Honeywell

MODEL 645 PROCESSOR REFERENCE MANUAL

RESTRICTED DISTRIBUTION

SUBJECT:

Programming Information for the Model 645 Processor, Including Machine Instructions, Data Representation, Registers, and Addressing.

SPECIAL INSTRUCTIONS:

This manual supersedes document G0098, dated August, 1970, and its revisions 1 through 4, dated September, 1970, October, 1970, February, 1971, and March, 1971.

DATE:

May, 1972

ORDER NUMBER:

AH82, Rev. 0

PREFACE

This publication describes the Model 645 processor, a modified version of the 635 processor. It is assumed that the reader is familiar with the overall modular organization of the 635 and 645 systems and with the philosophy of the asynchronous operation of these modules.

Emphasis has been placed on those Model 645 features which augment the function of the 635. However, the entire repertoire of Model 645 instructions is explained. In addition, this manual presents a thorough discussion of virtual memory addressing concepts including segmentation and paging.

The manual is intended primarily 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 deal with machine instructions — particularly language processor implementors.

Although this manual makes occasional references to Multics (Multiplexed Information and Computing Service) and tends to emphasize the use of the 645's features in conjunction with the Multics software, the information is generally applicable to any Model 645.

TABLE OF CONTENTS

TABLE OF CONTENTS, Continued

TABLE OF CONTENTS, Continued

TABLE OF CONTENTS, Continued

CHAPTER 1 INTRODUCTION TO PROCESSOR

FEATURES OF THE 645 PROCESSOR

The 645 Processor was designed for use with the Multiplexed Information and Computing Service (Multics) and contains, in addition to the standard 635 processor features, a number of special features that support Multics. The addressing mechanisms, in particular, are designed to permit the software to compute relative and absolute addresses, locate data and programs on different devices and retrieve such data and programs as necessary. Chapters 5, 6, and 7 describe the special features of the 645 including segmentation and paging; address modification and address appending; and faults and interrupts. These features are closely related and each is described briefly in the paragraphs that follow.

Segmentation and Paging

A segment is merely a collection of data or instructions that is assigned a symbolic name by the programmer and addressed symbolically by him. Paging is at the discretion of the software; the user may not be aware of the existence of pages. When a segment is paged, all of the pages are the same size. Segments and their pages are addressed by a segment number and a segment address.

The user may view each of his segments as if it were stored in an independent memory unit. Each segment has its own origin which can be addressed as location zero. The size of each segment may vary without affecting the addressing of the other segments. Each segment can be addressed like a conventional core image starting at location zero. Maximum segment size is 2^{18} words; however, the current Multics implementation restricts the size to 2^{16} words.

When viewed from the processor, memory consists of blocks of 64 or 1024 words. Each block begins at an absolute address which can be either 0 modulo 64 or 0 modulo 1024. A segment can similarly contain pages which consist of 64 or 1024 words. Any page of a segment can be placed in any available memory block of similar size. These pages may be addressed as if they were physically contiguous even though they are in widely scattered absolute locations. Only currently referenced pages need to be in memory at one time. If a segment is not paged, the complete segment must be brought into memory and located in contiguous address locations. In the current Multics implementation all user segments are paged in 1024 word pages. Each of these is developed by the appending hardware as described in Chapter 5.

Address Modification and Address Appending

Prior to each memory access for an operand or indirect word, two major phases of address preparation take place:

1. Address modification, if specified by the instruction or indirect word.
2. Address appending, which is a hardware process to form an address to access memory.

Although the above two types of modification are combined in most operations, they are described separately in Chapters 5 and 6.

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 memory 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 memory access.

Faults and Interrupts

The processor detects illegal operations by the software, faulty communication with memory, 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 interrupts. When a fault or interrupt is recognized, a trap results. This causes the forced execution of a pair of instructions in the memory location 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, else the faulting program will be resumed without the fault having been processed. Faults and interrupts are described in greater detail in Chapter 7.

Brief Summary of 645 Processor Features

The 645 has the following features:

1. Storage protection to place access restrictions on specified segments and pages.
2. Capability to interrupt a process in execution at any point, save processor status, and restore the status at a later time without loss of continuity of the process.
3. Capability to fetch instruction pairs. Capability to buffer four instructions including the pair 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; the operand for instruction N+1 could be buffered in the operations unit (M register); and the control unit could be preparing the address to fetch instructions N+4 and N+5 or it could be preparing the address for the operand for instruction N+3.
5. Capability to detect memory instructions that alter the contents of a buffered instruction. 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 memory accesses to the proper system controller module.
7. Intermediate storage of base address and control information in high speed registers addressable by partial contents (associative memory).
8. Intermediate storage of base address and control information in base address registers which are loaded by the executing program.
9. Absolute address computation at execution time.

PROCESSOR MODES OF OPERATION

There are two modes of memory addressing (Absolute mode and Append mode) and two modes of instruction execution (Master mode and Slave mode). In the Absolute address mode, memory is addressed directly by the address field of instructions, and all addresses are relative to the "zeroth" location of memory. The address spectrum is limited to 2^{18} locations, and Master execution mode is implied. In the Append mode, the address is calculated using the information contained in "appending words". Instructions may be executed in either Master or Slave mode, and the address spectrum is 2^{24} memory locations. All addresses are relative to the first location of the segment referred to.

Slave Mode

The Slave mode is the normal mode of operation, and most instructions can be executed in this mode. Certain instructions, classed as privileged, cannot be executed in Slave mode. These are identified in the individual instruction descriptions. An attempt to execute privileged instructions while in the Slave mode results in an illegal procedure fault. In the Slave mode, an interrupt cannot be inhibited, and various restrictions are indicated in segment descriptor words and page table words which are explained in Chapter 5. Address formation is through the appending process. The processor executes in Slave mode when the class bits of the segment descriptor word specifies either the Slave procedure or the Execute-only procedure.

Master Mode

In Master mode, all instructions can be executed. All classes of information may be accessed regardless of restrictions, with the exception that a data class may not be accessed for an instruction fetch. The timer runout fault is ignored in Master mode. An interrupt can be inhibited. Address formation is through the appending process. The processor executes in Master mode when the class bits of the segment descriptor word specify master procedure. Please refer to Chapter 5 for more detailed information.

Absolute Mode

All instructions can be executed in the Absolute mode and unrestricted access is permitted to privileged hardware features. Interrupts may be inhibited in this mode.

Instruction fetches are made with the absolute addresses relative to location zero. During instruction fetches, only the instruction counter is used; the procedure base register is ignored. Since instruction fetching is by the 18-bit absolute address, only the lower 256K of memory can be accessed while in Absolute mode.

The processor enters Absolute mode immediately after a fault or interrupt or after an instruction which restores the indicators is executed. The processor remains in Absolute mode until it executes a transfer instruction whose operand address is obtained via the appending mechanism.

Append Mode

This is the normal memory addressing mode. Operands and indirect words may be accessed via the appending mechanism by placing a one in bit position 29 of the instruction word. In this mode the effective address is either added to a base address, or its offset is linked to the base address.

The modes of operation are summarized in the table that follows.

FUNCTIONS	SLAVE	MASTER	ABSOLUTE
Executes privileged instructions.	No	Yes	Yes
Interrupt inhibited by bit 28 of an instruction.	No	Yes	Yes
Address for instruction fetch.	Appending	Appending	Absolute
Address for operand fetch.	Appending	Appending	Controlled by bit 29 of instruction.
Restriction of access to other segments and pages.	Some	Some (less restrictive than Slave)	NA

Table of Modes of Operation

Detailed information on modes can be found in the discussion of descriptor segment words and page table words in Chapter 5.

PROCESSOR UNIT FUNCTIONS

Major functions of each principal logic element are listed below and described in subsequent chapters. A block diagram on the following page shows the relationship among the processor units.

Appending Unit

Controls data input/output to memory.
Performs memory selection and interlace.
Does address appending.
Controls fault recognition.
Does power on/off sequencing.

Associative Memory Unit

Consists of sixteen 60-bit registers. The registers are used to hold pointers to most recently used segments or pages (descriptor segment words or page table words). This unit relieves the need for possible multiple memory accesses before obtaining an absolute memory address of a word.

Control Unit

Performs all processor control functions.
Performs address modification.
Controls mode of operation (Master, Slave, Absolute).
Performs interrupt recognition.
Does operation decoding.

