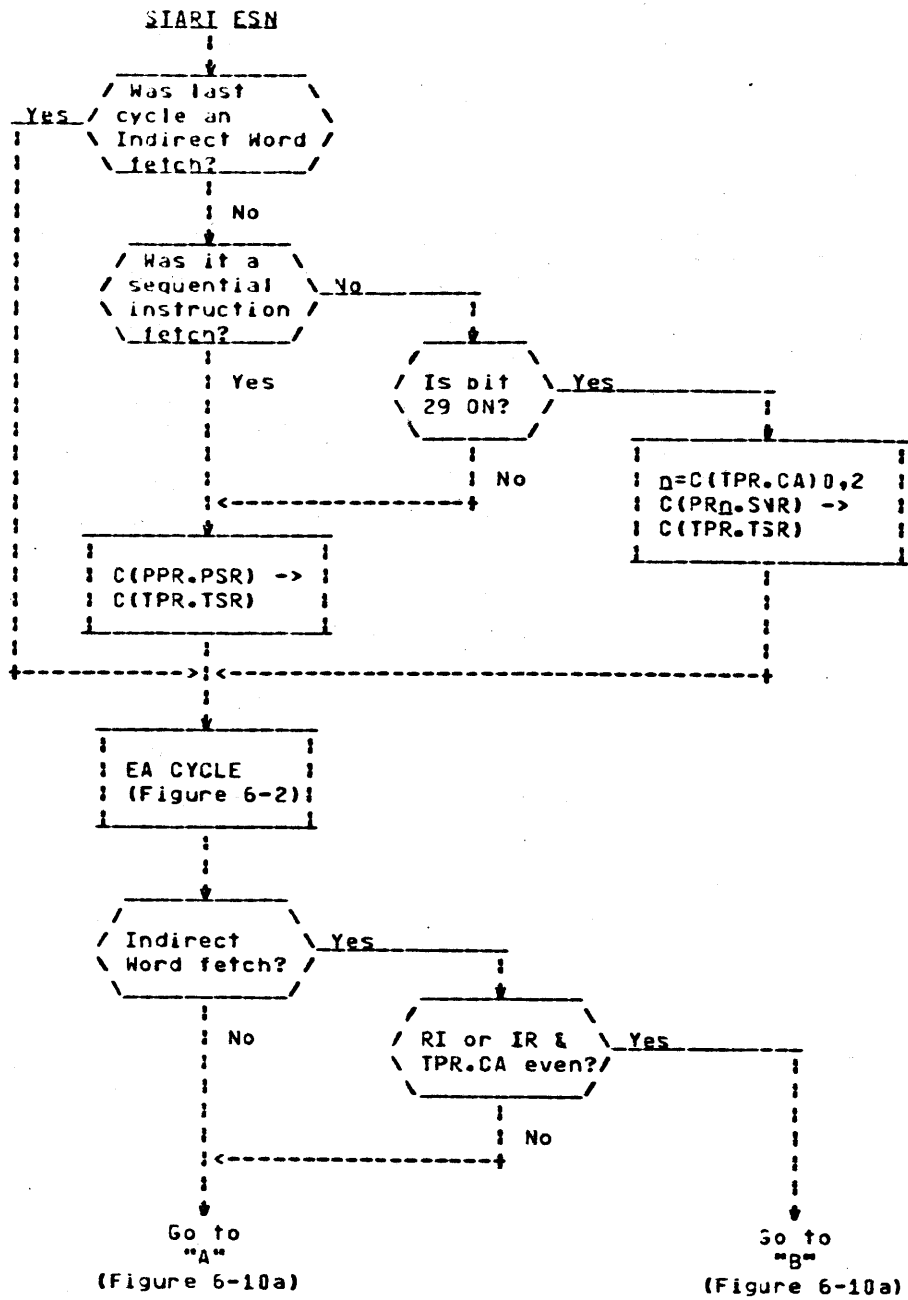


Figure 6-10 Effective Segment Number Generation Flowchart

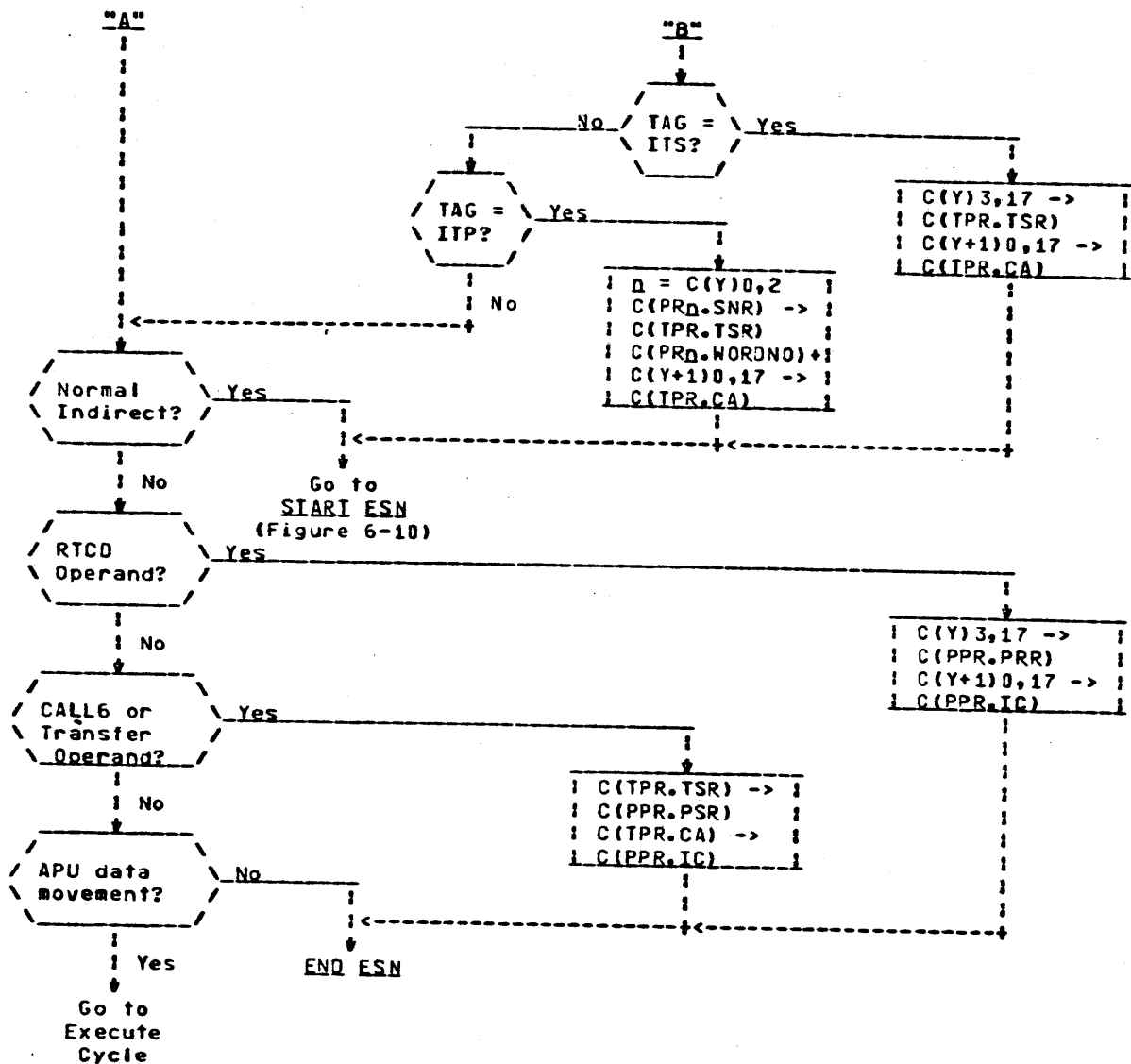


Figure 6-10a Effective Segment Number Generation Flowchart (Con't.)

EFFECTIVE ADDRESS FORMATION FOR EXTENDED INSTRUCTION SET

A flowchart of the steps involved in Operand Descriptor Effective Address Formation is shown in Figure 6-11 below. The flowchart depicts the Effective Address Formation for operand k as described by its Modification Field, MFk. This Effective Address Formation is performed for each operand as its Operand Descriptor is decoded.

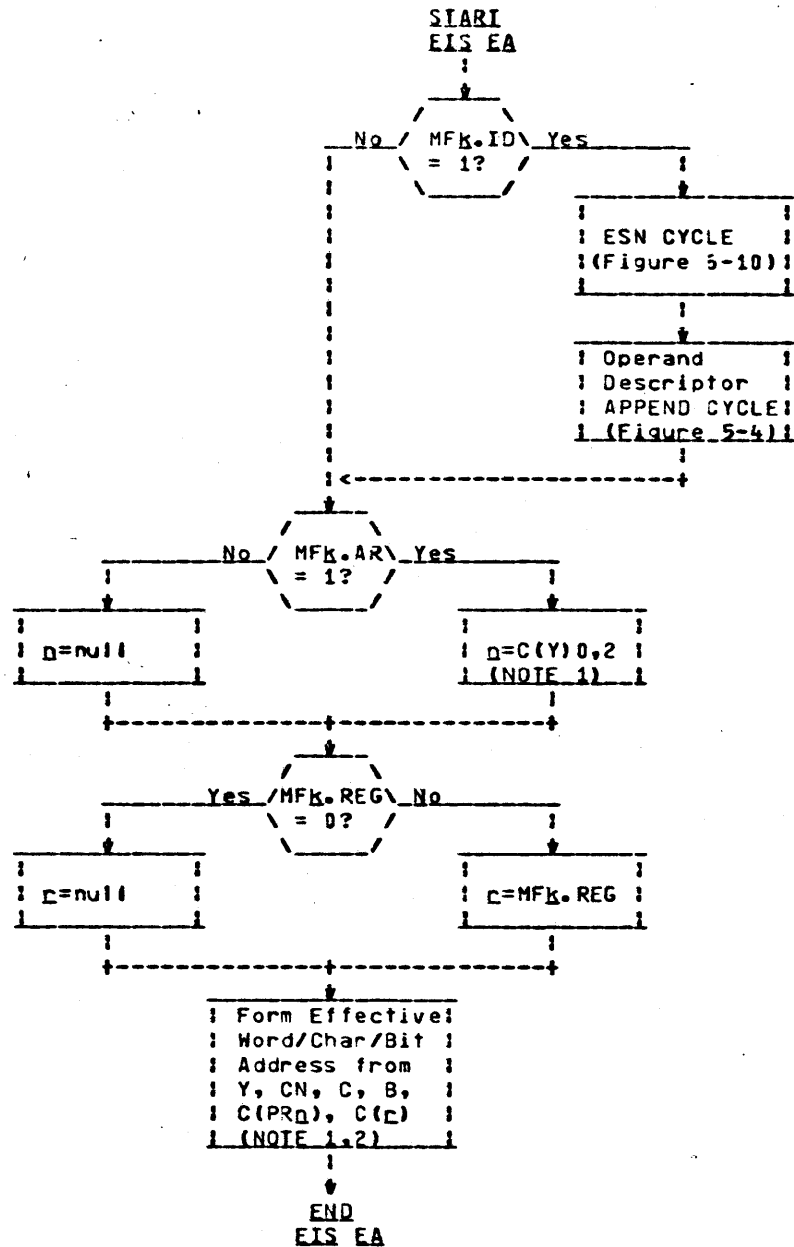


Figure 6-11 EIS Effective Address Formation Flowchart

NOTE 1: The symbol "y" stands for the contents of the ADDRESS field of the Operand Descriptor. The symbols "CN" and "C" stand for the contents of the Character Number field. The symbol "B" stands for the contents of the Bit Number field.

NOTE 2: The algorithms used in the formation of the Effective Word/Char/Bit Address are described in "Character- and Bit-String Addressing" following.

Character- and Bit-String Addressing

The Processor represents the Effective Address of a character- or bit-string operand in three different forms as follows:

1. Pointer Register Form

This form consists of a word value ($PR_n.WORDNO$) and a bit value ($PR_n.BITNO$). The word value is the word offset of the word containing the first character or bit of the operand and the bit value is the bit position of that character or bit within the word. This form is seen when $C(PR_n)$ are stored as an ITS Pointer Pair or as a Packed Pointer (See "Indirect to Segment (its) Modification" earlier in this Section).

2. Address Register Form

This form consists of a word value ($AR_n.WORDNO$), a character number ($AR_n.CHAR$), and a bit value ($AR_n.BITNO$). The word value is the word offset of the word containing the first character or bit of the operand. The character number is the number of the 9-bit character containing the first character or bit. The bit value is the bit bit position within $AR_n.CHAR$ of the first character or bit. This form is seen when $C(AR_n)$ are stored with the Store Address Register n (SAR_n) instruction.

3. Operand Descriptor Form

This form is valid for character-string operands only. It consists of a word value (ADDRESS) and a character number (CN). The word value is the word offset of the word containing the first character of the operand and the character number is the number of that character within the word. This form is seen when $C(AR_n)$ is stored with the AR_n to Alphanumeric Descriptor (ARA_n) or AR_n to Numeric Descriptor (ARN_n) instructions. (The Operand Descriptor form for bit-string operands is identical to the Address Register form.)

NOTE: The terms "Pointer Register" and "Address Register" both apply to the same physical hardware register. The distinction arises from the manner in which the register is invoked and used and in the interpretation of the register contents. "Pointer Register" refers to the register as used by the Appending Unit and "Address Register" refers to the register as used by the Decimal unit.

The three forms are compatible and may be freely intermixed. For example, PR_n may be loaded in Pointer Register form with the Effective Pointer to PR_n (EPP_n) instruction, then modified in Pointer Register form with the Effective Address to Word/Bit Number of PR_n ($EAWP_n$), then further modified in Address Register form (assuming character size k) with the Add k -Bit Displacement to

Address Register ($AkBD$) instruction, and finally invoked in Operand Descriptor form by the use of $MF.AR$ in an EIS Multiword instruction.

Character- and Bit-String Address Arithmetic Algorithms

The arithmetic algorithms for calculating character- and bit-string addresses are presented below. The symbols "ADDRESS" and "CN" represent the

ADDRESS and CN fields of the Operand Descriptor being decoded. "C" and "D" are set according to the flowchart in Figure 6-11 above. If either has the value "null", the contents of all fields shown is identically zero.