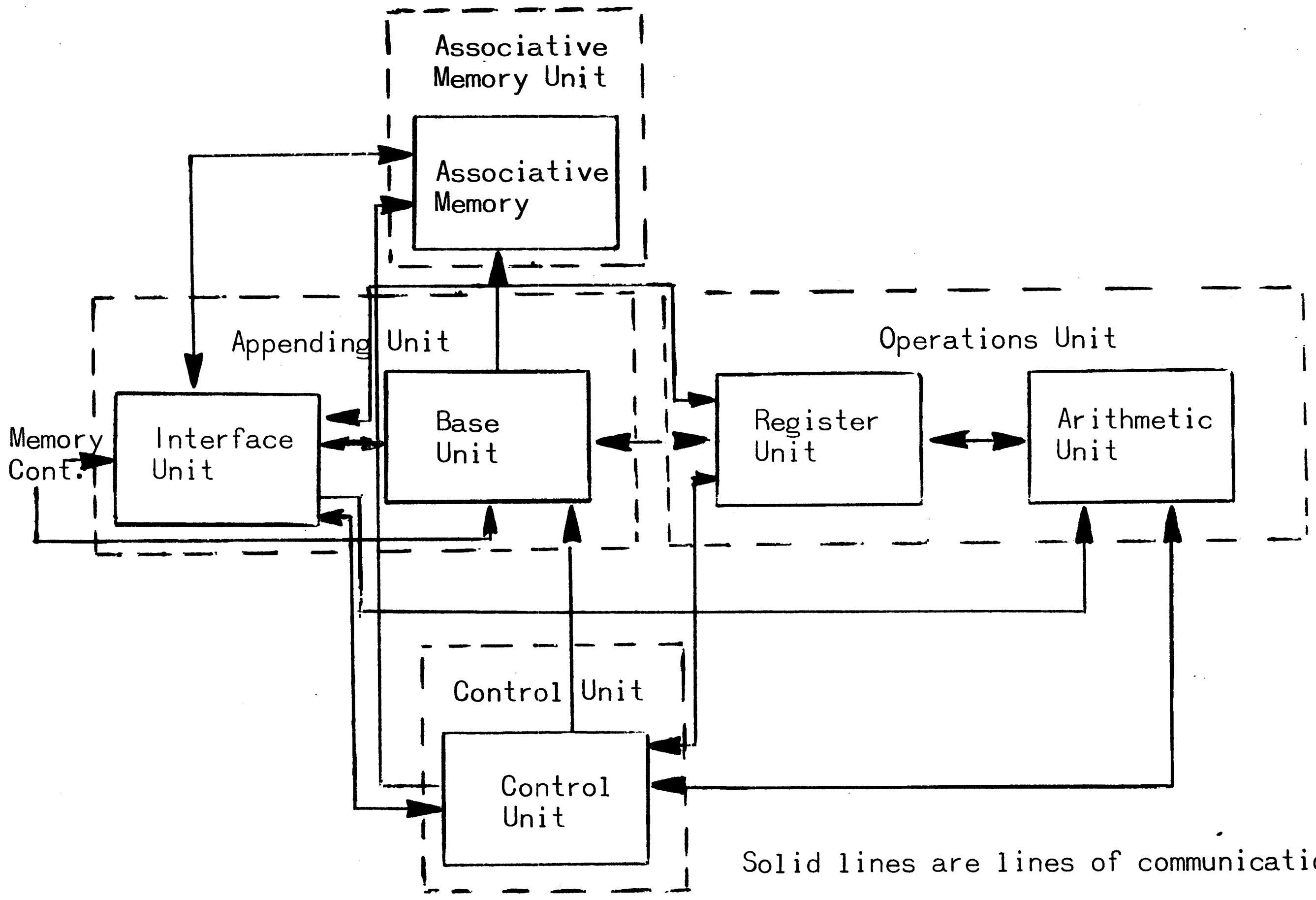
Operations Unit

Does fractional and integer divisions and multiplications.
Performs automatic alignment of floating-point numbers for addition and subtraction.
Performs inverted divisions on floating-point numbers.
Performs automatic normalization of floating-point resultants.
Does shifts.
Performs indicator register loading and storing.
Performs timer register loading and decrementing.

MAJOR UNITS OF THE PROCESSOR

The 645 processor consists of two standard 600 line cabinets which contain power supplies, blowers for cooling, and the following four principal logic elements:

Appending Unit
Associative Memory Unit
Control Unit
Operations Unit



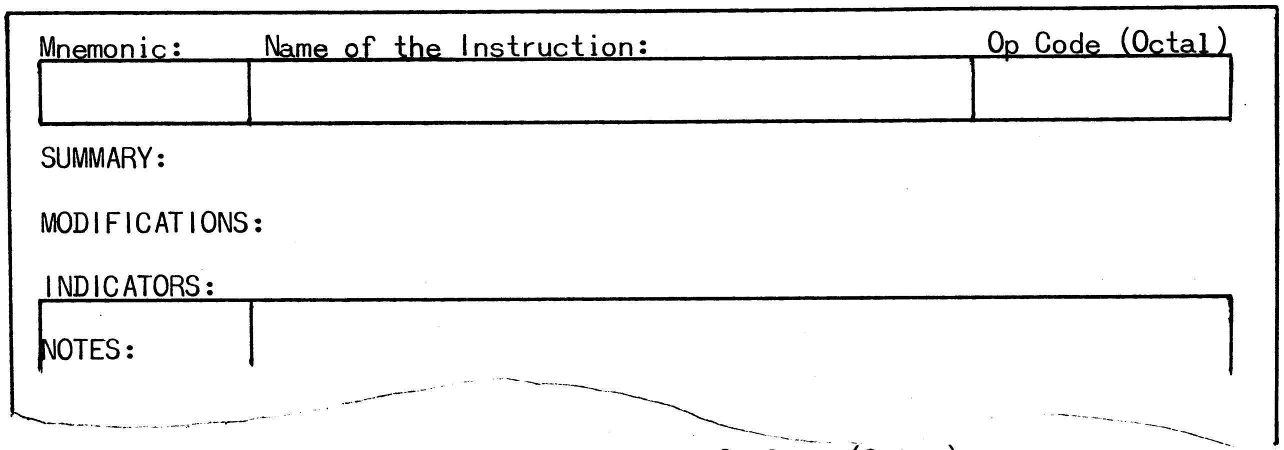
645 PROCESSOR

CHAPTER 2   645 MACHINE INSTRUCTIONS

For the description of the machine instructions that follow it is assumed that the reader is familiar with the general structure of the processor, the representation of information, the data formats, and the method of address modifications.

FORMAT OF INSTRUCTION DESCRIPTION

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



Mnemonic: Name of the Instruction: Op Code (Octal)

SUMMARY:

MODIFICATIONS:

INDICATORS:

NOTES:

Line 1: Mnemonic, Name of the Instruction, Op Code (Octal)

This line has three headings that appear over boxes containing the following:

1. Mnemonic--The mnemonic code for the Operation field of the assembler statement.

2. Name of the Instruction--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.

Line 2: SUMMARY

The change in the status of the information processing system effected by the execution of the instruction's operations is described in a short and generally symbolic form. If reference is made here to the status of an indicator, then it is the status of this indicator before the operation is executed.

Line 3: MODIFICATIONS

Those designators are listed explicitly that shall not be used with this instruction either because they are not permitted with this instruction or because their effect cannot be predicted from the general address modification procedure. (See Chapter 6.)

Line 4: INDICATORS

Only those indicators are listed whose status can be changed by the execution of this 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 this indicator remains unchanged. Unless explicitly stated otherwise, the conditions refer to the contents of registers, etc., as existing after the execution of the instruction's operation.

Line 5: NOTES

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

Abbreviations and Symbols

A	=	Accumulator Register (36 bits)
ABRn	=	Address Base Register n (n=0, 1..., 7) (24 bits)
AM	=	Associative Memory (16 registers of 60 bits per register)
AR	=	Associative Register (60 bits)
AQ	=	Combined Accumulator-Quotient Register (72 bits)
C	=	"contents of"
DBR	=	Descriptor Segment Base Register (29 bits)
E	=	Exponent Register (8 bits)
EA	=	Combined Exponent-Accumulator Register (8 + 36 bits)
EAQ	=	Combined Exponent-Accumulator-Quotient Register (8 + 72 bits)
IC (ICTC)	=	Instruction Counter (18 bits)
IR	=	Indicator Register (18 bits, 11 of which are used at this time)
PBR	=	Procedure Base Register (18 bits)
Q	=	Quotient Register (36 bits)
TBR	=	Temporary Base Register (18 bits)
TR	=	Timer Register (24 bits)
Xn	=	Index Register n (n=0, 1..., 7) (18 bits)
Z	=	Temporary Psuedo-result of a non-store comparative operation.

Absolute Address and Memory Locations

Y	=	the absolute address (24 bits specifying the core location) in memory.
Y-pair	=	a symbol denoting that the absolute address Y designates a pair of memory locations with successive addresses, the smaller address being even. When the absolute address is even, then it designates the pair Y(even), Y+1, and when it is odd, then the pair Y-1, Y(odd). The memory location with the smaller (even) address contains the most significant part of a double-precision number or the first of a pair of instructions.

Register Postions and Contents:

("R" standing for any of the registers listed above as well as for a memory location of a pair of memory locations.)

R_i	=	the ith position of R
$R_{i...j}$	=	the positions i through j of R
C(R)	=	the contents of the full register R
$C(R)_i$	=	the contents of the ith position of R
$C(R)_{i...j}$	=	the contents of the positions i through j of R

When the description of an instruction states a change for a part of a register or memory location, then it is always understood that the part of the register or memory location which is not mentioned remains unchanged.

Other Symbols:

=>	=	replaces
::	=	compare with
AND	=	the Boolean connective AND
OR	=	the Boolean connective OR
$\not\equiv$	=	the Boolean connective NON-EQUIVALENCE (or EXCLUSIVE OR)
P	=	pointer

Parity Indicator

The parity indicator is turned on at the end of a memory access which has incorrect parity.

Mnemonics

On the 635 an "effective address" corresponds to an "offset" on the 645 (see Chapter 6). Although 635 instructions implemented on the 645 actually deal with offsets, the effective address mnemonics are retained here for compatibility.

Arrangement of Instructions

Instructions in this Chapter are presented in 27 functional categories. The table on the following page identifies these and lists the first page in each category.

All instructions are listed in alphabetical order in the index.

Instruction Categories

Mnemonic:	Name of the Instruction:	Op Code (Octal)
EAA	Effective Address to A	635

SUMMARY: $Y \Rightarrow C(A)_{0-17}$; $00\ldots0 \Rightarrow C(A)_{18-35}$

MODIFICATIONS: All except DU, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF

NOTE: This instruction, and the instructions EAQ and EAXn, facilitate inter-register data movements; the data source is specified by the address modification, and the data destination by the operation code of the instruction.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
EAQ	Effective Address to Q	636

SUMMARY: $Y \Rightarrow C(Q)_{0-17}$; $00\ldots0 \Rightarrow C(Q)_{18-35}$

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDA	Load A	235

SUMMARY: $C(Y) \Rightarrow C(A)_{0-35}$

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDQ	Load Q	236

SUMMARY: $C(Y) \Rightarrow C(Q)_{0-35}$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDAQ	Load AQ	237

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

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDT	Load Timer Register	637

SUMMARY: $C(Y)_{0-23}$ => C(TR)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	If C(TR) = 0, then ON; otherwise OFF
Negative	If $C(TR)_0$ = 1, then ON; otherwise OFF

NOTE: This instruction should be used in Master mode only. If its use is attempted in the Slave mode, it generates a 635/645 Compatibility Fault.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDXn	Load Xn (n = 0, 1,..., 7)	22n

SUMMARY: $C(Y)_{0-17}$ => 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LXLn	Load Xn from Lower (n = 0, 1,...,7)	72n

SUMMARY: $C(Y)_{18-35}$ => C(Xn)

MODIFICATIONS: All except DU, DL, 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDI	Load Indicator Register	634

SUMMARY: $C(Y)_{18-28}$ => C(IR)

(Absolute mode indicator $C(Y)_{28}$ not affected)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Absolute Mode	Not Affected
All other indicators	If corresponding bit in C(Y) is ONE, then ON; otherwise OFF

NOTE: 1. The relation between bit positions of C(Y) and the indicators is as follows:

Bit Position	Indicators
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Absolute Mode (not affected)

2. The parity indicator is turned on at the end of a memory access which has incorrect parity.

3. The parity mask inhibits the parity fault, and is turned on by program control.

4. The Tally Runout indicator will reflect $C(Y)_{25}$ regardless of what address modification is performed on the LDI instruction (for Tally Operations).

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LREG	Load Registers	073

SUMMARY: $C(Y)_{0-17,\ 18-35} \Rightarrow C(X0,X1)$ $C(Y+4)_{0-35} \Rightarrow C(A)$

$C(Y+1)_{0-17,\ 18-35} \Rightarrow C(X2,X3)$ $C(Y+5)_{0-35} \Rightarrow C(Q)$

$C(Y+2)_{0-17,\ 18-35} \Rightarrow C(X4,X5)$ $C(Y+6)_{0-7} \Rightarrow C(E)$

$C(Y+3)_{0-17,\ 18-35} \Rightarrow C(X6,X7)$ $C(Y+7) \Rightarrow C(E)$

where Y must be 0 modulo(8). (If Y is not 0 modulo(8) the next smaller such address is used.)

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

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LCA	Load Complement A	335

SUMMARY: $- C(Y) \Rightarrow C(A)$ if $C(Y) \neq 0$; $C(Y) \Rightarrow C(A)$ if $C(Y) = 0$

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

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LCQ	Load Complement Q	336

SUMMARY: $-C(Y) \Rightarrow C(Q)$ for $C(Y) \neq 0$; $C(Y) \Rightarrow C(Q)$ for $C(Y) = 0$

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

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LCAQ	Load Complement AQ	337

SUMMARY: $- C(\text{Y-pair}) \Rightarrow C(AQ)$ if $C(\text{Y-pair}) \neq 0$
$C(\text{Y-pair}) \Rightarrow C(AQ)$ if $C(\text{Y-pair}) = 0$

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
Overflow	If range of AQ is exceeded, then ON

NOTE: This instruction changes the number to its negative (if ≠ 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.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LCXn	Load Complement Xn	32n

SUMMARY: $-C(Y)_{0-17}$ => C(Xn) for C(Y) ≠ 0

$C(Y)_{0-17}$ => C(Xn) for C(Y) = 0

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

NOTE: This instruction changes the number to its negative (if≠0) while moving it from Y_{0-17} to Xn. The operation is executed by forming the two's complement of the string of 18 bits.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
EAXn	Effective Address to Xn (n=1, 1,..., 7)	62n

SUMMARY: Y => C(Xn)

MODIFICATIONS: All except DU, DL

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STCD	Store Control Double	357

SUMMARY: $C(PBR) \Rightarrow C(Y)_{0-17}$, $00\ldots0 \Rightarrow C(Y)_{18-29}$, ITS Tag (100011) $\Rightarrow C(Y)_{30-35}$;

$C(ICTC) + 00\ldots010 \Rightarrow C(Y+1)_{0-17}$;

$C(IR) \Rightarrow C(Y+1)_{18-28}$, $00\ldots0 \Rightarrow C(Y+1)_{29-35}$

where Y is an even location.

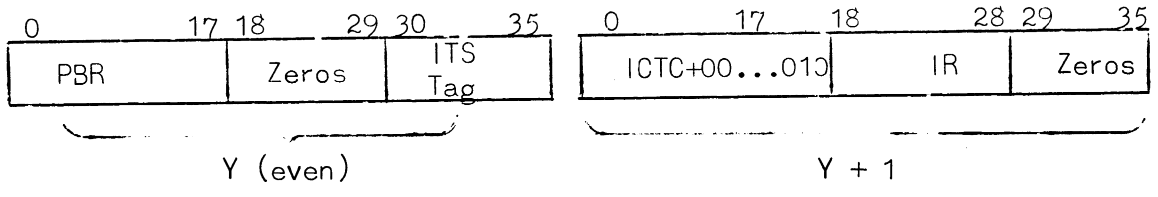
MODIFICATIONS: All except DU, DL, CI, SC, SCR. The state of the Tally Runout Indicator $C(Y+1)_{25}$ is not changed regardless of what address modification is performed on the STCD instruction for tally operations.

INDICATORS: None affected.