9-BIT CHARACTER STRING ADDRESS ARITHMETIC

$$\begin{aligned} \text{Effective BITNO} &= 0000 \\ \text{Effective CHAR} &= (\text{CN} + \text{C}(\text{AR}_D.\text{CHAR}) + \text{C}(\text{C})) \text{ modulo } 4 \\ \text{Effective WORDNO} &= \text{ADDRESS} + \text{C}(\text{AR}_D.\text{WORDNO}) + \\ &\quad (\text{CN} + \text{C}(\text{AR}_D.\text{CHAR}) + \text{C}(\text{C})) / 4 \end{aligned}$$

6-BIT CHARACTER STRING ADDRESS ARITHMETIC

$$\begin{aligned} \text{Effective BITNO} &= (9 * \text{C}(\text{AR}_D.\text{CHAR}) + 6 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) \text{ modulo } 9 \\ \text{Effective CHAR} &= ((9 * \text{C}(\text{AR}_D.\text{CHAR}) + 6 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) \text{ modulo } 36) / 9 \\ \text{Effective WORDNO} &= \text{ADDRESS} + \text{C}(\text{AR}_D.\text{WORDNO}) + \\ &\quad (9 * \text{C}(\text{AR}_D.\text{CHAR}) + 6 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) / 36 \end{aligned}$$

4-BIT CHARACTER STRING ADDRESS ARITHMETIC

$$\begin{aligned} \text{Effective BITNO} &= 4 * (\text{C}(\text{AR}_D.\text{CHAR}) + 2 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) / 4 \text{ modulo } 2 + 1 \\ \text{Effective CHAR} &= ((9 * \text{C}(\text{AR}_D.\text{CHAR}) + 4 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) \text{ modulo } 36) / 9 \\ \text{Effective WORDNO} &= \text{ADDRESS} + \text{C}(\text{AR}_D.\text{WORDNO}) + \\ &\quad (9 * \text{C}(\text{AR}_D.\text{CHAR}) + 4 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) / 36 \end{aligned}$$

BIT STRING ADDRESS ARITHMETIC

$$\begin{aligned} \text{Effective BITNO} &= (9 * \text{C}(\text{AR}_D.\text{CHAR}) + 36 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) \text{ modulo } 9 \\ \text{Effective CHAR} &= ((9 * \text{C}(\text{AR}_D.\text{CHAR}) + 36 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) \text{ modulo } 36) / 9 \\ \text{Effective WORDNO} &= \text{ADDRESS} + \text{C}(\text{AR}_D.\text{WORDNO}) + \\ &\quad (9 * \text{C}(\text{AR}_D.\text{CHAR}) + 36 * \text{C}(\text{C}) + \text{C}(\text{AR}_D.\text{BITNO})) / 36 \end{aligned}$$

SECTION VII

FAULTS AND INTERRUPTS

Faults and Interrupts both result in an interruption of normal sequential processing, but there is a difference in how they originate. Generally, Faults are caused by events or conditions that are internal to the Processor and Interrupts are caused by events or conditions that are external to the Processor. Faults and Interrupts enable the Processor to respond promptly when conditions occur that require system attention. A unique word-pair is dedicated for the instructions to service each Fault and Interrupt condition. The instruction pair associated with a Fault is called the Fault Vector. The instruction pair associated with an Interrupt is called the Interrupt Vector.

FAULT CYCLE SEQUENCE

Following the detection of a Fault condition, the Control Unit determines the proper time to initiate the Fault Sequence according to the Fault Group. At that time, the Control Unit interrupts normal sequential processing with an Abort Cycle. The Abort Cycle brings all overlapped and asynchronous functions within the Processor to an orderly halt. At the end of the Abort Cycle, the Control Unit initiates a Fault Cycle.

In the Fault Cycle, the Processor safe-stores the Control Unit Data (See Section IV, Program Accessible Registers) into program-invisible holding registers in preparation for a Store Control Unit (SCU) instruction, then enters Temporary Absolute Mode and generates an Effective Address for the Fault Vector by concatenating the setting of the FAULT CONTROL switches on the Processor Maintenance panel with twice the Fault Number (See Table 7-1). This Effective Address and the Operation Code for the Execute Double (XED) instruction are forced into the Instruction Register and executed as an instruction. Note that the execution of the instruction is not done in a normal Execute Cycle but in the Fault Cycle with the Processor in Temporary Absolute Mode.

If the attempt to fetch and execute the instruction pair at the Fault Vector results in another Fault, the current Fault Cycle is aborted and a new Fault Cycle for the Trouble Fault (Fault Number 31) is initiated. In the Fault Cycle for a Trouble Fault, the Processor does not safe-store the Control Unit

Data. Therefore, it may be possible to recover the conditions for the original Fault by use of the Store Control Unit (SCU) instruction.

If either of the two instructions in the Fault Vector results in a transfer of control to an Effective Address generated in Absolute Mode, the Absolute Mode Indicator is set ON for the transfer and remains ON thereafter until changed by program action.

If either of the two instructions in the Fault Vector results in a transfer of control to an Effective Address generated in Append Mode (through the use of bit 29 of the instruction word or by use of the itp or itp modifiers), the transfer is made in the Normal Mode and the Processor remains in Normal Mode thereafter.

If no transfer of control takes place, the Processor returns to the mode in effect at the time of the Fault and resumes normal sequential execution with the instruction following the Faulting instruction (C(PPR.IC) + 1).

Many of the Fault conditions are deliberately or inadvertently caused by the software and do not necessarily involve error conditions. The operating supervisor determines the proper action for each Fault condition by analyzing the machine conditions at the time of the Fault. Therefore, it is necessary that the first instruction in each of the Fault Vectors be the Store Control Unit (SCU) instruction and the second be a transfer to a routine to analyze the machine conditions. If a Fault condition is to be intentionally ignored, the Fault Vector for that condition should contain an SCU/RCU pair referencing a unique Y-block8. By use of this pair, the machine conditions for the ignored Fault condition may be recovered if the ignored Fault causes a Trouble Fault.

FAULT PRIORITY

The Multics Processor has provision for 32 Faults of which 27 are implemented. The Faults are classified into seven Fault Priority Groups that roughly correspond to the severity of the Faults. Fault Priority Groups are defined so that Fault recognition precedence may be established when two or more Faults exist concurrently. Overlap and asynchronous functions in the Processor allow the simultaneous occurrence of Faults. Group 1 has the highest priority and Group 7 has the lowest. In Groups 1 through 6, only one Fault within each Group is allowed to be active at any one time. The first Fault within a Group occurring through the normal program sequence is the one serviced.

In Group 7 Faults are saved by the hardware for eventual recognition. In the case of simultaneous Faults within group 7, Shutdown has the highest priority, Timer Runout is next, and Connect has the lowest priority.

There is a single exception to the handling of Faults in Priority Group order. If an operand fetch generates a Parity Fault and the use of the operand in "closing out" instruction execution generates an Overflow Fault or a Divide Check Fault, these Faults are considered simultaneous but the Parity Fault takes precedence.

Table 7-1. List of Faults

OCTAL NUMBER	DECIMAL NUMBER	MNEMONIC	NAME	PRIORITY	GROUP
0	0	sdf	Shutdown	27	7
1	1	str	Store	10	4
2	2	mme	Master Mode Entry 1	11	5
3	3	f1	Fault Tag 1	17	5
4	4	tro	Timer Runout	26	7
5	5	cmd	Command	9	4
6	6	drl	Derail	15	5
7	7	luf	Lockup	5	4
10	8	con	Connect	25	7
11	9	par	Parity	8	4
12	10	ipr	Illegal Procedure	15	5
13	11	onc	Op Not Complete	4	2
14	12	suf	Startup	1	1
15	13	off	Overflow	7	3
16	14	div	Divide Check	6	3
17	15	exf	Execute	2	1
20	16	df0	Directed Fault 0	20	6
21	17	df1	Directed Fault 1	21	6
22	18	df2	Directed Fault 2	22	6
23	19	df3	Directed Fault 3	23	6
24	20	acv	Access Violation	24	6
25	21	mme2	Master Mode Entry 2	12	5
26	22	mme3	Master Mode Entry 3	13	5
27	23	mme4	Master Mode Entry 4	14	5
30	24	f2	Fault Tag 2	18	5
31	25	f3	Fault Tag 3	19	5
32	26		Unassigned		
33	27		Unassigned		
34	28		Unassigned		
35	29		Unassigned		
36	30		Unassigned		
37	31	trb	Trouble	3	2

FAULT RECOGNITION

For the discussion following, the term "function" is defined as a major Processor functional cycle. Examples are: APPEND CYCLE, EA CYCLE, Instruction Fetch Cycle, Operand Store Cycle, Divide Execution Cycle.

Faults in Groups 1 and 2 cause the Processor to abort all functions immediately by initializing itself and enter a Fault Cycle.

Faults in Group 3 cause the Processor to "close out" current functions without taking any irrevocable action (such as setting PTW.J in an APPEND CYCLE

or modifying an Indirect Word in an EA CYCLE), then to discard any pending functions (such as an APPEND CYCLE needed during an EA CYCLE), and to enter a Fault Cycle.

Faults in Group 4 cause the Processor to suspend overlapped operation, complete current and pending functions for the current instruction, and then enter a Fault Cycle.

Faults in groups 5 or 6 are normally detected during Address Preparation and Instruction Decode. These Faults cause the Processor to suspend overlapped operation, complete the current and pending instructions, and to enter a Fault Cycle. If a Fault in a higher Priority Group is generated by the execution of the current or pending instructions, that higher priority Fault will take precedence and the Group 5 or 6 Fault will be lost. If a Group 5 or 6 Fault is detected during execution of the current instruction, (for example, an Access Violation, Out of Segment Bounds Fault during certain interruptible EIS instructions), the instruction is considered "complete" upon detection of the Fault.

Faults in Group 7 are held and processed (with Program Interrupts) at the completion of the current instruction pair. Group 7 Faults are inibitable by use of bit 28 of the Instruction Word.

Faults in Groups 3 through 6 must wait for the System Controller to acknowledge the last access request before entering the Fault Cycle.

FAULT DESCRIPTIONS

Group 1 Faults

Startup

- DC POWER has been turned on. When the POWER ON button is depressed, the Processor is first initialized and then the Startup Fault is recognized.

Execute

1. The EXECUTE pushbutton on the Processor maintenance panel has been pressed.
2. An external gate signal has been substituted the EXECUTE pushbutton. for EXECUTE pushbutton.

The selection between the above conditions is made by settings of verious switches on the Processor Maintenance panel.

Group 2 Faults

Op Not Complete

Any of the following will cause an Op Not Complete Fault:

1. The Processor has addressed a System Controller to which it is not attached.
2. The addressed System Controller failed to acknowledge the Processor.
3. The Processor has not generated a main store access request or a direct operand within 1 to 2 milliseconds and is not in the DIS

state.

4. A Processor port received a data strobe without a preceding acknowledgement from the System Controller that it has received the access request.
5. A Processor port received a data strobe before the data previously sent to it was unloaded.

Trouble

The Trouble Fault is defined as the occurrence of a Fault during the fetch or execution of a Fault Vector or Interrupt Vector. Such Faults may be hardware generated (for example, Op Not Complete or Parity), or operating system generated (for example, the page containing the effective address of an instruction is missing).

Group 3 Faults

Overflow

An arithmetic overflow, exponent overflow, or exponent underflow has been generated. The generation of this Fault is inhibited with the Overflow Mask indicator set ON. Subsequent resetting of the Overflow Mask indicator to OFF does not generate this Fault from previously set overflow indicators. The Overflow Fault Mask state does not affect the setting, testing or storing of indicators. The determination of the specific overflow condition is by indicator testing by the control program.

Divide Check

A Divide Check Fault occurs when the actual division cannot be carried out for one of the reasons specified with individual divide instructions.

Group 4 Faults

Store

The Processor attempted to select a disabled port, an out-of-bounds address was generated in the BAR Mode or Absolute Mode, or an attempt was made to access a store unit that was not ready.

Command

1. The Processor attempted to load or read the Interrupt Mask Register in a System Controller in which it did not have an Interrupt Mask assigned.
2. The Processor issued an XEC command to a System Controller in which it did not have an Interrupt Mask assigned.
3. The Processor issued a Connect to a System Controller port that is masked OFF.

4. The selected System Controller is in TEST mode and a condition determined by certain System Controller Maintenance panel switches has been trapped.
5. An attempt was made to load a Pointer Register with Packed Pointer data in which the BITNO field value was greater than 60 octal.

Lockup

The program is in a code sequence which has inhibited sampling for an external interrupt (whether present or not) or Group 7 Fault for longer than the prescribed time. In Absolute Mode or Privileged Mode the lockup time is 32 milliseconds. In Normal Mode or BAR Mode the lockup time is specified by the setting of the Lockup Timer in the Cache Mode Register. The Lock Timer is program settable to 2, 4, 8, or 16 milliseconds.

While in Absolute Mode or Privileged Mode the Lockup Fault is signalled at the end of the time limit set in the Lockup Timer but is not recognized until the 32 millisecond limit. If the Processor returns to Normal Mode or BAR Mode after the Fault has been signalled but before the 32 millisecond limit, the Fault is recognized before any instruction in the new mode is executed.

Parity

1. The selected System Controller has returned an Illegal Action signal with an Illegal Action Code for one of the various main store parity error conditions.
2. A Cache data parity error has occurred either for read, write, or block load. Cache status bits for the condition have been set in the Cache Mode Register.
3. The Processor has detected a parity error in the System Controller interface port while either generating outgoing parity or verifying incoming parity.

Group 5 Faults

Master Mode Entries 1-4

The corresponding Master Mode Entry instruction has been decoded.

Fault Tags 1-3

The corresponding Indirect Then Tally variation designator has been

detected during Address Preparation.

Derail

The Derail instruction has been decoded.

Illegal Procedure

1. An illegal operation code has been decoded or an illegal instruction sequence has been encountered.
2. An illegal modifier or modifier sequence has been encountered during Address Preparation.
3. An illegal address has been given in an instruction that the ADDRESS field for register selection.
4. An attempt was made to execute a privileged instruction in Normal Mode or BAR Mode..
5. An illegal digit was encountered in a Decimal Numeric operand.

The conditions for the Fault will be set in the Fault Register, Word 1 of the Control Unit Data, or in both.

Group 6 Faults

Directed Faults 0-3

A faulted Segment Descriptor Word (SDW) or Page Table Word (PTW) with the corresponding Directed Fault number has been fetched by the Appending Unit.

Access Violation

The Appending Unit has detected one of the several access violations below. Word 1 of the Control Unit Data contains status bits for the condition.

1. Not in read bracket (ACV3=ORB)
2. Not in write bracket (ACV5=OWB)
3. Not in execute bracket (ACV1=OEB)
4. No read permission (ACV4=R-OFF)
5. No write permission (ACV6=W-OFF)
6. No execute permission (ACV2=E-OFF)
7. Invalid ring crossing (ACV12=CRT)
8. Call limiter fault (ACV7=NO GA)
9. Outward call (ACV9=OCALL)
10. Bad outward call (ACV10=BOC)
11. Inward return (ACV11=INRET)
12. Ring alarm (ACV13=RALR)
13. Associative Memory error
14. Out of segment bounds
15. Illegal ring order (ACV0=IRO)
16. Out of call brackets (ACV8=OCB)

Group 7 Faults

Shutdown

An external power shutdown condition has been detected. DC POWER shutdown will occur in approximately one millisecond.

Timer Runout

The Timer Register has decremented to or through the value zero. If the Processor is in Privileged Mode or Absolute Mode, recognition of this Fault is delayed until a return to Normal Mode or BAR Mode. Counting in the Timer Register continues.

Connect

A connect signal (\$CON strobe) has been received from a System Controller. This event is to be distinguished from a CIOC (connect) instruction encountered in the program sequence.

PROGRAM INTERRUPTS AND EXTERNAL FAULTS

Each System Controller contains 32 Execute Interrupt Cells that are used for communication among the active system modules (Processors, I/O Multiplexers, etc.). The Execute Interrupt Cells are organized in a numbered priority chain. Any active system module connected to a System Controller port may request the setting of an Execute Interrupt Cell with the SXC command.

When one or more Execute Interrupt Cells in a System Controller is set, the System Controller activates the Execute Interrupt Present (XIP) line to all System Controller ports having an Execute Interrupt Mask assigned in which one or more of the Execute Interrupt Cells that are set is unmasked. Execute Interrupt Masks are assigned only to Processors. Each Execute Interrupt Cell has a unique Interrupt Vector located at an Absolute Address equal to twice the cell number.

Execute Interrupt Sampling

The Processor always fetches instructions in pairs. At an appropriate point (as early as possible) in the execution of a pair of instructions, the next sequential instruction pair is fetched and held in a special instruction buffer register. The exact point depends on instruction sequence and other conditions.

If the Interrupt Inhibit Bit (bit 28) is not set in the current instruction word at the point of next sequential instruction address preparation, the Processor samples the Group 7 Faults. If any of the Group 7 Faults is found, the next sequential instruction pair is not fetched and an internal flag is set reflecting the presence of the Fault. The Processor next samples the Execute Interrupt Present lines from all eight Processor ports and loads a register with bits corresponding to the states of the lines. If any bit in the register is set ON, the next sequential instruction pair is not fetched and an internal flag is

set reflecting the presence of the bit(s) in the register.

NOTE: If the instruction pair address is being prepared as the result of a transfer of control condition or if the current instruction is Execute (XEC), Execute Double (XED), Repeat (RPT), Repeat Double (RPD), or Repeat Link (RPL), the Group 7 Faults and Execute Interrupt Present lines are not sampled.

At the completion of the current instruction pair (if no transfer of control has occurred) and the Processor is ready for the next instruction pair and the Group 7 Fault flag is set, the Processor will enter a Fault Cycle for the highest priority Group 7 Fault present.

At the completion of the current instruction pair (if no transfer of control has occurred) and the Processor is ready for the next instruction pair and the Execute Interrupt Present flag is set, the Processor will enter an Execute Interrupt Cycle.

Execute Interrupt Cycle Sequence

In the Execute Interrupt Cycle, the Processor safe-stores the Control Unit Data (See Section IV, Program Accessible Registers) into program-invisible holding registers in preparation for a Store Control Unit (SCU) instruction, then enters Temporary Absolute Mode. It then issues an XEC command to the System Controller on the highest priority port for which there is a bit set in the Execute Interrupt Present register.

The selected System Controller responds by clearing its highest priority Execute Interrupt Cell and returning the Interrupt Vector address for that cell to the Processor.

NOTE: If there is no Execute Interrupt Cell set in the selected System Controller (implying that all have been cleared in response to XEC commands from other Processors), the System Controller will return the address value 1 which is not a valid Interrupt Vector address. The Processor senses this value, aborts the Execute Interrupt Cycle, and returns to normal sequential instruction processing.

The Interrupt Vector address returned and the Operation Code for the Execute Double (XED) instruction are forced into the Instruction Register and executed as an instruction. Note that the execution of the instruction is not done in a normal Execute Cycle but in the Execute Interrupt Cycle with the Processor in Temporary Absolute Mode.

If the attempt to fetch and execute the instruction pair at the Interrupt Vector results in a Fault, the Execute Interrupt Cycle is aborted and a Fault Cycle for the Trouble Fault (Fault Number 31) is initiated. In the Fault Cycle for a Trouble Fault, the Processor does not safe-store the Control Unit Data. Therefore, it may be possible to recover the conditions for the Execute Interrupt by use of the Store Control Unit (SCU) instruction.

If either of the two instructions in the interrupt Vector results in a transfer of control to an Effective Address generated in Absolute Mode, the Absolute Mode indicator is set ON for the transfer and remains ON thereafter until changed by program action.

If either of the two instructions in the Interrupt Vector results in a transfer of control to an Effective Address generated in Append Mode (through the use of bit 29 of the instruction word or by use of the itp or itp modifiers), the transfer is made in the Normal Mode and the Processor

remains in Normal Mode thereafter.

If no transfer of control takes place, the Processor returns to the mode in effect at the time of the Fault and resumes normal sequential execution with the instruction following the interrupted instruction (C(PPR.IC) + 1).

NOTE: Due the time required for many of the EIS data movement instructions, additional Group 7 Fault and Execute Interrupt present sampling is done during these instructions. After the initial load of the Decimal Unit input data buffer, Group 7 Faults and Execute Interrupt Present are sampling for each input operand address preparation. The instruction in execution is interrupted before the operand is fetched and flags are set into Control Unit Data to signal the restart of the instruction.

SECTION VIII

HARDWARE RING IMPLEMENTATION

RING PROTECTION PHILOSOPHY

The basic concept in the ring protection philosophy is the existence of a set of hierarchical levels of protection. A graphic representation of the concept may be given by a set of N consecutive circles, numbered $0, 1, 2, \dots, N-1$ from the inside out. The space included in circle 0 is called ring 0 , the space included between circle $i-1$ and i is called ring i . Any segment in the system is placed in one and only one ring. The closer a segment to the center, the greater its protection and access privileges.

When a process is executing a procedure segment placed in ring R , the process is said to be in ring R or also it is said that the current ring of the process is ring R . A process in ring R potentially has access to any segment located in ring R and in outer rings. The word "potentially" is used because the final decision is subject to what access rights (read, write, execute) the user has for the given segment. On the other hand, this same process in ring R has no access to any segment located in inner rings, except to special procedures called "gates." Gates are procedures residing in a given ring and intended to provide controlled access to this ring. A process that is in ring R can enter an inner ring r only by calling one of the gate procedures associated with this inner ring r . Gates must be carefully coded and must not trust any data that has been manufactured or modified by the caller in a less privileged ring. In particular, they must validate all arguments passed to them by the caller so as not to compromise the protection of any segment residing in the inner ring.

Calls from an outer ring to an inner ring are referred to as "inward calls." They are associated with an increase in the access capability of the process and are controlled by gates. On the other hand, calls from an inner ring to an outer ring, referred to as "outward calls" are associated with a decrease in the access capability of the process and do not need to be controlled.

RING PROTECTION IN MULTICS

The ring protection designed for the Multics System uses the philosophy described above, but a few points have been altered in order to obtain more flexibility and better efficiency.

First, the assignment of a segment to one and only one ring, although sufficient to implement the solution of the protection problem, may be very inconvenient for a class of procedure segments, such as the library routines. Such procedures operate correctly in whatever ring the process is at the time they are called; they need no more access than the caller, and they might not perform correctly with less access than the caller. One solution could have been to have one copy of the library in each ring. Instead, the solution adopted by Multics was to relax the condition that a segment can be assigned to only one ring and allow a procedure segment to be assigned to a set of consecutive rings defined by two integers (r_1, r_2) , with $r_1 \leq r_2$. Such a procedure now, resides in rings r_1 to r_2 . If it is called from ring R such that $r_1 \leq R \leq r_2$, then it behaves as if it were in ring R , and executes without changing the current ring of the process. If it is called from ring R such that $R > r_2$, then it behaves like a gate associated with ring r_2 , accepting the call as an Inward Call and decreasing the current ring of the process from R to r_2 . Upon return to the caller, the current ring is restored to R , of course. Note that by allowing the multiple ring residency for a procedure segment, the current ring of a process is no longer defined by the procedure in execution; a new variable must be introduced to keep track of the value of the current ring.

Second, it was found desirable to be able to specify the maximum ring number from which a given gate was allowed to be called. And a third integer r_3 was added to the pair of integers already associated with a segment. Any procedure segment, now, is associated with three ring numbers (r_1, r_2, r_3) called its "ring brackets", such that $r_1 \leq r_2 \leq r_3$. By convention, if $r_3 > r_2$, the procedure is a gate for ring r_2 , accessible from rings no higher than r_3 ; if $r_2 = r_3$, the procedure is not a gate.

Third, it was found useful to relax, also for data segments, the condition that they be assigned to only one ring. One would like to be able to specify that a segment resides in ring r_1 for "write" purposes but resides in a less privileged ring r_2 for "read" purposes.

Fourth, several difficulties were encountered in the implementation of outward calls and their associated returns. Because outward calls were not found essential for implementing the Multics system, they were simply declared illegal, and as a result, a procedure with ring brackets (r_1, r_2, r_3) cannot be called from a ring R such that $R < r_1$.

In summary, the operations that are potentially permitted to a process in ring R on a segment whose ring brackets are (r_1, r_2, r_3) are as follows:

Write	:	if $0 \leq R \leq r_1$
Read	:	if $0 \leq R \leq r_2$
Execute	:	if $r_1 \leq R \leq r_2$ (Execution in ring R)
Inward call	:	if $r_2 < R \leq r_3$ (Execution in ring r_2)

The attempted operations are permitted if, in addition, the user has the appropriate access rights (read, write, execute) on that segment.

RING PROTECTION IN THE MULTICS PROCESSOR

The Multics Processor offers hardware support for the implementation of the Multics ring protection. A particular effort was made to minimize the overhead associated with all authorized ring crossings, which the processor performs without operating system intervention, and to minimize the overhead associated with the validation of arguments, for which the processor provides a valuable

assistance.

The number of rings available in the processor is eight, numbered from 0 to 7. The current ring R of a process is recorded in a hardware register (PPR.PRR).

The ring brackets (r1, r2, r3) of a Segment are recorded in the Segment Descriptor Word (SDW) used by the hardware to access the segment. In addition, the SDW contains the number of gates (SDW.CL) existing in the segment. The hardware assumes that all gates are located from word 0 to word (CL-1) and does not accept an inward call to this segment if the word number specified in the call is greater than (CL-1). The reason for this control is to prevent a malicious user from generating a call that would transfer control to any machine instruction of the gate procedure. (Such a call would defeat the purpose of the gate.) The SDW also contains the access rights (read, write, execute) that the user has on that segment. If the same segment is used by several processes, there is an SDW describing the segment in the Descriptor Segment of each process. In all SDWs pointing to the same segment, the values of r1, r2, r3 and CL are identical since they are user independent. The value of the access rights (read, write, execute) are not necessarily the same because they are user dependent.

In order to provide assistance in argument validation, any pointer, being stored into an ITS Pointer Pair or loaded into a Pointer Register, also contains a ring number. Although the hardware does not prevent a process from writing any ring number in an ITS Pointer Pair, it ensures that, if (r1, r2, r3) are the ring brackets of the segment in which the ITS Pointer Pair is located, the ring number field of this ITS Pointer Pair can be set or modified only from ring R such that $R \leq r1$. As for the ring number recorded in a Pointer Register, the hardware ensures that a process in ring R can set it to a value equal to or greater than R, but never smaller.

During the execution of a machine instruction, the hardware may examine several SDWs, ITS Pointer Pairs and Pointer Registers. For any given such examination, the hardware records the maximum of the current ring, the r1 value found in an SDW, the ring number found in an ITS Pointer Pair, or the ring number found in a Pointer Register. This maximum, called the Temporary Ring Number, is kept in a hardware register (TPR.TRR) that is updated each such examination.

The reason for having this Temporary Ring Number available at any point of a machine instruction is because it represents the highest ring (least privileged) that might have created or modified any information that led the hardware to the target segment it is about to reference. Although the current ring is R, the hardware uses the most pessimistic approach and pretends the current ring is C(TPR.TRR), which is always equal to or greater than R. Thus the hardware uses C(TPR.TRR) instead of R in all comparisons with the ring brackets involved in the enforcement of the ring protection rules given in the previous paragraph.

The use of C(TPR.TRR) by the hardware allows the gate procedures to rely on the hardware to perform the validation of all addresses passed to the gate by the less privileged ring. The general rule enforced here by the hardware regarding argument validation can be stated as follows: whenever an inner ring performs an operation on a given segment and references that segment through pointers manufactured by an outer ring, the operation is considered valid only if it could have been performed while in the outer ring.

APPENDING UNIT OPERATION WITH RING MECHANISM

The complete flowchart for Effective Segment Number generation, including the hardware ring mechanism, is shown in Figure 8-1 below. See the description of the Access Violation Fault in Section VII of this document for the meanings of the coded faults. The current instruction is in the Instruction Working Buffer (IWB).