NOTE: This instruction stores the C(PBR, ICTC + 2, IR) in an ITS word pair. The relationship between the bit positions of C(Y+1) and the indicators are as follows:

Bit Position	Indicators
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Absolute Mode

The format for the ITS word pair in memory is as follows:



Mnemonic:	Name of the Instruction:	Op Code (Octal)
STA	Store A	755

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

MODIFICATIONS: All except DU, DL

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STAC	Store A Conditional	354

SUMMARY: Test C(Y) Then, 1. if C(Y) = 0, C(A) => C(Y) Zero indicator set ON
2. if C(Y) $\neq$ 0, Zero indicator set OFF

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

INDICATORS: (Indicators not listed are not affected)

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

NOTE: If the initial C(Y) is non-zero then C(Y) is not changed by this instruction. This instruction can be used for interlocking.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STQ	Store Q	756

SUMMARY: C(Q) = C(Y)

MODIFICATIONS: All except DU, DL

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STAQ	Store AQ	757

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

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

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STXn	Store Xn into Upper (n = 0, 1,...,7)	74n

SUMMARY: $C(Xn) \Rightarrow C(Y)_{0-17}$

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

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SXLn	Store Xn in Lower (n = 0, 1,...,7)	44n

SUMMARY: $C(Xn) = C(Y)_{18-35}$

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

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
· SREG	Store Registers	753

SUMMARY:

$C(X0) \Rightarrow C(Y)_{0-17}$	$C(X6) \Rightarrow C(Y+3)_{0-17}$
$C(X1) \Rightarrow C(Y)_{18-35}$	$C(X7) \Rightarrow C(Y+3)_{18-35}$
$C(X2) \Rightarrow C(Y+1)_{0-17}$	$C(A) \Rightarrow C(Y+4)_{0-35}$
$C(X3) \Rightarrow C(Y+1)_{18-35}$	$C(Q) \Rightarrow C(Y+5)_{0-35}$
$C(X4) \Rightarrow C(Y+2)_{0-17}$	$C(E) \Rightarrow C(Y+6)_{0-7}$; $00...0 \Rightarrow C(Y+6)_{8-35}$
$C(X5) \Rightarrow C(Y+2)_{18-35}$	$C(TR) \Rightarrow C(Y+7)_{0-23}$; $00...0 \Rightarrow C(Y+7)_{24-35}$

where Y must be a 0 modulo(8) address. (If Y is not 0 modulo(8) then the next lower such address is used.)

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

INDICATORS: None affected

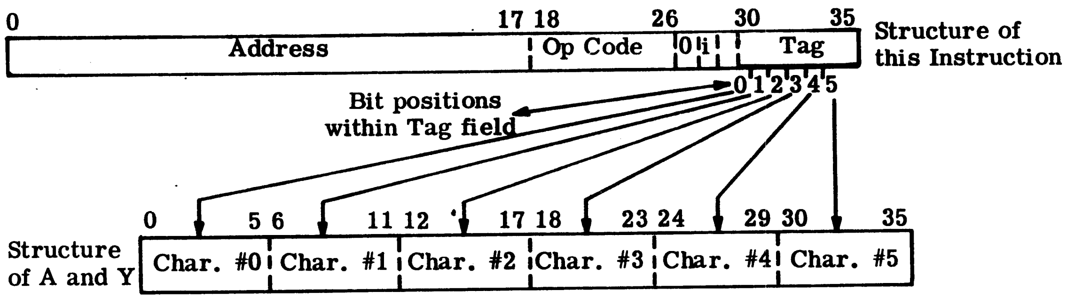
Mnemonic:	Name of the Instruction:	Op Code (Octal)
STCA	Store Character of A (Six Bit)	751

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

NOTE: Binary ones in the tag field of this instruction specify the character positions of A and Y that are affected by this instruction. The control relation is shown in the diagram below.*



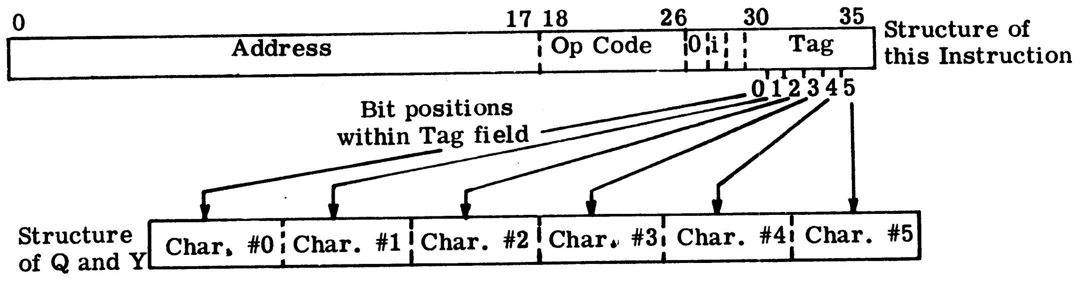
Mnemonic:	Name of the Instruction;	Op Code (Octal)
STCQ	Store Character of Q (Six Bit)	752

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

NOTE: Binary ones in the tag field of this instruction specify the character positions of Q and Y that are affected by this instruction. The control relation is shown in the diagram below.*



* Character positions in memory not stored into are left unchanged.

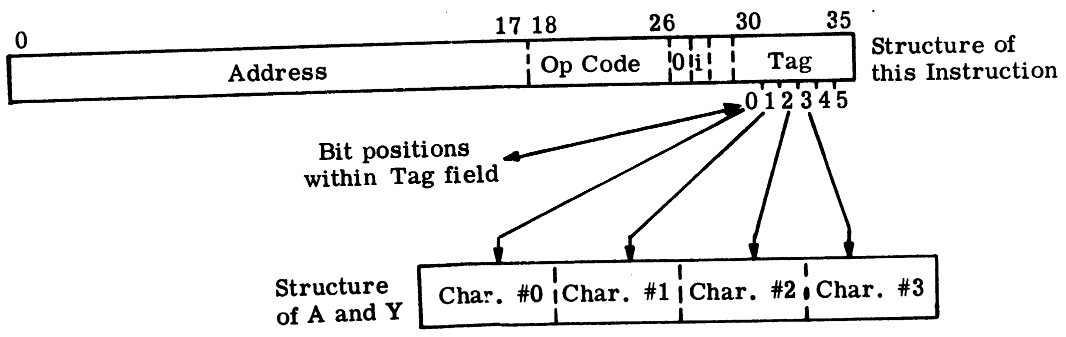
Mnemonic:	Name of the Instruction:	Op Code (Octal)
STBA	Store Character of A (Nine Bit)	551

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

NOTE: Binary ones in the tag field of this instruction specify the character positions of A and Y that are affected by this instruction. The control relation is shown in the diagram below:*



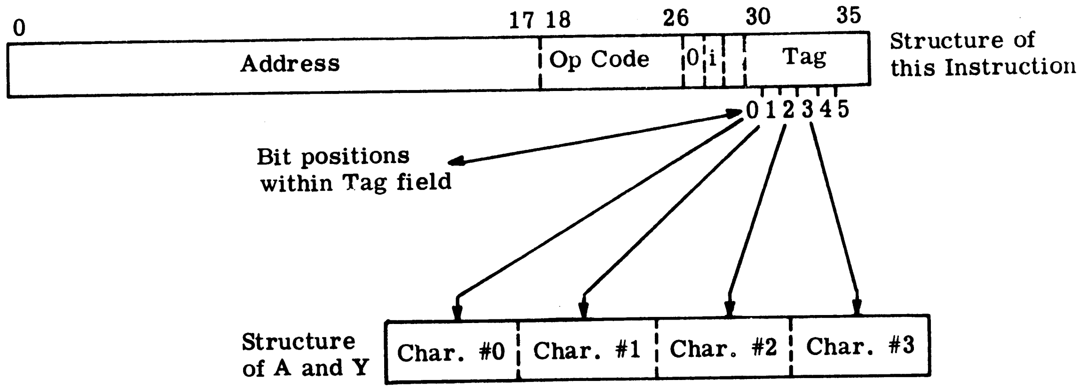
Mnemonic:	Name of the Instruction:	Op Code (Octal)
STBQ	Store Character of Q (Nine Bit)	552

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

MODIFICATIONS: None

INDICATIORS: None affected

NOTE: Binary ones in the tag field of this instruction specify the character positions of Q and Y that are affected by this instruction. The control relation is shown in the diagram below:*



* Character positions in memory not stored into are left unchanged.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STI	Store Indicator Register	754

SUMMARY: $C(IR) \Rightarrow C(Y)_{18-28}$

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

INDICATORS: None affected

NOTES:

1. The indicators (bits 18 through 28) are:

Bit Position	Indicators
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Absolute Mode

2. The parity indicator is turned on at the end of a memory access which has incorrect parity.

3. The ON state corresponds to a one bit, the OFF state to a zero bit.

4. The $C(Y)_{25}$ will contain the state of the Tally Runout indicator prior to address modification of the STI instruction (for Tally operations).