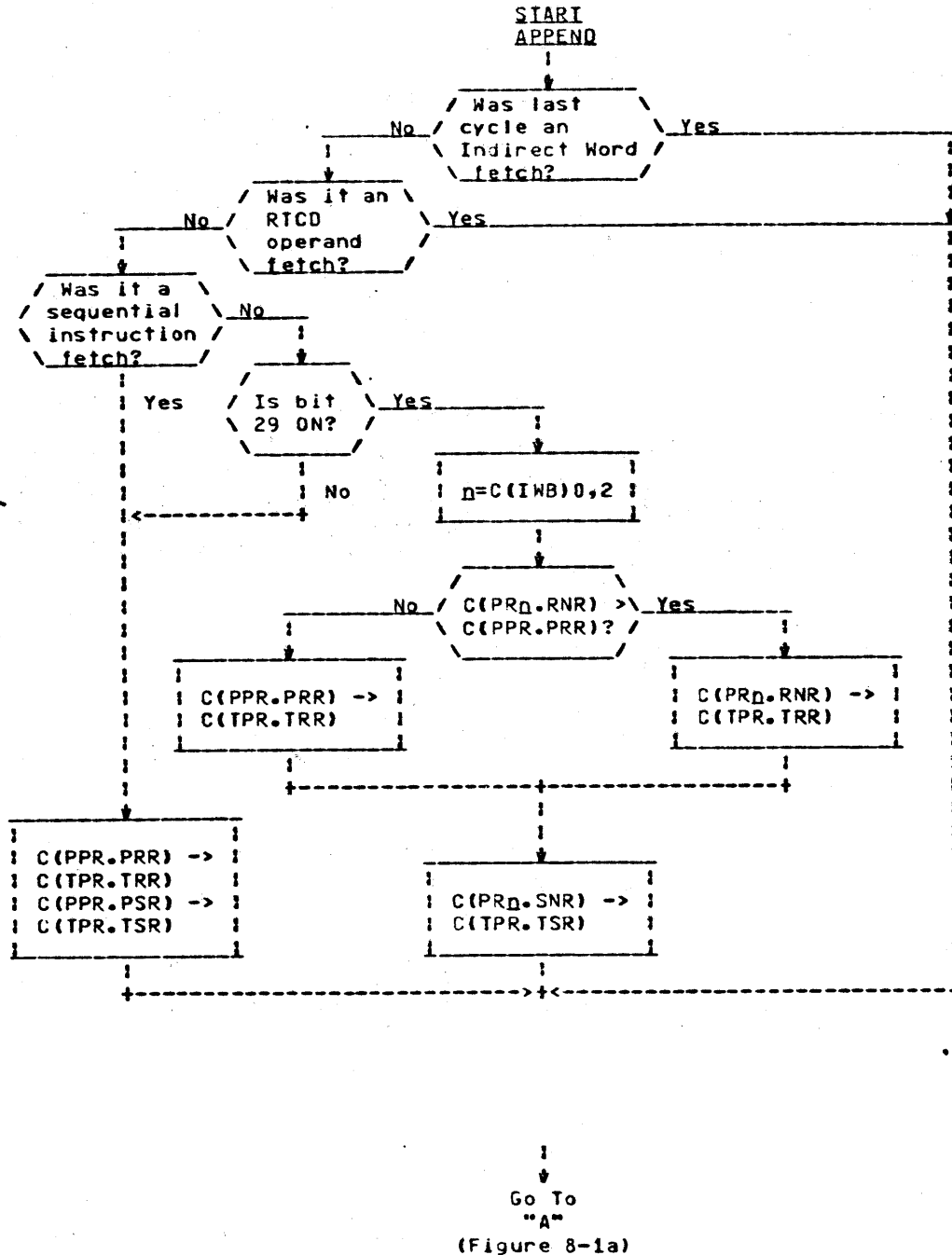
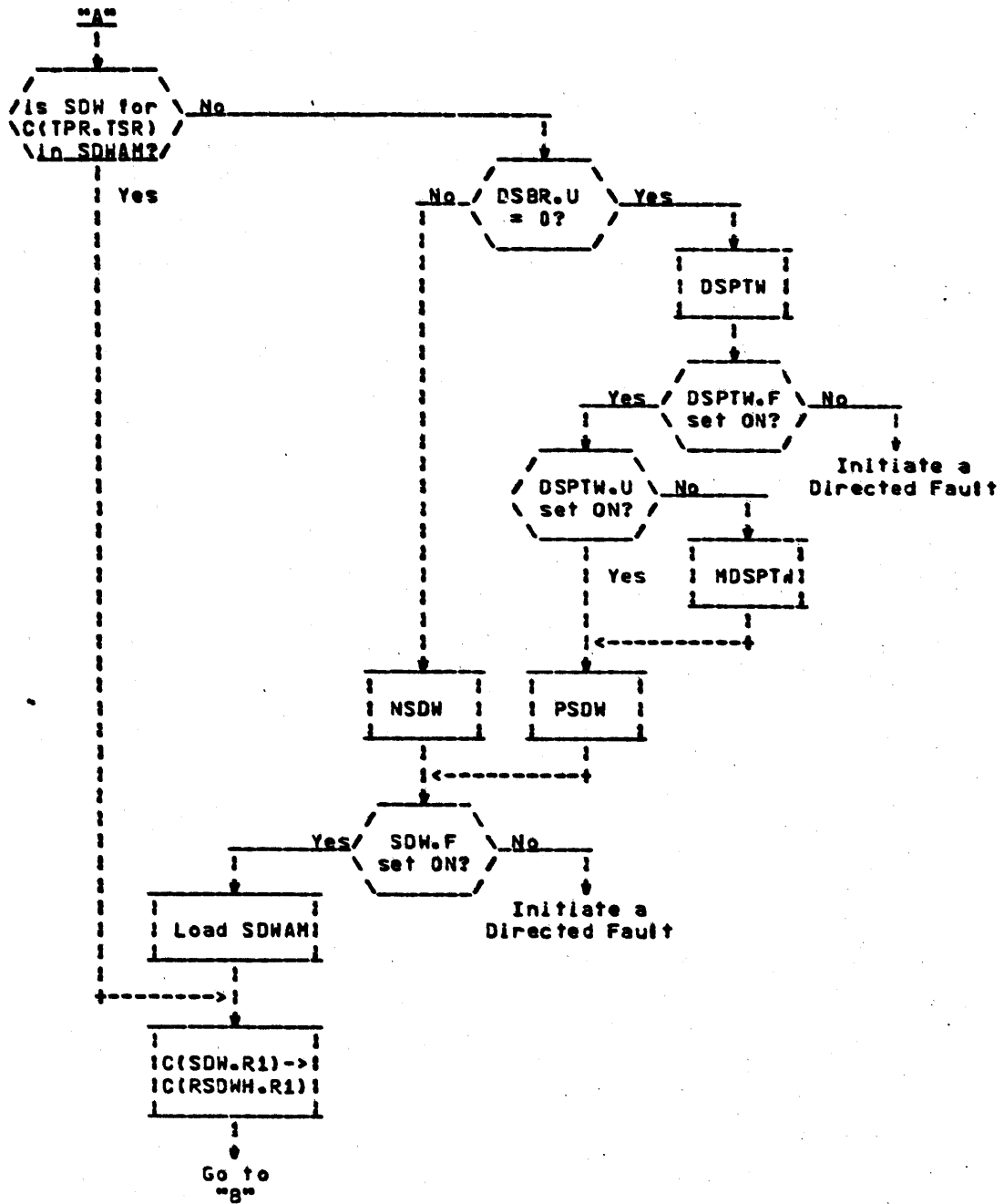


Figure 8-1 Complete Appending Unit Operation Flowchart



(Figure 8-1b)

Figure 8-1a Complete Appending Unit Operation Flowchart (con't.)

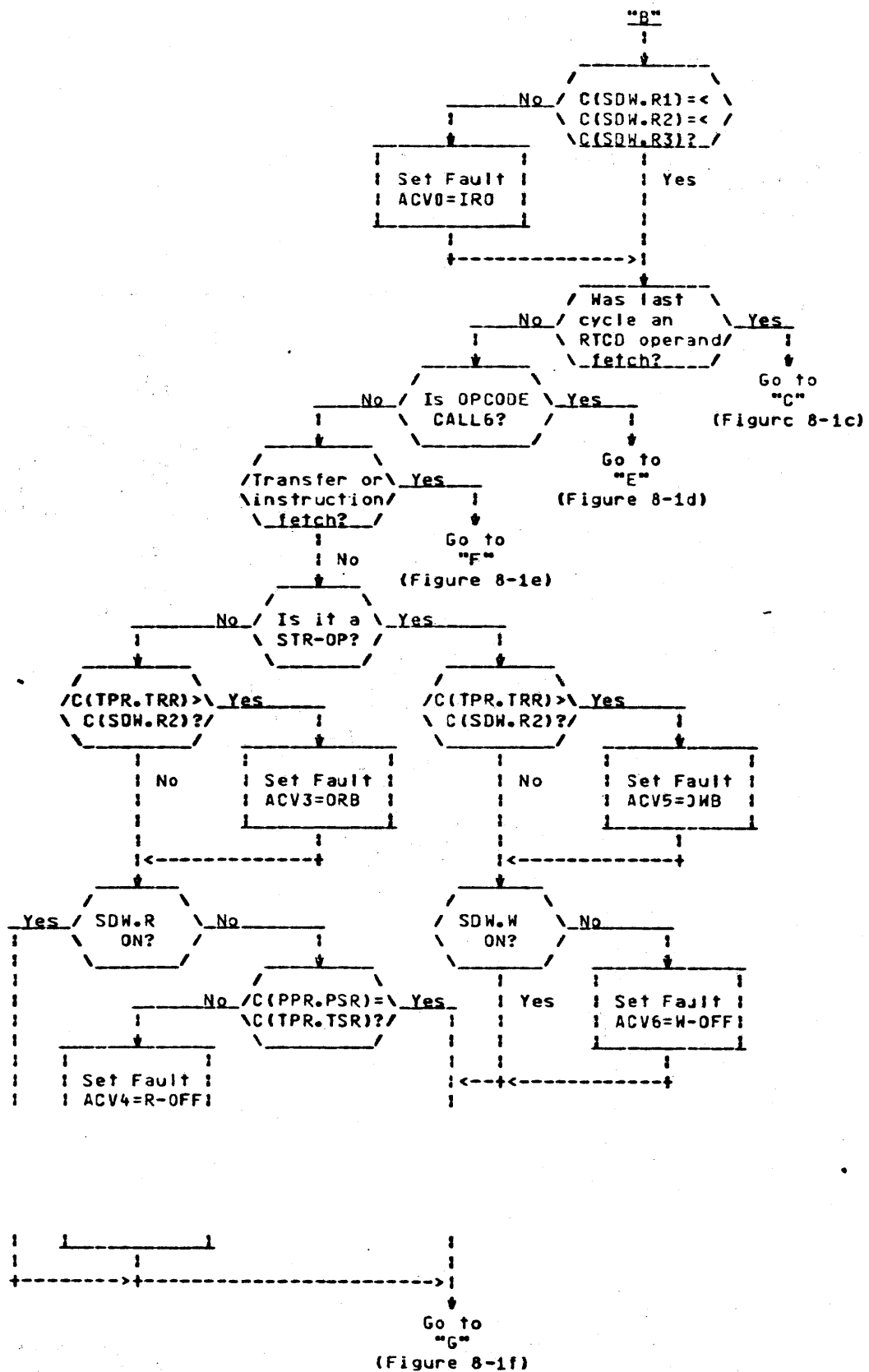


Figure 8-1b Complete Appending Unit Operation Flowchart (con't.)

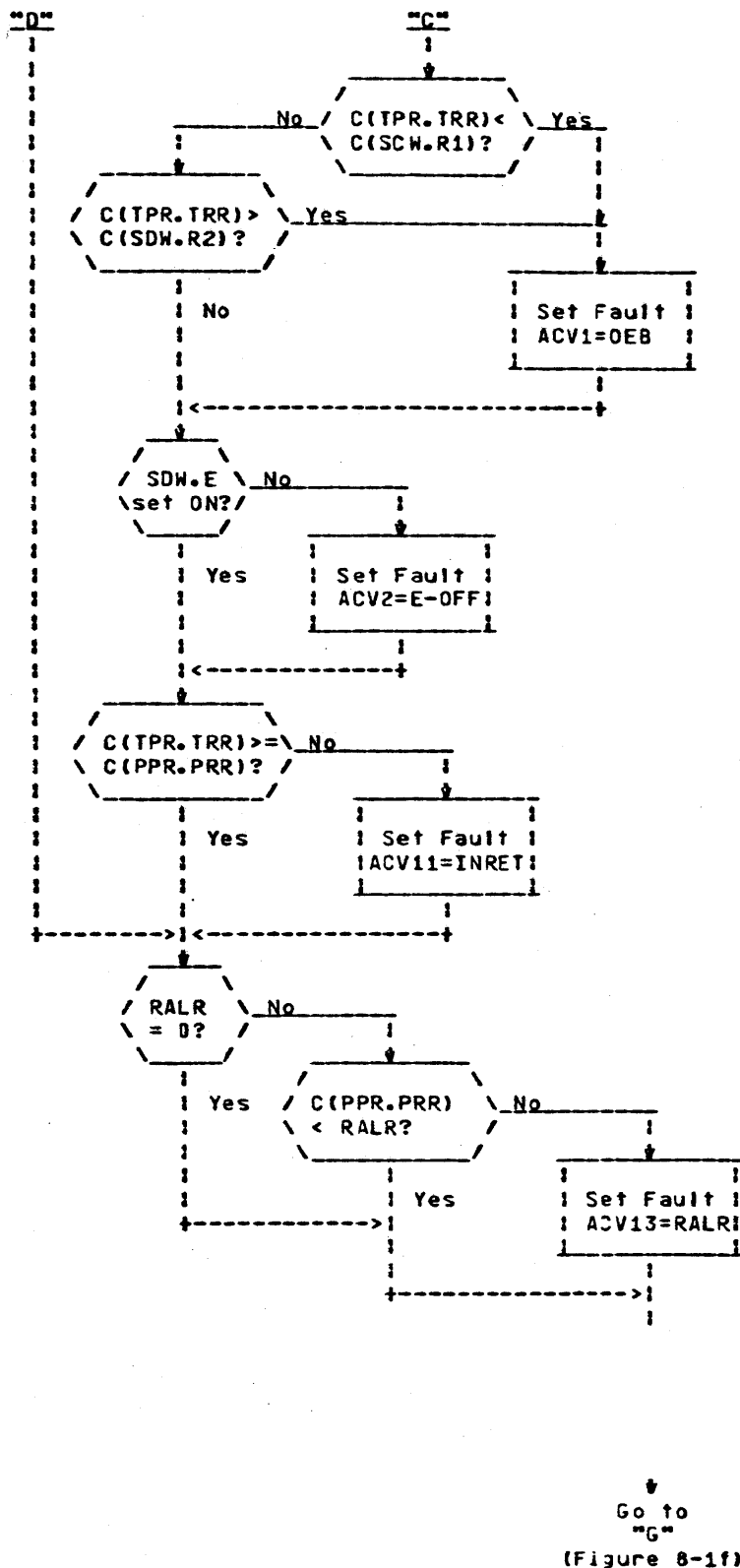


Figure 8-1c Complete Appending Unit Operation Flowchart (con't.)

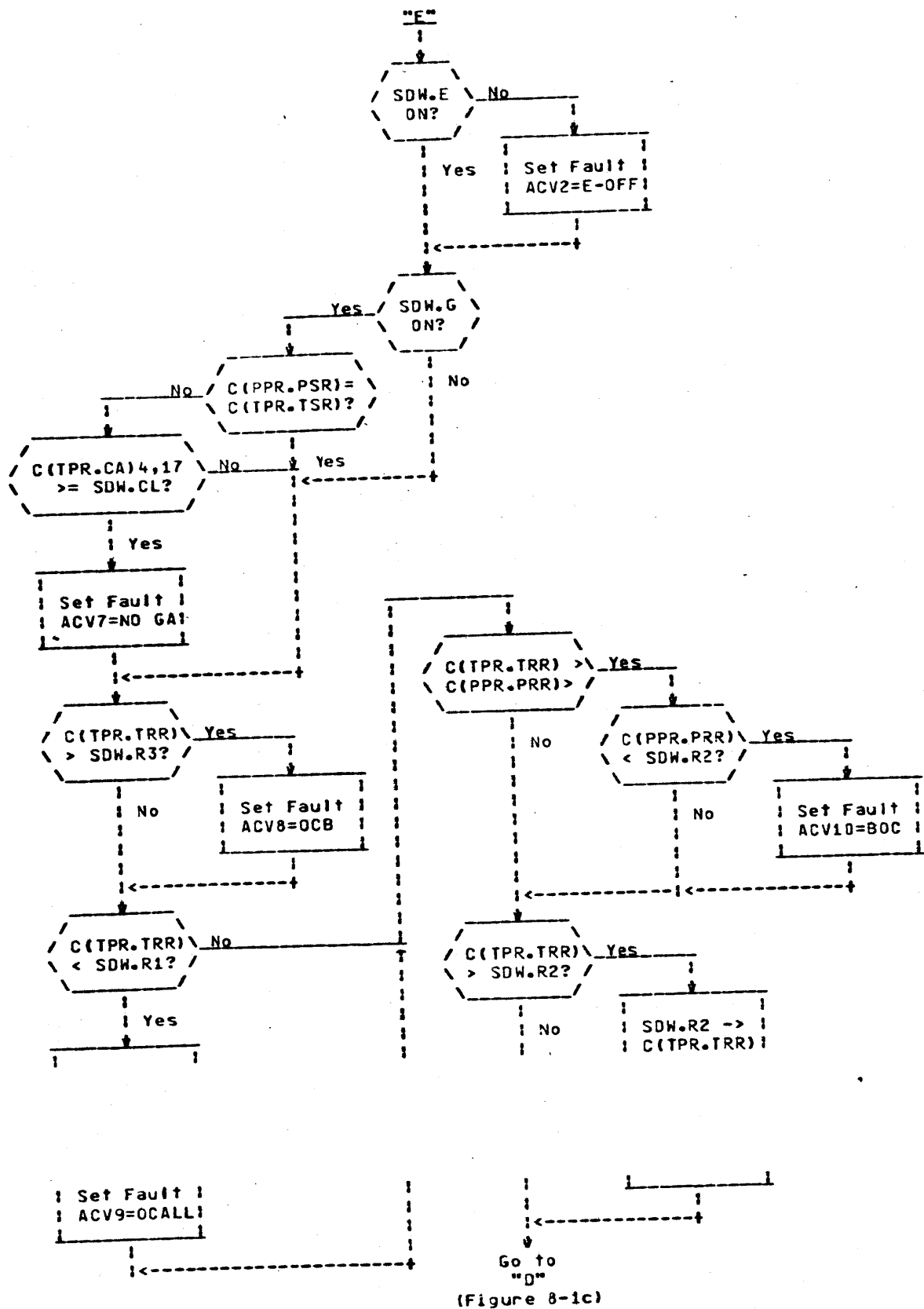


Figure 8-1d Complete Appending Unit Operation Flowchart (con't.)

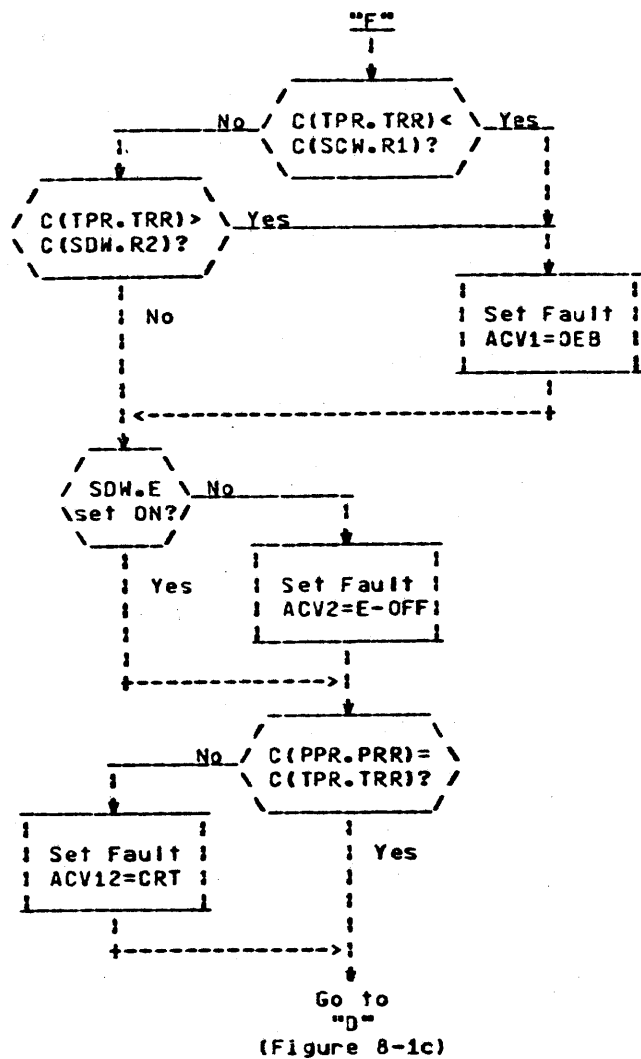
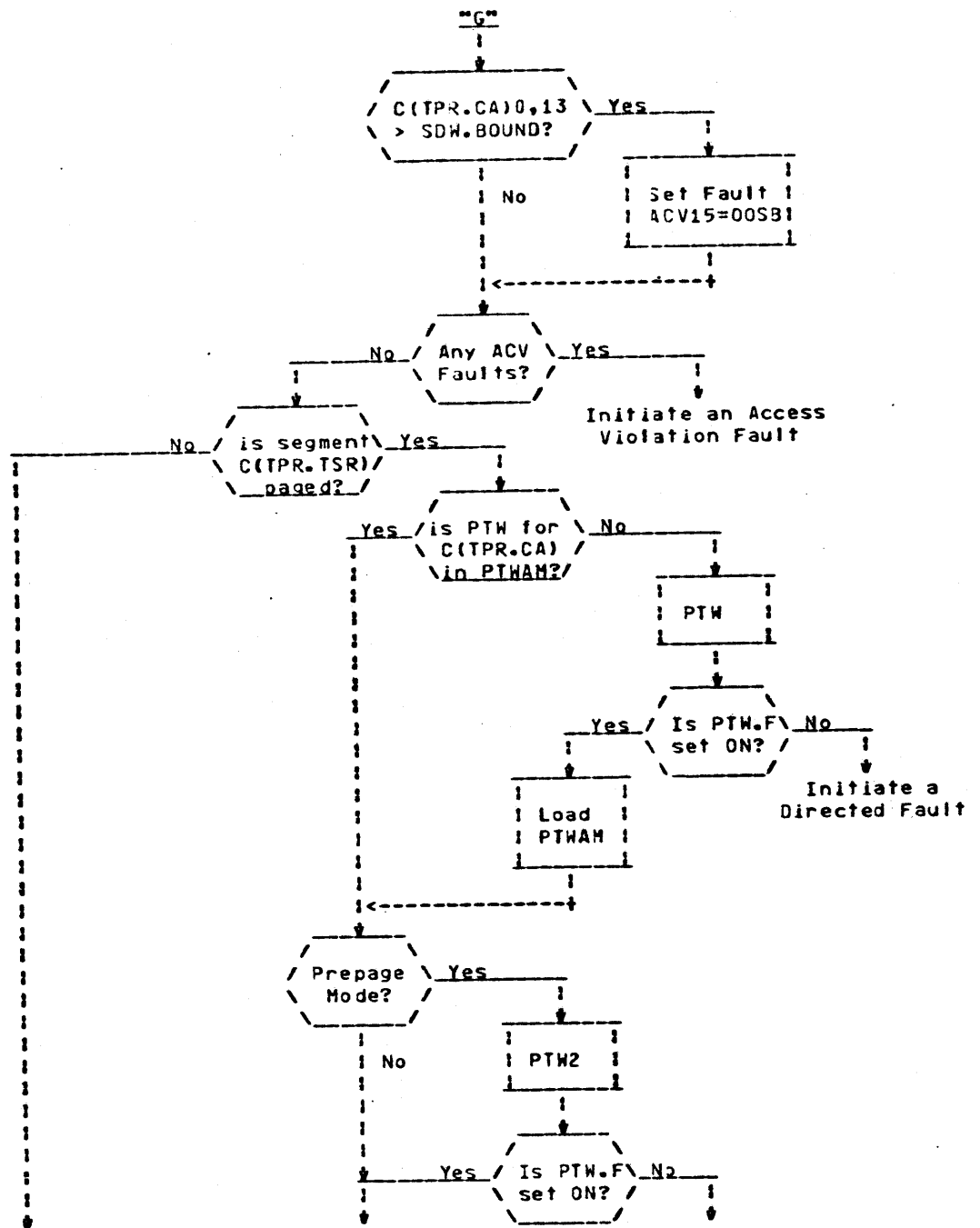


Figure 8-1e Complete Appending Unit Operation Flowchart (con't.)

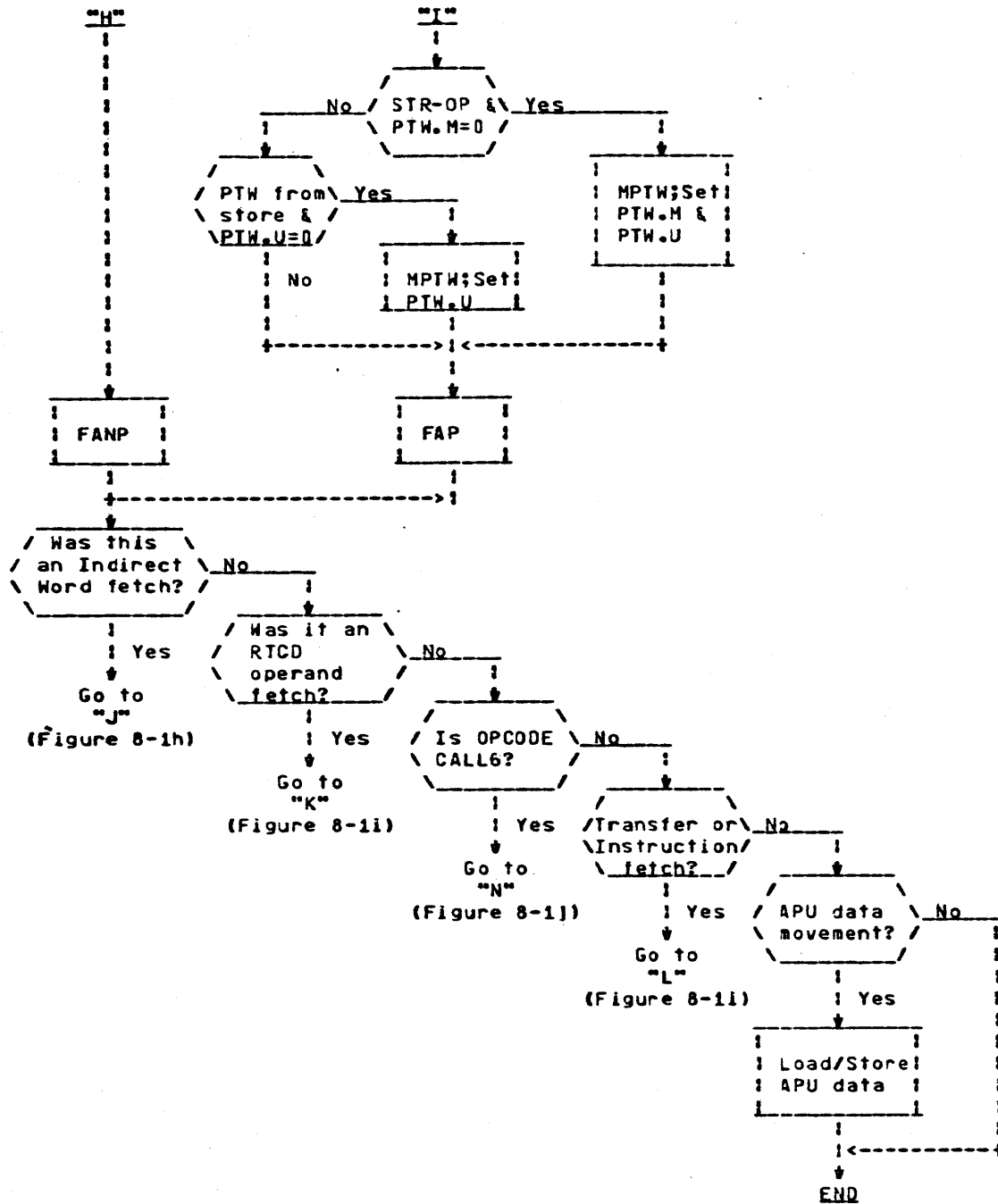


Go to
"H"
(Figure 8-1g)

Go to
"I"
(Figure 8-1g)

Initiate a
Directed Fault

Figure 8-1f Complete Appending Unit Operation Flowchart (con't.)