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STT	Store Timer Register	454

SUMMARY: $C(TR) \Rightarrow C(Y)_{0-23}$ $\quad$ $00...0 \Rightarrow C(Y)_{24-35}$

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

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STC1	Store Instruction Counter plus 1	554

SUMMARY: C(IC) + 0...01 => $C(Y)_{0-17}$

C(IR) => $C(Y)_{18-28}$; 00...0 => $C(Y)_{29-35}$

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

INDICATORS: None affected

NOTES: 1. The indicators (bits 18 through 28) are:

Bit Position	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Absolute Mode

2. The ON state corresponds to a one bit, the OFF state to a zero bit.

3. The $C(Y)_{25}$ will contain the state of the Tally Runout indicator prior to address modification of the STC1 instruction (for Tally operations).

4. Note the difference between STC1 and STC2.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STC2	Store Instruction Counter plus 2	750

SUMMARY: C(IC) + 0...010 => $C(Y)_{0-17}$; $C(Y)_{18-35}$ are unchanged.

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

INDICATORS: None affected

(Revised October 15, 1970)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STZ	Store Zero	450

SUMMARY: 00...0 => C(Y)

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

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
EABn	Effective Address to Base n (n = 0, 1,..., 7)	310-313 330-333

SUMMARY: Y => $C(ABRn)_{0-17}$; $C(ABRn)_{18-23}$ are unchanged, and any associated external base designated by $C(ABRn)_{18-20}$ is unchanged.

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

INDICATORS: None affected

NOTE: This instruction may be executed in Master or Slave mode. If attempted in Slave mode, an illegal procedure fault will occur unless $C(ABRn)_{22} = 0$.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
EAPn	Effective Address to Pair n (n = 0, 1,..., 7)	350-353 370-373

SUMMARY:
1. If $C(ABRn)_{21} = 0$, then EA => $C(ABRn)_{0-17}$; P => $C(ABRm)_{0-17}$
2. If $C(ABRn)_{21} = 1$, then P => $C(ABRm)_{0-17}$

where n is the designated internal ABR (unless n designates an external ABR), and m is the designated linked external ABR.

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

INDICATORS: None affected

NOTES:
1. When $C(ABRn)_{21} = 0$, the effective address EA replaces the contents of the internal ABRn designated by the EAPn instruction, and the pointer (P) generated as part of the address modification procedure replaces the contents of the associated external ABRm. If the base specified by the EAPn instruction is external, that is, $C(ABRn)_{21} = 1$, then P => $C(ABRn)_{0-17}$.

2. This instruction may be executed in Master or Slave mode. If attempted in Slave mode an illegal procedure fault is generated unless $C(ABRn,m)_{22} = 0$.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADBn	Add to Address Base Register n (n = 0, 1,..., 7)	050-053 150-153

SUMMARY: $C(Y)_{0-17} + C(ABRn)_{0-17} \Rightarrow C(ABRn)_{0-17}$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: None affected

NOTE: This instruction may be executed in Master or Slave mode. If attempted in Slave mode an illegal procedure fault is generated unless $C(ABRn)_{22} = 0$. The ABR specified by the ADBn instruction may be an internal or external base ($C(ABRn)_{21} = 0$ or 1).

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LBRn	Load Address Base Register n (n=0,1,...7)	76n

SUMMARY: $C(Y)_{0-23} \Rightarrow C(ABR)n$

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

INDICATORS: None affected

NOTE: If this instruction is attempted in Slave mode an illegal procedure fault is generated unless $C(ABRn)_{22} = 0$. The $C(ABRn)_{22}$ is not altered in Slave mode.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDB	Load Bases	173

SUMMARY: $C(Y,Y+1,\ldots,Y+7)_{0-23} \Rightarrow C(ABR0,\ldots,ABR7)$
where Y must be 0 modulo(8) address. (If Y is not 0 modulo(8) then the next lower such address is used.)

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

INDICATORS: None affected

NOTE: The 8 ABR's are loaded in sequence in double word loads. The contents of the affected ABR may be altered in Master or Slave mode if $C(ABRn)_{22} = 0$. However the ABR will not be affected and no fault will occur while executing LDB in Slave mode if $C(ABRn)_{22} = 1$.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDBR	Load Descriptor Segment Base Register	232

SUMMARY: $C(Y)_{0-28}$ => C(DBR)

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

INDICATORS: None affected

NOTE: This instruction may be executed only in Master mode. If attempted in Slave mode an illegal procedure fault will occur, and C(DBR) will remain unchanged. The associative memory is cleared (bit 54 of all AR's set to zero) when LDBR is executed.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDCF	Load Control Field	512

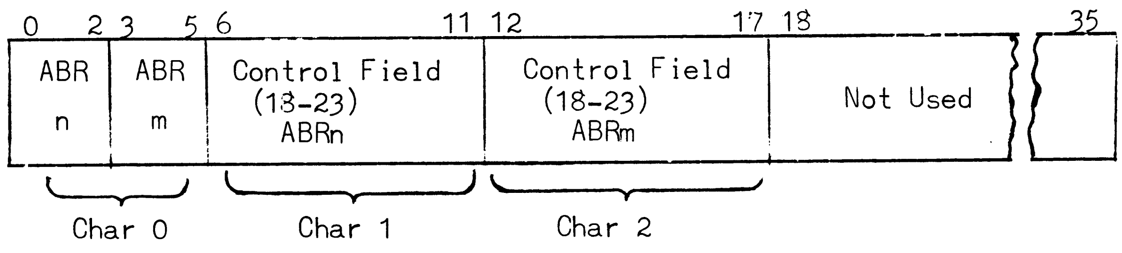
SUMMARY: $C(Y)_{6-11}$ => $C(ABRn)_{18-23}$; $C(Y)_{12-17}$ => $C(ABRm)_{18-23}$ where n is an internal ABR specified by $C(Y)_{0-2}$ and m is an associated external ABR specified by $C(Y)_{3-5}$.

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: None affected

NOTE: The $C(Y)_{0-17}$ is interpreted as three 6-bit characters specifying two ABR's and the corresponding new ABR control field information. This instruction may be executed in Master or Slave mode. If attempted in Slave mode an illegal procedure fault is generated unless $C(ABRn,m)_{22}$ = 0. The $C(ABRn,m)_{22}$ is not altered in Slave mode.

Format of C(Y) is as follows:



Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBRn	Store Address Base Register n (n = 0, 1,..., 7)	54n

SUMMARY: $C(ABRn) \Rightarrow C(Y)_{0-23}$; $00...0 \Rightarrow C(Y)_{24-35}$

where n may designate an internal or external ABR.

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

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STB	Store Bases	254

SUMMARY: $C(ABR0,..., ABR7) \Rightarrow C(Y,..., Y+7)_{0-23}$; $00...0 \Rightarrow C(Y,..., Y+7)_{24-35}$
where Y must be 0 modulo(8) address. (If Y is not 0 modulo(8) then the next lower such address is used.)

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

INDICATORS: None affected

NOTE: The contents of the eight ABR's are stored in sequence in double word stores.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SDBR	Store Descriptor Segment Base Register	154

SUMMARY: $C(DBR) \Rightarrow C(Y)_{0-28}$; $00...0 \Rightarrow C(Y)_{29-35}$

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

INDICATORS: None affected

NOTE: This instruction may be executed in Master mode only or an illegal procedure fault is generated.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STPn	Store Pair n(n = 0, 1,...,7)	250-253 650-653

SUMMARY:

1. If $C(ABRn)_{21} = 0$, then
   $C(ABRm) \Rightarrow C(Y)_{0-17}$;
   $00...0 \Rightarrow C(Y)_{18-29}$, ITS Tag (100011) $\Rightarrow C(Y)_{30-35}$;
   $C(ABRn) \Rightarrow C(Y+1)_{0-17}$, $00...0 \Rightarrow C(Y+1)_{18-35}$;

2. If $C(ABRn)_{21} = 1$, then
   $C(ABRm) \Rightarrow C(Y)_{0-17}$,
   $00...0 \Rightarrow C(Y)_{18-29}$, ITS Tag (100011) $\Rightarrow C(Y)_{30-35}$;
   $00...0 \Rightarrow C(Y+1)_{0-35}$

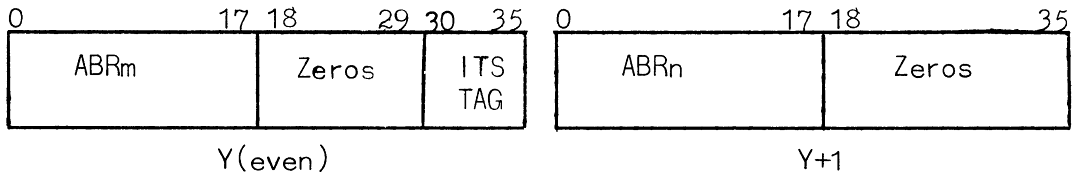
where n is the designated internal ABR
m is the designated linked ABR, and
Y is an even location.