APPEND

Figure 8-1g Complete Appending Unit Operation Flowchart (con't.)

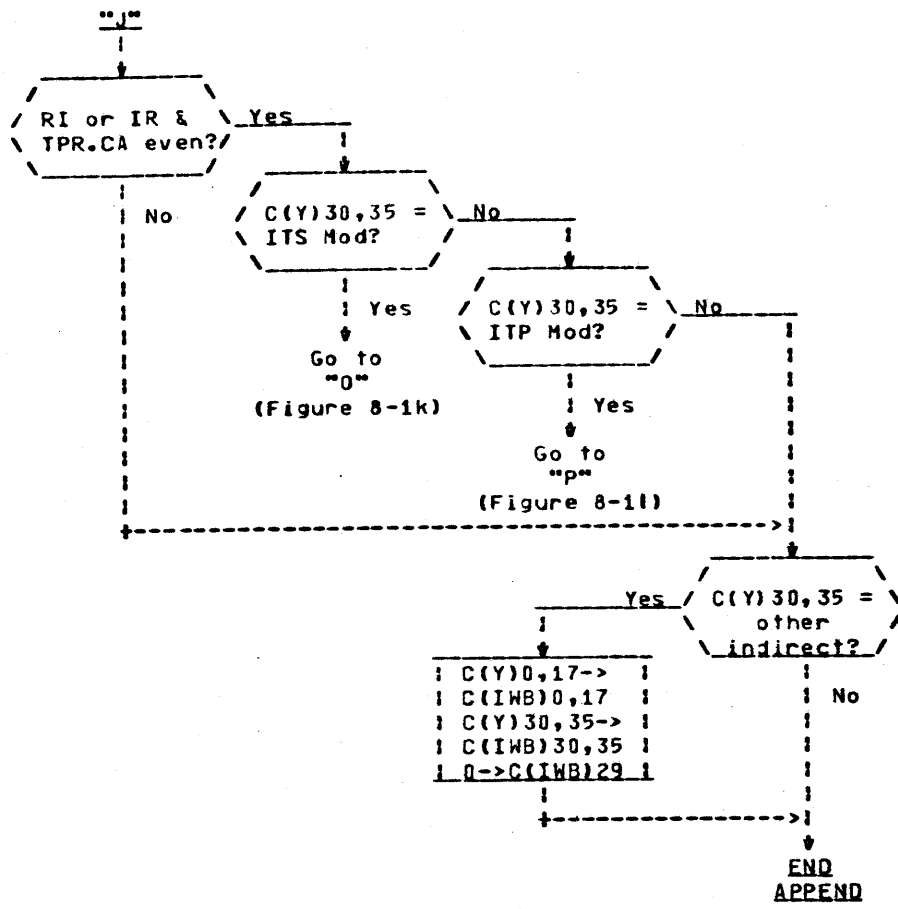


Figure 8-1h Complete Appending Unit Operation Flowchart (con't.)

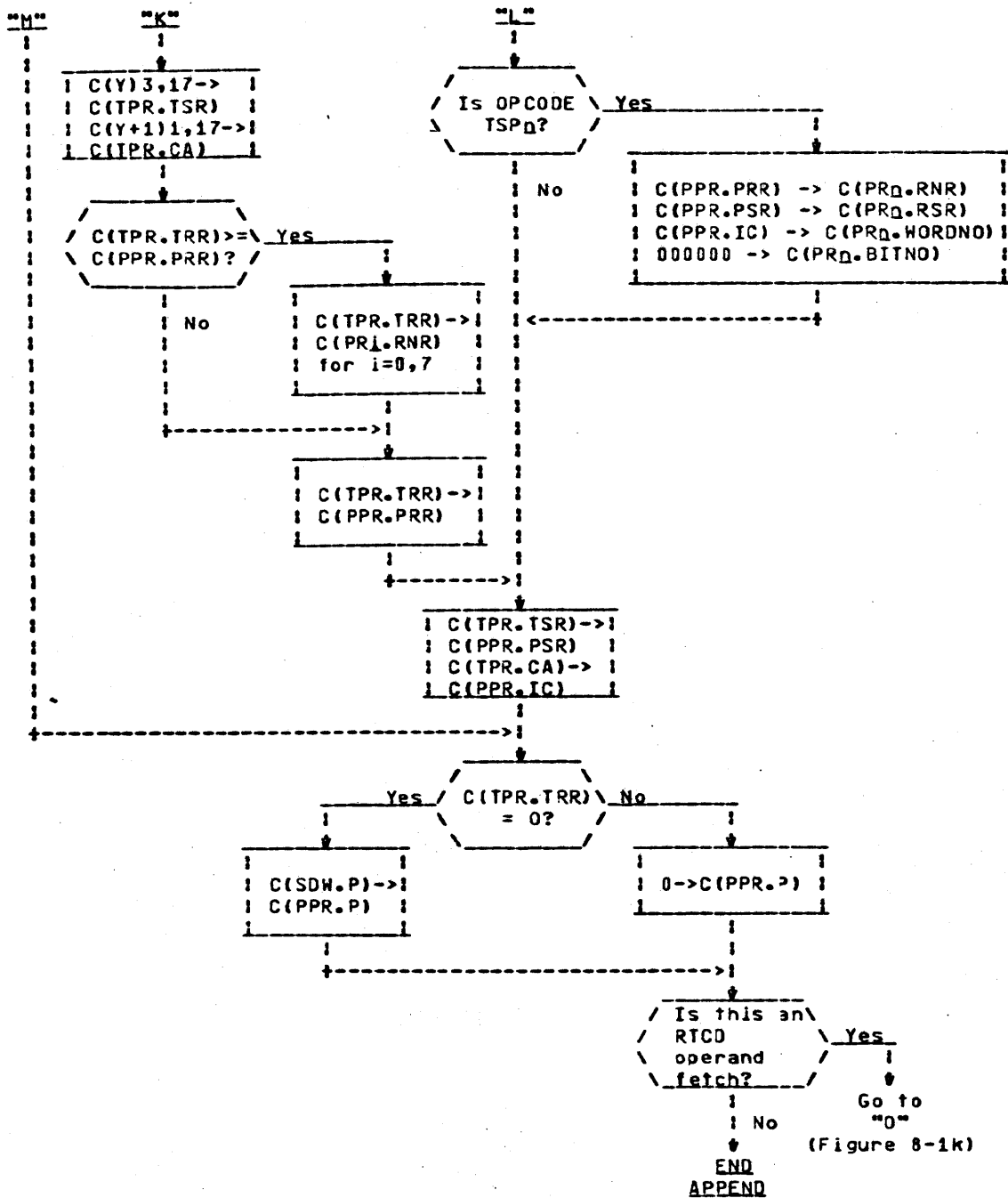


Figure 8-11 Complete Appending Unit Operation Flowchart (con't.)

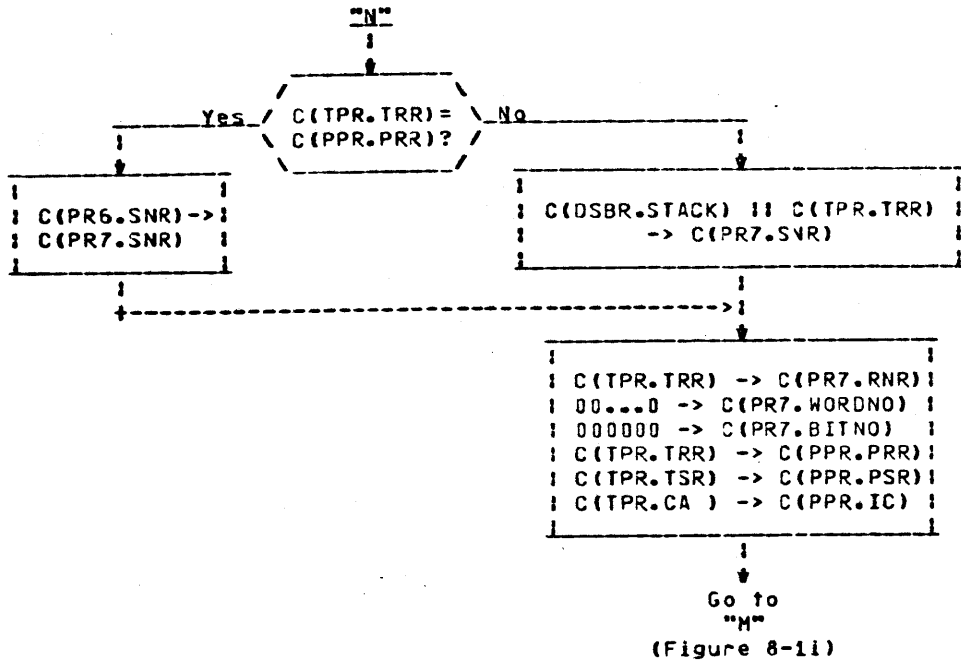


Figure 8-1] Complete Appending Unit Operation Flowchart (con't.)

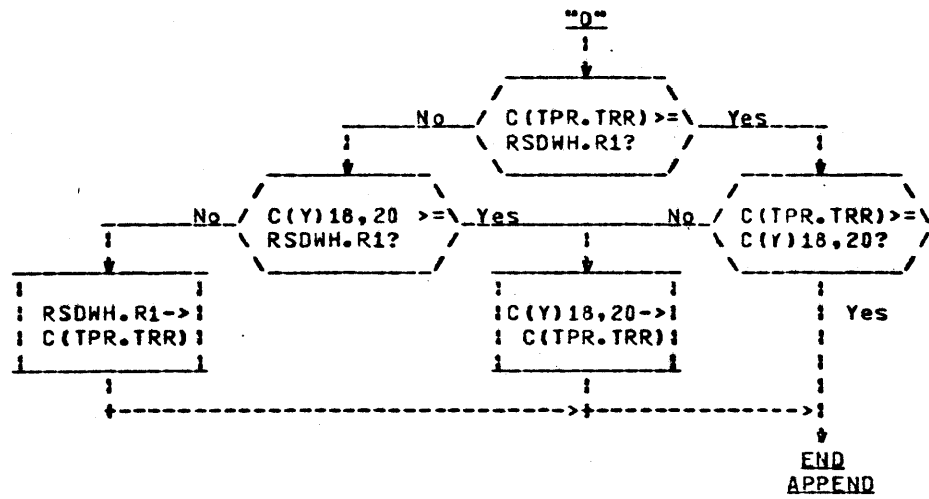


Figure 8-1k Complete Appending Unit Operation Flowchart (con't.)

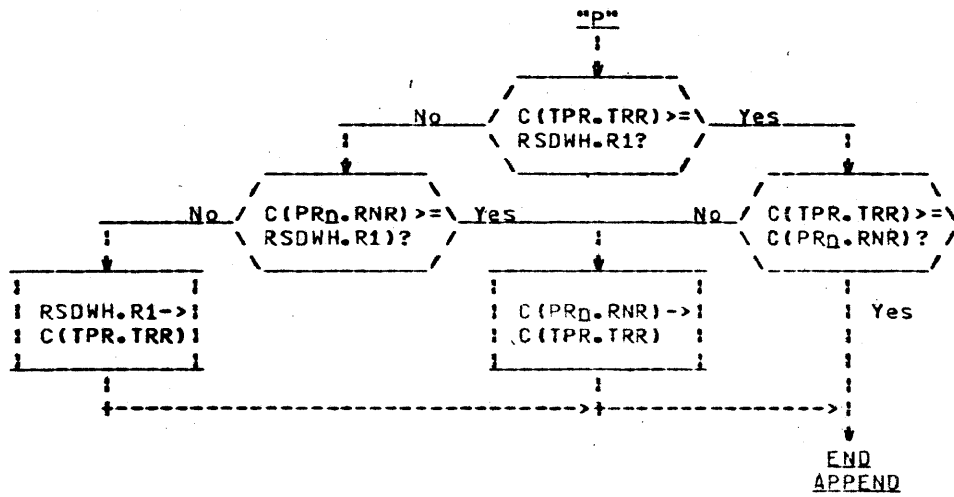


Figure 8-1k Complete Appending Unit Operation Flowchart (con't.)

SECTION IX

CACHE STORE OPERATION

The Multics processor may be fitted with an optional Cache Store. The operation of this Cache Store is described in this section.

PHILOSOPHY OF CACHE STORE

The Cache Store is a high speed buffer store located within the Processor that is intended to hold operands and/or instructions in expectation of their immediate use. This concept is different from that of holding a single operand (such as the Divisor for a Divide instruction) in the Processor during execution of a single instruction. A Cache Store depends on the Locality of Reference Principle. Locality of Reference involves the calculation of the probability, for any value of d , that the next instruction or operand reference after a reference to the instruction or operand at location A is to location $A+d$.

The calculation of probabilities for a set of values of d requires the statistical analysis of large masses of real and simulated instruction sequences and data organizations. If it can be shown that the average expected data/instruction access time reduction (over the range 1 to d) is statistically significant in comparison to the fixed Main Store access time, then the implementation of a Cache Store with block size d will contribute a significant improvement in performance.

The results of such studies for the Multics Processor with a Cache Store as described below show a hit probability ranging between 80% and 95% (depending on instruction mix and data organization) and a performance improvement ranging up to 30%.

CACHE STORE ORGANIZATION

The Cache Store is implemented as 2048 36-bit words of high speed register storage with associated control and content directory circuitry within the Processor. It is fully integrated with the normal data path circuitry and is virtually invisible

to all programming sequences. Parity is generated, stored, and checked just as in Main Store. The total storage is divided into 512 blocks of 4 words each and the blocks are organized into 128 "Columns" of four "Levels" each.

Cache Store/Main Store Mapping

Main Store is mapped into the Cache Store as described below and shown in Figure 9-1.

- Main Store is divided into N blocks of 4 words each arranged in ascending order and numbered with the value of Final Address bits 15 through 21 of the first word of the block.
- All Main Store blocks with numbers Ω modulo 128 are grouped associatively with Cache Store Column Ω .
- Each Cache Store Column may hold any four blocks of the associated set of Main Store blocks.
- Each Cache Store column has associated with it a four entry directory (one entry for each Level) and a two bit "round robin" counter. Parity is generated, stored, and checked on each directory entry.
- A Cache Directory entry consists of a fifteen bit ADDRESS register, a pre-set, two bit TAG or Level Number register and a Level Full flag bit.
- When a Main Store block is loaded into a Cache Store block at some Level in the associated Column, the Directory ADDRESS register for that Column and Level is loaded with the Final Address bits 0 through 14. (Level selection is discussed in Cache Store Control following.)

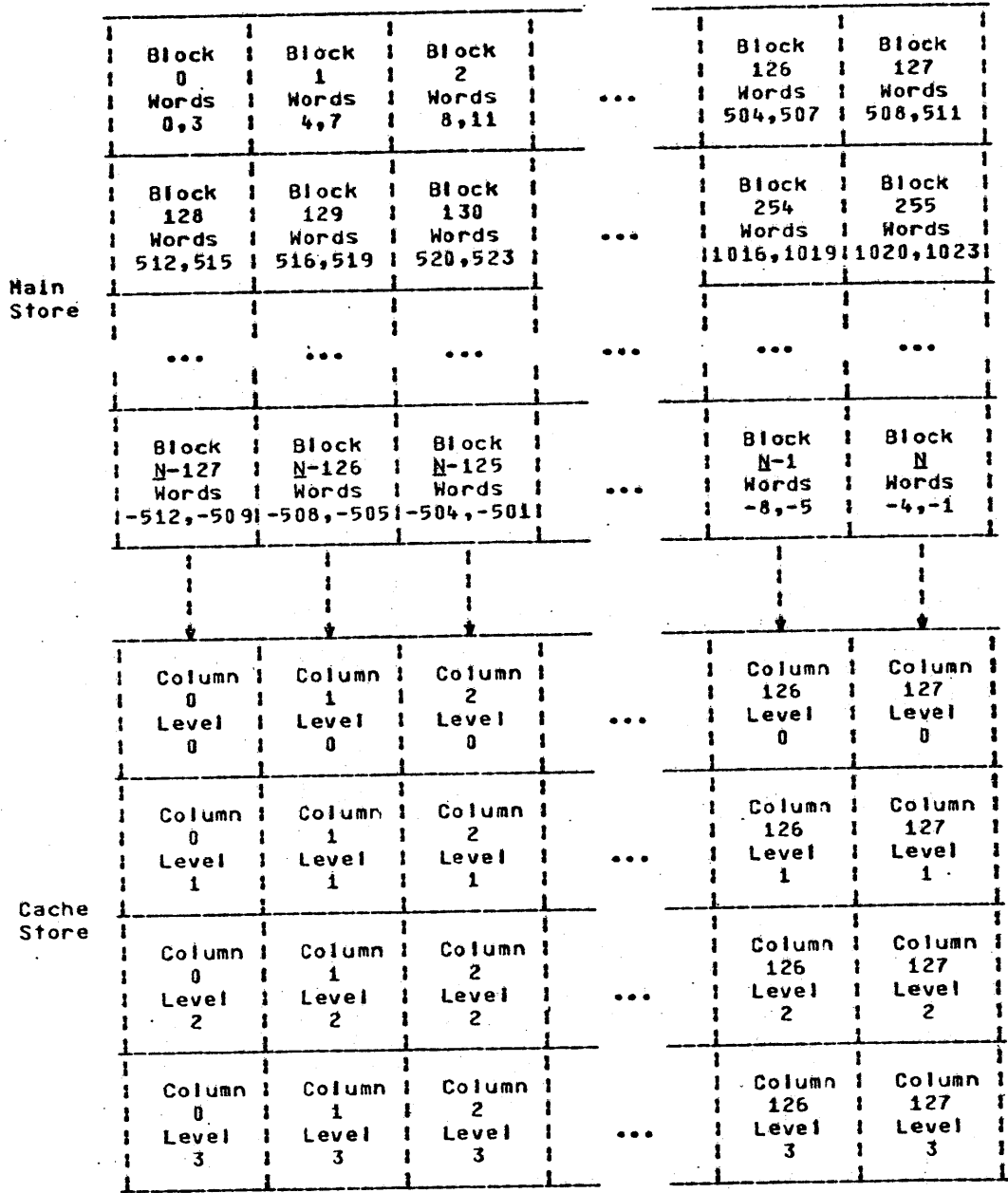


Figure 9-1 Main Store/Cache Store Mapping

Cache Store Addressing

For a read operation, the 24 bit Final Address prepared by the Appending Unit is presented simultaneously to the Cache Control and to the Main Store port selection circuitry. While port selection is being accomplished, the Cache Store is accessed as follows.

- Final Address bits 15 through 21 are used to select a Cache Store Column.
- Final Address bits 0 through 14 are matched associatively against the four Directory ADDRESS registers for the selected Column.
- If a match occurs for a Level whose Full flag is 0N, a hit is signalled, the Main Store reference cycle is cancelled, and the TAG register is read out.
- The TAG value and Final Address bits 22 and 23 are used to select the Level and Word in the selected Column and the Cache Store data is read out into the data circuitry.
- If no hit is signalled, the Main Store reference cycle proceeds and a Cache Store block load cycle is initiated (See Cache Store Control below).

For a write operation, the 24 bit Final Address prepared by the Appending Unit is presented simultaneously to the Cache Control and to the Main Store port selection circuitry. While port selection is being accomplished, the Cache Store is accessed as follows.

- Final Address bits 15 through 21 are used to select a Cache Store Column.
- Final Address bits 0 through 14 are matched associatively against the four Directory ADDRESS registers for the selected Column.
- If a match occurs for a Level whose Full flag is 0N, a hit is signalled and the TAG register is read out.
- The TAG value and Final Address bits 22 and 23 are used to select the Level and Word in the selected Column, a Cache Store write cycle is enabled, and the data is written to the Main Store and the Cache Store simultaneously.
- If no hit is signalled, the Main Store reference cycle proceeds with no further Cache Store action.

CACHE STORE CONTROL

Enabling and Disabling Cache Store

The Cache Store is controlled by the state of several bits in the Cache Mode Register (See Section IV, Program Accessible Registers, for a discussion of the Cache Mode Register). The Cache Mode Register may be loaded with the Load Central Processor Register (LCPR) instruction. The Cache Store control bits are as follows:

bit	Value	Action
18	0	The lower half of the Cache Store (Levels 0 and 1) is disabled and is totally inactive.
	1	The lower half of the Cache Store is active and enabled as per the state of bits 20 and 21.

19	0	The upper half of the Cache Store (Levels 2 and 3) is disabled and is totally inactive.
	1	The upper half of the Cache Store is active and enabled as per the state of bits 20 and 21.
20	0	The Cache Store (if active) <u>is not</u> used for Operands and Indirect Words.
	1	The Cache Store (if active) <u>is</u> used for Operands and Indirect Words.
21	0	The Cache Store (if active) <u>is not</u> used for Instructions.
	1	The Cache Store (if active) <u>is</u> used for Instructions.
23	0	The Cache-to-Register mode <u>is not</u> in effect (See "Dumping the Cache Store" following in this Section).
	1	The Cache-to-Register mode <u>is</u> in effect.

NOTE: The Cache Store option furnishes a switch panel maintenance aid that attaches to the free edge of the Cache Store Control Logic Board. The switch panel provides six switches for manual control of the Cache Store. Four of the switches inhibit the control functions of bits 18 through 21 of the Cache Mode Register and have the effect of forcing the corresponding function to be disabled. The fifth switch inhibits the "store-aside" feature wherein the Processor is permitted to proceed immediately after the Cache Store write cycle on write operations without waiting for a data acknowledgement from Main Store. (There is no software control corresponding this switch). The sixth switch forces the "enabled" condition on all Cache Store controls without regard to the corresponding Cache Mode Register control bit. There is no switch corresponding to the Cache-to-Register control bit.

While these switches are intended primarily for Maintenance sessions, they have been found useful in testing the Cache Store during normal operation and in permitting operation of the Processor with the Cache Store in degraded or partially disabled mode.

Cache Store Control in Segment Descriptor Words

Certain data have characteristics such that they should never be loaded into the Cache Store. Primary examples of such data are hardware mailboxes for the I/O Multiplexer, Bulk Store Controller, etc., status return words, and various dynamic system data base segments such as the System Segment Table and shared Directory Segments. In general, any data that is purposely modified by an agency external to the Processor with the intent to convey information to the Processor should never be loaded into Cache Store.

Bit 57 of the Segment Descriptor Word is used to reflect this property of "encachability" for each segment (See Section V, Addressing -- Segmentation and Paging, for a discussion of the Segment Descriptor Word). If the bit is set ON, data from the segment may be loaded into the Cache Store; if the bit is OFF, they may not.

The encachability property may be treated as permanent (e.g., for hardware mailboxes) or dynamic (e.g., certain shared data bases) by the operating system. The operating system sets bit 57 ON or OFF as appropriate for the function to be performed on the segment.

Loading the Cache Store

The Cache Store is loaded with data implicitly whenever a Cache Store Block Load is signalled (See the discussion of read operations in "Cache Store Addressing" above in this section). There is no explicit method or instruction to load data into the Cache Store.

When a Cache Store Block Load is signalled, the Level is selected from the value of the Round Robin Counter for the selected Column, and the Cache Store Write function is enabled. (The Round Robin Counter contains the number of the least recently loaded Level.) When the data arrives from Main Store, it is written into the Cache Store and entered into the data circuitry. The Processor proceeds with the execution of the instruction requiring the operand if appropriate.

When the Cache Store Write is complete, further Address Preparation is inhibited, bit 22 of the Final Address is inverted, and a second Main Store access for the other half of the block is made. When the second half data arrives from Main Store, it is written into the Cache Store, Final Address bits 0 through 14 are loaded into the Directory ADDRESS Register, the Level Full flag is set ON, the Round Robin Counter is advanced by 1, and Address Preparation is permitted to proceed.

If all four Level Full flags for a Column are set ON, a Column Full flag is also set ON and remains ON until one or more Levels in the Column are cleared.

Clearing the Cache Store

Cache Store can be cleared in two ways; General Clear and Selective Clear. The clearing action is the same in both cases, namely, the Full flags of the selected Column(s) and/or Level(s) are set OFF.

GENERAL CLEAR

The entire Cache Store is cleared by setting all Column and Level Full flags to OFF in the following situations:

- Upper or lower Cache Store or both becoming enabled by appropriate bits in the operand of a Load Central Processor Register (LCPR) instruction, or by action of the Logic Board free edge switches
- Execution of a Clear Associative Memory Segments (CAMS) instruction with bit 15 of the address field 0

SELECTIVE CLEAR

The Cache Store is cleared selectively as follows:

- If a Read-and-Clear operation (LDAC, SZNC, etc.) results in a hit on the Cache Store, the Cache Store block hit is cleared.
- Execution of a Clear Associative Memory Pages (CAMP) with address bit 15 set ON causes Final Address bits 0 through 14 to be matched against all Cache Directory ADDRESS Registers. All Cache Store blocks hit are cleared.

Dumping the Cache Store

When the Cache-to-Register mode flag (bit 24 of the Cache Mode Register) is set ON, the Processor is forced to fetch the operands of all Double Precision Operations Unit Load operations from

the Cache Store.
 Final Address bits 0 through 14 are ignored, Final Address bits 15 through 21
 select a Column, and Final address bits 22 and 23 select a Level.
 All other operations (e.g., Instruction Fetches, Single Precision Operands, etc.) are
 treated normally.

WARNING: Note that the phrase "treated normally" as used here includes the case where
 the Cache Store is enabled. If the Cache Store is enabled, the "other" operations will
 cause normal Block Loads and Cache Store Writes thus destroying the original contents
 of the Cache Store.

The user is warned that the Cache Store should be disabled before dumping is attempted.

An indexed program loop involving the LDAQ and STAQ instructions with the Cache-to-Register mode bit set ON
 will serve to dump any or all of the Cache Store.

Note: If a Fault or Program Interrupt should occur during the execution
 of a Cache Store dumping loop, the Cache-to-Register mode bit would seriously interfere with normal address
 in the servicing of such Fault or Interrupt. Hence, the Cache-to-Register mode bit is reset
 automatically by any Fault or Program Interrupt.