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

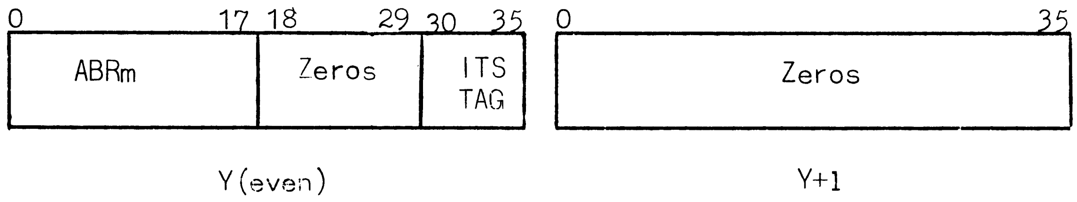
INDICATORS: None affected

NOTE: This instruction stores the contents of the internal ABR designated by the STPn instruction and the linked external ABR as an ITS word pair.
If an external base is designated by the STPn instruction, that is, $C(ABRn)_{21} = 1$, then an odd word of all zeros will be stored. The format of the word pair in memory is as follows:

1. = 0



2. = 1



Mnemonic:	Name of the Instruction:	Op Code (Octal)
ARL	A Right Logic	771

SUMMARY: Shift right C(A) the number of positions specified by the address field of the instruction (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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
QRL	Q Right Logic	772

SUMMARY: Shift right C(Q) the number of positions specified by the address field of the instruction (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(Q) = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LRL	Long Right Logic	773

SUMMARY: Shift right C(AQ) the number of positions specified by the address field of the instruction (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(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ALR	A Left Rotate	775

SUMMARY: Rotate C(A) by Y_{11-17} positions; enter each bit leaving position 0 into position 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
QLR	Q Left Rotate	776

SUMMARY: Rotate C(Q) by Y_{11-17} positions; enter each bit leaving position 0 into position 35.

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LLR	Long Left Rotate	777

SUMMARY: Rotate the C(AQ) left by Y_{11-17} positions; enter each bit leaving A_0 into Q_{35}.

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ARS	A Right Shift	731

SUMMARY: Shift right C(A) the number of positions specified by the address field of the instruction (Y_{11-17}); 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
QRS	Q Right Shift	732

SUMMARY: Shift right C(Q) the number of positions specified by the address field of the instruction (Y_{11-17}); fill vacated positions with $C(Q)_0$.

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LLS	Long Left Shift	737

SUMMARY: Shift left C(AQ) the number of positions specified by the address field of the instruction (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(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Carry	If $C(AQ)_0$ ever changes during the shift, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
QLS	Q Left Shift	736

SUMMARY: Shift left C(Q) the number of positions specified by the address field of the instruction (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(Q) = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ = 1, then ON; otherwise OFF
Carry	If $C(Q)_0$ ever changes during the shift, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ALS	A Left Shift	735

SUMMARY: Shift left C(A) the number of positions specified by the address field of the instruction (Y_{11-17}); fill vacated positions with zeros.

MODIFICATIONS: All except DU, CL, 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 $C(A)_0$ ever changes during the shift, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LRS	Long Right Shift	733

SUMMARY: Shift right C(AQ) the number of positions specified by the address field of the instruction (Y_{11-17}); 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

FIXED-POINT ADDITION

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADA	Add to A	075

SUMMARY: C(A) + C(Y) => 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
Carry	If a carry out of A_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADQ	Add to Q	076

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
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of Q_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADAQ	Add to AQ	077

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
Overflow	If range of AQ is exceeded, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADXn	Add to Xn (n = 0, 1,...,7)	06n

SUMMARY: C(Xn) + $C(Y)_{0-17}$ => 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
Carry	If a carry out of Xn_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ASA	Add Stored to A	055

SUMMARY: C(A) + C(Y) => C(Y)   C(A) unchanged

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ASQ	Add Stored to Q	056

SUMMARY: C(Q) + C(Y) => C(Y)   C(Q) unchanged

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ASXn	Add Stored to Xn	04n

SUMMARY: $C(Xn) + C(Y)_{0-17} \Rightarrow C(Y)_{0-17}$

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADLA	Add Logic to A	035

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	Not Affected
Carry	If a carry out of A_0 is generated then ON; otherwise OFF

NOTE: This instruction is identical to the ADA instruction with the exception that the overflow indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADLQ	Add Logic to Q	036

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
Overflow	Not Affected
Carry	If a carry out ot Q_0 is generated then ON; otherwise OFF

NOTE: This instruction is identical to the ADQ instruction with the exception that the overflow indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADLAQ	Add Logic to AQ	037

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

MODIFICATIONS: All except DU, DL, CI, SC

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	Not Affected
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the ADAQ instruction with the exception that the overflow indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADLXn	Add Logic to Xn (n = 0, 1,...,7)	02n

SUMMARY: $C(Xn) + C(Y)_{0-17} \Rightarrow C(Xn)$

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	Not Affected
Carry	If a carry out of Xn_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the ADXn instruction with the exception that the overflow indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
AWCA	Add with Carry to A	071

SUMMARY: Carry Indicator OFF: $C(A) + C(Y) \Rightarrow C(A)$
Carry Indicator ON: $C(A) + C(Y) + 0...01 \Rightarrow C(A)$

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

NOTE: This instruction is identical to the ADA instruction with the exception that, when the Carry indicator is ON at the beginning of the instruction, then +1 is added to the least-significant position.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
AWCQ	Add with Carry to Q	072

SUMMARY: Carry Indicator OFF: C(Q) + C(Y) => C(Q)
Carry Indicator ON: C(Q) + C(Y) + 0...01 => 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
Carry	If carry out of Q_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the ADQ instruction with the exception that, in case the Carry indicator is ON at the beginning of the instruction, then +1 is added to the least-significant position.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADL	Add Low to AQ	033

SUMMARY: C(AQ) + C(Y), sign extended, => C(AQ)

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
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

NOTE: a 72-bit number is formed:

$$\underbrace{C(Y_0), C(Y_0), \ldots\ldots, C(Y_0),}_{36 \text{ bits}} C(Y).$$

Its lower half (bits 36-71) is identical to C(Y), and each of the bits of its upper half (bits 0-35) is identical to the sign bit of C(Y), i.e., to $C(Y_0)$.

This number is added to the contents of the combined AQ-register, effecting the addition of C(Y) to the lower half of the combined AQ-register with a possible carry out of the Q-part being passed on to the A-part.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
AOS	Add One to Storage	054

SUMMARY: C(Y) + 0...01 => 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
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBA	Subtract from A	175

SUMMARY: C(A) - C(Y) => 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
Carry	If a carry out of A_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBQ	Subtract from Q	176

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
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of Q_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBAQ	Subtract from AQ	177

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
Overflow	If range of AQ is exceeded, then ON; otherwise OFF
Carry	If carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBXn	Subtract from Xn (n = 0,1,...,7)	16n

SUMMARY: C(Xn) - $C(Y)_{0-17}$ => 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
Carry	If a carry out of Xn_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SSA	Subtract Stored from A	155

SUMMARY: C(A) - C(Y) => 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
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SSQ	Subtract Stored from Q	156

SUMMARY: C(Q) - C(Y) => 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
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SSXn	Subtract Stored from Xn	14n

SUMMARY: $C(Xn) - C(Y)_{0-17} \Rightarrow C(Y)_{0-17}$

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBLA	Subtract Logic from A	135

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	Not Affected
Carry	If a carry out of A_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the SBA instruction with the exception that the overflow indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBLQ	Subtract Logic from Q	136

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
Overflow	Not Affected
Carry	If a carry out of Q_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the SBQ instruction with the exception that the overflow indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive integers.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBLAQ	Subtract Logic from AQ	137

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
Overflow	Not affected
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the SBAQ instruction with the exception that the overflow indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SWCA	Subtract with Carry from A	171

SUMMARY: Carry Indicator ON: $C(A) - C(Y) \Rightarrow C(A)$
Carry Indicator OFF: $C(A) - C(Y) - 0\ldots01 \Rightarrow C(A)$

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

NOTES:
1. This instruction is identical to the SBA instruction with the exception that, when the Carry indicator is OFF at the beginning of the instruction, then +1 is subtracted from the least-significant position.
2. This instruction is used for multiple-word precision arithmetic.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SWCQ	Subtract with Carry from Q	172

SUMMARY: Carry Indicator ON: C(Q) - C(Y) => C(Q)
C(Q) - C(Y) - 0...01 => 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
Carry	If carry out of Q_0 is generated, then ON; otherwise OFF

NOTES: 1. This instruction is identical to the SDQ instruction with the exception that, in case the Carry indicator is OFF at the beginning of the instruction, then +1 is subtracted from the least-significan position.

2. This instruction is used for multiple-word precision arithmetic.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBLXn	Subtract Logic from Xn (n = 0,1,...,7)	12n

SUMMARY: C(Xn) - $C(Y)_{0-17}$ => 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	Not Affected
Carry	If a carry out of Xn_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the SBXn instruction with the exception that the overflow indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MPY	Multiply Integer	402

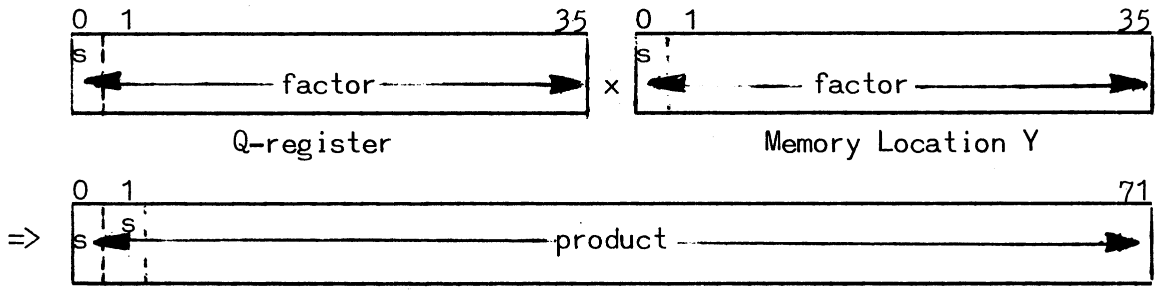
SUMMARY: C(Q) x C(Y) => C(AQ), right-adjusted

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF

NOTES: 1. 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. Bit position AQ_0 is filled with an "extended sign bit".



Combined AQ-register

2. In the case of $(-2^{35}) \times (-2^{35}) = +2^{70}$, the position AQ_1 is used to represent the product rather than as an extension of the sign. No overflow can occur.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MPF	Multiply Fraction	401

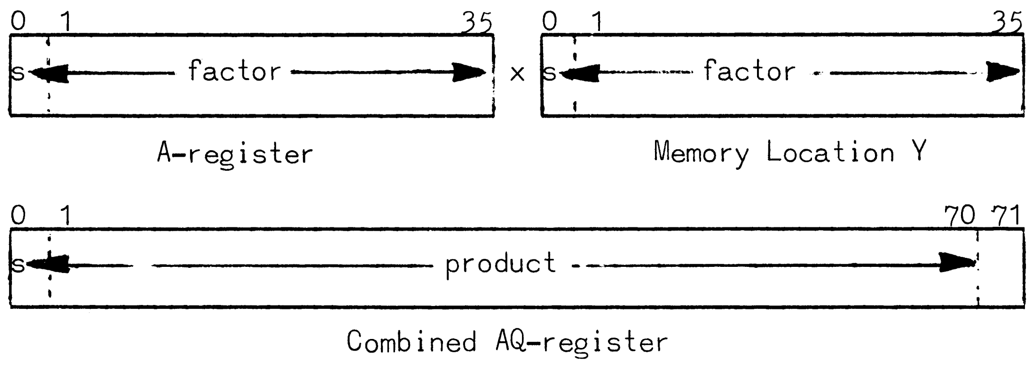
SUMMARY: C(A) x C(Y) => C(AQ), left justified

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

NOTE: Two 36-bit fractional factors (including sign) are multiplied to form a 71-bit fractional product (including sign), which is stored left-justified in the AQ register; bit position AQ_{71} 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.



FIXED-POINT DIVISION

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DIV	Divide Integer	506

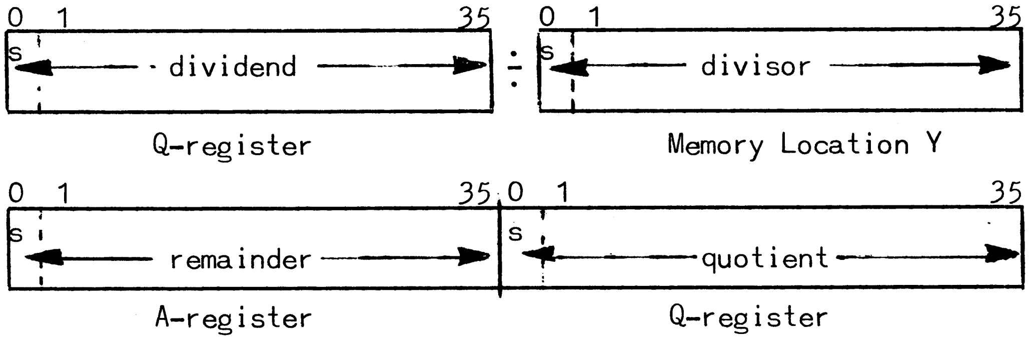
SUMMARY: $C(Q) \div C(Y)$; integer quotient => C(Q)
integer remainder => C(A)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
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: 1. 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 fractional remainder (including sign). The remainder sign is equal to the dividend sign unless the remainder is zero.



2. If dividend = -2^{35} and divisor = -1 or if divisor = 0, then the division itself does not take place.

Instead, a Divide-Check fault trap occurs; the divisor C(Y) remains unchanged, C(Q) contains the dividend magnitude in absolute, and the negative indicator reflects the dividend sign.

FIXED-POINT DIVISION

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DVF	Divide Fraction	507

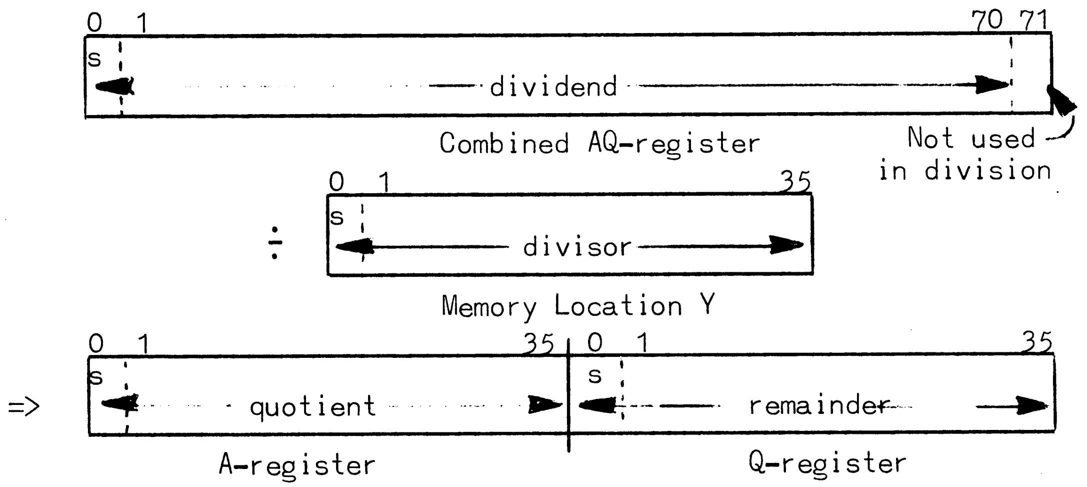
SUMMARY: C(AQ) ÷ C(Y); fractional quotient => C(A)
remainder => C(Q)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
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: 1. A 71-bit fractional dividend (including sign) is divided by a 36-bit fractional divisor (including sign) to form a 36-bit fractional quotient (including sign) and a 36-bit remainder (including sign), bit position 35 of the remainder is corresponding to bit position 70 of the dividend. The remainder sign is equal to the dividend sign unless the remainder is zero.