APPENDIX A

OPERATION CODE MAP (BIT 27 = 0)

	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017
000	I	IMME	IDRL	I	IMME2	IMME3	I	IMME4	I	INOP	IPULS1	IPULS2	I	ICIOC	I	I
020	IADLX0	IADLX1	IADLX2	IADLX3	IADLX4	IADLX5	IADLX6	IADLX7	I	ILDQC	IADL	ILDAC	IADLA	IADLQ	IADLAQ	I
040	IASX0	IASX1	IASX2	IASX3	IASX4	IASX5	IASX6	IASX7	IADWP0	IADWP1	IADWP2	IADWP3	IAOS	IASA	IASQ	ISSCR
060	IADX0	IADX1	IADX2	IADX3	IADX4	IADX5	IADX6	IADX7	I	IADCA	IADCO	ILREG	I	IADJA	IADQ	IADAQ

100	ICMPX0	ICMPX1	ICMPX2	ICMPX3	ICMPX4	ICMPX5	ICMPX6	ICMPX7	ICHL				ICMPA	ICMPQ	ICMPAQ
120	ISBLX0	ISBLX1	ISBLX2	ISBLX3	ISBLX4	ISBLX5	ISBLX6	ISBLX7					IS3LA	ISBLQ	ISBLAQ
140	ISSX0	ISSX1	ISSX2	ISSX3	ISSX4	ISSX5	ISSX6	ISSX7	ADWP4	ADWP5	ADWP6	ADWP7	SOBR	ISSA	ISSQ
160	ISBX0	ISBX1	ISBX2	ISBX3	ISBX4	ISBX5	ISBX6	ISBX7	ISCA	SHCQ	ILPRI		IS3A	ISBQ	ISBAQ
200	ICNAX0	ICNAX1	ICNAX2	ICNAX3	ICNAX4	ICNAX5	ICNAX6	ICNAX7	ICMK	ABSA	IEPAQ	ISZNC	ICNAA	ICNAQ	ICNAAQ
220	ILDXX0	ILDXX1	ILDXX2	ILDXX3	ILDXX4	ILDXX5	ILDXX6	ILDXX7	ILBAR	IRSW	ILDBR	IRMCM	ISZN	ILJA	ILDQ
240	IORSX0	IORSX1	IORSX2	IORSX3	IORSX4	IORSX5	IORSX6	IORSX7	ISPBP0	ISPRI1	ISPBP2	ISPRI3	ISPRI	IORSA	IORSQ
260	IORX0	IORX1	IORX2	IORX3	IORX4	IORX5	IORX6	IORX7	ITSP0	ITSP1	ITSP2	ITSP3		IORA	IORQ
300	ICANX0	ICANX1	ICANX2	ICANX3	ICANX4	ICANX5	ICANX6	ICANX7	IEAWP0	IEASP0	IEAWP2	IEASP2		ICANA	ICANQ
320	ILCX0	ILCX1	ILCX2	ILCX3	ILCX4	ILCX5	ILCX6	ILCX7	IEAWP4	IEASP4	IEAWP6	IEASP6		ILCA	ILCQ
340	IANSX0	IANSX1	IANSX2	IANSX3	IANSX4	IANSX5	IANSX6	IANSX7	IEPP0	IEBP1	IEPP2	IEBP3	STAC	IANSA	IANSQ
360	IANX0	IANX1	IANX2	IANX3	IANX4	IANX5	IANX6	IANX7	IEPP4	IEBP5	IEPP6	IEBP7		IANA	IANQ
400		IMPF	IMPY			ICMG				ILDE		IRSCR		IADE	
420		IUFM		IDUFM		IFCMG		IDFCMG	IFSZN	IFLD		IDFLD		IUFA	IDUFA
440	ISXL0	ISXL1	ISXL2	ISXL3	ISXL4	ISXL5	ISXL6	ISXL7	ISTZ	ISMIC	ISCPR		ISTT	IFST	ISTE
460		IEMP		IDEMP					IFSTR	IF3D		IDFSTR	IDFRD	IFAD	IDFA
500	IRPL				IBCO	IDIV	IDVF					IFNEG		IFCMP	IDFC
520	IRPT				IFDI		IDFDI		INEG	ICAMS	INEGL		IUFS		IDUFS
540	ISPRP0	ISPRP1	ISPRP2	ISPRP3	ISPRP4	ISPRP5	ISPRP6	ISPRP7	ISBAR	ISTBA	ISTBQ	ISMCM	ISTC1		ISSDP
560	IRPD				IFDV		IDFDV					IFNO		IFSB	IDFSB
600	ITZE	ITNZ	ITNC	ITRC	ITMI	ITPL		ITTF	IRTC0			IRCU	ITEO	ITEU	IDIS
620	IEAX0	IEAX1	IEAX2	IEAX3	IEAX4	IEAX5	IEAX6	IEAX7	IRET			IRCCL	ILDI	IEAA	IEAQ
640	IERSX0	IERSX1	IERSX2	IERSX3	IERSX4	IERSX5	IERSX6	IERSX7	ISPRI4	ISPBP5	ISPRI6	ISPBP7	STACQ	IERSA	IERSQ
660	IEXX0	IEXX1	IEXX2	IEXX3	IEXX4	IEXX5	IEXX6	IEXX7	ITSP4	ITSP5	ITSP6	ITSP7	ILCPR	IEXA	IEXQ
700	ITSX0	ITSX1	ITSX2	ITSX3	ITSX4	ITSX5	ITSX6	ITSX7	ITRA			ICALL6		ITSS	IXEC
720	ILXL0	ILXL1	ILXL2	ILXL3	ILXL4	ILXL5	ILXL6	ILXL7		IARS	IQRS	ILRS		IALS	IQLS
740	ISTX0	ISTX1	ISTX2	ISTX3	ISTX4	ISTX5	ISTX6	ISTX7	ISTC2	ISTCA	ISTCQ	ISREG	ISTI	ISTA	ISTQ
760	ILPRP0	ILPRP1	ILPRP2	ILPRP3	ILPRP4	ILPRP5	ILPRP6	ILPRP7	IARL	IQRL	ILRL	IGTB	IAR	IQLR	ILLR

OPERATION CODE MAP (BIT 27 = 0)

	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017
000																
020	MVE				MVNE											
040																
060	CSL	CSR			SZTL	SZTR	CMPB									
100	MLR	MRL					CMPC									
120	SCD	SCDR			SCM	SCMR										
140													SPTR			
160	MVT				TCT	TCTR						LPTR				
200			A02D	SB2D			MP2D	DV2D								
220			A03D	SB3D			MP3D	DV3D				LSDR				
240																
260										SPBP0	SPRI1	SPBP2	SPRI3	SSDR		LPTP
300	MVN	BTD			CMPN		DTB			EASP1	EAWP1	EASP3	EAWP3			
320										EASP5	EAWP5	EASP7	EAWP7			
340										EPBP0	EPP1	EPBP2	EPP3			
360										EPBP4	EPP5	EPBP6	EPP7			
400																
420																
440					SAREG					SPL						
460					LAREG					LPL						
500	A9BD	A6BD	A4BD	ABD					AWD							
520	S9BD	S6BD	S4BD	SBD					SWD			CAMP				
540	ARA0	ARA1	ARA2	ARA3	ARA4	ARA5	ARA6	ARA7								SPTP
560	AAR0	AAR1	AAR2	AAR3	AAR4	AAR5	AAR6	AAR7								
600	TRTN	TRTF			TMOZ	TPNZ	TTN									
620																
640	ARN0	ARN1	ARN2	ARN3	ARN4	ARN5	ARN6	ARN7		SPBP4	SPRI5	SPBP6	SPRI7			
660	NAR0	NAR1	NAR2	NAR3	NAR4	NAR5	NAR6	NAR7								
700																
720																
740	SAR0	SAR1	SAR2	SAR3	SAR4	SAR5	SAR6	SAR7					SRA			
760	LAR0	LAR1	LAR2	LAR3	LAR4	LAR5	LAR6	LAR7					LRA			

APPENDIX B

ALPHABETIC OPERATION CODE LIST

This appendix presents a list of all Processor instruction operation codes sorted on mnemonic and giving the octal operation code value, the instruction name, and the functional category.

The function category codes are as follows:

FXD	Fixed Point
BOOL	Boolean Operations
FLTG	Floating Point
PREG	Pointer Register
PRIV	Privileged
MISC	Miscellaneous
EIS	Extended Instruction Set
TXFR	Transfer of Control

Mnemonic	Code	Category	Name
A4BD	502(1)	EIS	Add 4-bit Character Displacement to AR
A6BD	501(1)	EIS	Add 6-bit Character Displacement to AR
A9BD	500(1)	EIS	Add 9-bit Character Displacement to AR
AAR _n	56 _n (1)	EIS	Alphanumeric Descriptor to AR _n
ABD	503(1)	EIS	Add Bit Displacement to AR
ABSA	212(0)	PRIV	Absolute Address to A-Register
AD2D	202(1)	EIS	Add Using 2 Decimal Operands
AD3D	222(1)	EIS	Add Using 3 Decimal Operands
ADA	075(0)	FXD	Add to A-Register
ADAQ	077(0)	FXD	Add to AQ-Register
ADE	415(0)	FLTG	Add to E-Register
ADL	033(0)	FXD	Add Low to AQ-Register
ADLA	035(0)	FXD	Add Logical to A-Register
ADLAQ	037(0)	FXD	Add Logical to AQ-Register
ADLQ	036(0)	FXD	Add Logical to Q-Register
ADLX _n	02 _n (0)	FXD	Add Logical to Index _n
ADQ	076(0)	FXD	Add to Q-Register
ADWP0	050(0)	PREG	Add to Word Number Field of PR0
ADWP1	051(0)	PREG	Add to Word Number Field of PR1
ADWP2	052(0)	PREG	Add to Word Number Field of PR2
ADWP3	053(0)	PREG	Add to Word Number Field of PR3
ADWP4	150(0)	PREG	Add to Word Number Field of PR4
ADWP5	151(0)	PREG	Add to Word Number Field of PR5
ADWP6	152(0)	PREG	Add to Word Number Field of PR6
ADWP7	163(0)	PREG	Add to Word Number Field of PR7

ADXN	86N(0)	FXD	Add to Index N
ALR	775(0)	FXD	A-Register Left Rotate
ALS	735(0)	FXD	A-Register Left Shift
ANA	375(0)	BOOL	AND to A-Register
ANAQ	377(0)	BOOL	AND to AQ-Register
ANQ	376(0)	BOOL	AND to Q-Register
ANSA	355(0)	BOOL	AND to Storage from A-Register
ANSQ	356(0)	BOOL	AND to Storage from Q-Register
ANSXD	34D(0)	BOOL	AND to Storage from Index D
ANXD	36D(0)	BOOL	AND to Index D
AOS	054(0)	FXD	Add One to Storage
ARA _D	54D(1)	EIS	AR _D to Alphanumeric Descriptor
ARL	771(0)	FXD	A-Register Right Logical Shift
ARN _D	64D(1)	EIS	AR _D to Numeric Descriptor
ARS	731(0)	FXD	A-Register Right Shift
ASA	055(0)	FXD	Add Stored to A-Register
ASQ	056(0)	FXD	Add Stored to Q-Register
ASXD	04D(0)	FXD	Add Stored to Index D
AWCA	071(0)	FXD	Add With Carry to A-Register
AWCQ	072(0)	FXD	Add With Carry to Q-Register
AWD	507(1)	EIS	Add Word Displacement to AR
BCD	505(0)	MISC	Binary-to-BCD
BTD	301(1)	EIS	Binary-to-Decimal
CALL6	713(0)	TXFR	Call
CAMP	532(1)	PRIV	Clear Associative Memory Pages
CAMS	532(0)	PRIV	Clear Associative Memory Segments
CANA	315(0)	BOOL	Comparative AND with A-Register
CANAQ	317(0)	BOOL	Comparative AND with AQ-Register
CANQ	316(0)	BOOL	Comparative AND with Q-Register
CANXD	30D(0)	BOOL	Comparative AND with Index D
CIOC	015(0)	PRIV	Connect
CMG	405(0)	FXD	Compare Magnitude
CHK	211(0)	FXD	Compare Masked
CMPA	115(0)	FXD	Compare with A-Register
CMPAQ	117(0)	FXD	Compare with AQ-Register
CMPB	066(1)	EIS	Compare Bit Strings
CMPC	106(1)	EIS	Compare Alphanumeric Character Strings
CMPN	303(1)	EIS	Compare Numeric
CMPQ	116(0)	FXD	Compare with Q-Register
CMPXD	10D(0)	FXD	Compare with Index D
CNAA	215(0)	BOOL	Comparative NOT with A-Register
CNAAQ	217(0)	BOOL	Comparative NOT with AQ-Register
CNAQ	216(0)	BOOL	Comparative NOT with Q-Register
CNAXD	20D(0)	BOOL	Comparative NOT with Index D
CSL	060(1)	EIS	Combine Bit Strings Left
CSR	060(1)	EIS	Combine Bit Strings Right
CWL	111(0)	FXD	Compare With Limits
DFAD	477(0)	FLTG	Double Precision Floating Add
DFCMG	427(0)	FLTG	Double Precision Floating Compare Magnitude
DFCMP	517(0)	FLTG	Double Precision Floating Compare
DFDI	527(0)	FLTG	Double Precision Floating Divide Inverted
DFDV	567(0)	FLTG	Double Precision Floating Divide
DFLD	433(0)	FLTG	Double Precision Floating Load
DFMP	463(0)	FLTG	Double precision Floating Multiply
DFRD	473(0)	FLTG	Double Precision Floating Round

DFSB	577(0)	FLTG	Double Precision Floating Subtract
DFST	457(0)	FLTG	Double Precision Floating Store
DFSTR	472(0)	FLTG	Double Precision Floating Store Rounded
DIS	616(0)	PRIV	Delay Until Interrupt Signal
DIV	506(0)	FLTG	Divide Integer
DRL	002(0)	MISC	Derail
DTB	305(1)	EIS	Decimal-to-Binary Convert
DUFA	437(0)	FLTG	Double Precision Unnormalized Floating Add
DUFM	423(0)	FLTG	Double Precision Unnormalized Floating Multiply
DUFS	537(0)	FLTG	Double Precision Unnormalized Floating Subtract
DV2D	207(1)	EIS	Divide Using 2 Decimal Operands
DV3D	227(1)	EIS	Divide Using 3 Decimal operands
DVF	507(0)	FXD	Divide Fraction
EAA	635(0)	FXD	Effective Address to A-Register
EAQ	636(0)	FXD	Effective Address to Q-Register
EASP0	311(0)	PREG	Effective Address to Segment Number Field of PR0
EASP1	310(1)	PREG	Effective Address to Segment Number Field of PR1
EASP2	313(0)	PREG	Effective Address to Segment Number Field of PR2
EASP3	312(1)	PREG	Effective Address to Segment Number Field of PR3
EASP4	331(0)	PREG	Effective Address to Segment Number Field of PR4
EASP5	330(1)	PREG	Effective Address to Segment Number Field of PR5
EASP6	333(0)	PREG	Effective Address to Segment Number Field

The Other Computer Company:
Honeywell

HONEYWELL INFORMATION SYSTEMS