2. If |dividend| ≥ |divisor| or if divisor = 0, then the division itself does not take place.

Instead, a Divide-Check fault trap occurs; the divisor C(Y) remains unchanged, C(AQ) contains the dividend magnitude in absolute, and the negative indicator reflects the dividend sign.

FIXED-POINT NEGATE

Mnemonic:	Name of the Instruction:	Op Code (Octal)
NEG	Negate A	531

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

MODIFICATIONS: None

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 0, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON

NOTE: This instruction changes the number in A to its negative (if ≠ 0). The operation is executed by forming the two's complement of the string of 36 bits.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
NEGL	Negate Long	533

SUMMARY: -C(AQ) => C(AQ)

MODIFICATIONS: None

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

NOTE: This instruction changes the number in AQ to its negative (if ≠ 0). The operation is executed by forming the two's complement of the string of 72 bits.

FLOATING POINT LOAD

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FLD	Floating Load	431

SUMMARY: $C(Y)_{0-7}$ => C(E), $C(Y)_{8-35}$ => $C(AQ)_{0-27}$; 00...0 => $C(AQ)_{28-71}$

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFLD	Double-Precision Floating Load	433

SUMMARY: $C(Y\text{-pair})_{0-7}$ => C(E), $C(Y\text{-pair})_{8-71}$ => $C(AQ)_{0-63}$, 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDE	Load Exponent Register	411

SUMMARY: $C(Y)_{0-7}$ => C(E)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

ZERO	Set OFF
Negative	Set OFF

FLOATING POINT STORE

Mnemonic:	Name of the Instruction	Op Code (Octal)
FST	Floating Store	455

SUMMARY: C(E,A) => C(Y)

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

INDICATORS: None affected

NOTE: This instruction is executed as follows:

$C(E) \Rightarrow C(Y)_{0-7}$
$C(A)_{0-27} \Rightarrow C(Y)_{8-35}$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFST	Double-Precision Floating Store	457

SUMMARY: C(E,A,Q) => C(Y-pair)

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

INDICATORS: None affected

NOTE: This instruction is executed as follows:

$C(E) \Rightarrow C(\text{Y-pair})_{0-7}$
$C(AQ)_{0-63} \Rightarrow C(\text{Y-pair})_{8-71}$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FSTR	Floating Store Rounded	470

SUMMARY: [C(EAQ)] rounded => C(Y)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Exp. Overflow	If exponent above +127, then ON

NOTES:

1. During single-precision floating point stores, this instruction rounds the number (positive or negative) as it is stored.

2. The instruction is executed by adding a binary one to bit position 28 of AQ, truncating, then storing the contents of AQ. Steps in the execution may be conceived of as follows:

   $C(AQ)_{0-71} + 2^{-28} => C(AQ)_{0-27}$

   $00...0 => C(AQ)_{28-71}$

   $C(E) => C(Y)_{0-7}$

   $C(A)_{0-27} => C(Y)_{8-35}$

   Restore C(EAQ) to original values.

3. FSTR is a special type of store instruction, which is handled in a different manner than are normal stores. In the execution of this instruction, the mantissa in the A-register is transferred to N_{0-35} and a one is added to bit 28, which in effect rounds the mantissa to bits 0-27. The rounded mantissa and the exponent from the E-register are stored in storage location Y as a normal floating store. If the mantissa overflows when it is rounded, it is shifted one bit position right, and the exponent is increased by one before it is stored. If increasing the exponent by one causes it to overflow, the Exp. Overflow indicator will be set. In no case will the contents of A or E be altered. If the mantissa is rounded to all zeros, the zero indicator will not be set and the exponent will not be forced to -128.

4. All registers remain unchanged.

5. An exponent overflow occurs only if C(E) = +127 and $C(AQ)_{0-28}$ = 0.111...111 before rounding.

(FSTR Instruction, Continued)

6. If the original operand is a negative number

   $[C(AQ)_{0-28} = 1.0111...111 \text{ and } C(AQ)_{29-71} = 0]$,

   the number is rounded towards zero, not towards a more negative value, and the result becomes unnormalized.

7. Normalization occurs only if the mantissa overflows when it is rounded.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STE	Store Exponent Register	456

SUMMARY: $C(E) \Rightarrow C(Y)_{0-7}$; $00...0 \Rightarrow C(Y)_{8-17}$

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

INDICATORS: None affected

FLOATING POINT ADDITION

Mnemonic:	Name of the Instruction	Op Code (Octal)
FAD	Floating Add	475

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
Exp. Overflow	If Exponent above + 127, then ON
Exp. Underflow	If Exponent below - 128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction	Op Code (Octal)
UFA	Unnormalized Floating Add	435

SUMMARY: [C(EAQ) + C(Y)] not 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
Exp. Overflow	If Exponent above + 127, then ON
Exp. Underflow	If Exponent below - 128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFAD	Double-Precision Floating Add	477

SUMMARY: [C(EAQ) + C(Y-pair)] normalized => 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal
DUFA	Double-Precision Unnormalized Floating Add	437

SUMMARY: [C(EAQ) + C(Y-pair)] not normalized => 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADE	Add to Exponent Register	415

SUMMARY: $C(E) + C(Y)_{0-7} \Rightarrow C(E)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	Set OFF
Negative	Set OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FSB	Floating Subtract	575

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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
UFS	Unnormalized Floating Subtract	535

SUMMARY: [C(EAQ) - C(Y)] not 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

FLOATING POINT SUBTRACTION

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFSB	Double-Precision Floating Subtract	577

SUMMARY: [C(EAQ) - C(Y-pair)] normalized => 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DUFS	Double-Precision Unnormalized Floating Subtract	537

SUMMARY: [C(EAQ) - C(Y-pair)] not normalized => 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

FLOATING POINT MULTIPLICATION

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FMP	Floating Multiply	461

SUMMARY: [C(EAQ) x 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTE: This multiplication is executed as follows:

$C(E) + C(Y)_{0-7}$ => C(E)

$C(AQ) \times C(Y)_{8-35}$ results in a 98-bit product plus sign, the leading 71 bits plus sign of which => C(AQ)

C(EAQ) normalized => C(EAQ)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
UFM	Unnormalized Floating Multiply	421

SUMMARY: [C(EAQ) x C(Y)] not 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTE: This multiplication is executed like the instruction FMP with the exception that the final normalization is performed only in the case of both factor mantissas being = -1.00...0.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFMP	Double-Precision Floating Multiply	463

SUMMARY: [C(EAQ) x C(Y-pair)] normalized => 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTE: This multiplication is executed as follows:

C(E) + $C(Y\text{-pair})_{0-7}$ => C(E)

C(AQ) x $C(Y\text{-pair})_{8-71}$ results in a 134-bit product plus sign, the leading 71 bits plus sign of which => C(AQ)

C(EAQ) normalized => C(EAQ)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DUFM	Double-Precision Unnormal Floating Multiply	423

SUMMARY: [C(EAQ) x C(Y-pair)] not normalized => 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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTE: This multiplication is executed like the instruction DFMP, with the exception that the final normalization is performed only in the case of both factor mantissas being = -1.00...0.

FLOATING POINT DIVISION

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FDV	Floating Divide	565

SUMMARY: $C(EAQ) \div C(Y) \Rightarrow C(EA)$; $00\ldots0 \Rightarrow C(Q)$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
Zero	If $C(A) = 0$, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON	
Exp. Underflow	If exponent below -128, then ON	

NOTES:

1. This division is executed as follows:

   The dividend mantissa C(AQ) is shifted right and the dividend exponent C(E) increased accordingly until

   $\left| C(AQ)_{0-27} \right| < \left| C(Y)_{8-35} \right|$;

   $C(E) - C(Y)_{0-7} \Rightarrow C(E)$;

   $C(AQ) \div C(Y)_{8-35} \Rightarrow C(A)$;

   $00\ldots0 \Rightarrow C(Q)$.

2. If mantissa of divisor = 0, then the division itself does not take place. Instead, a Divide-Check fault trap occurs. The divisor C(Y) remains unchanged, C(AQ) contains the dividend magnitude in absolute, and the negative indicator reflects the dividend sign.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FDI	Floating Divide Inverted	525

SUMMARY: C(Y) $\div$ C(EAQ) => C(EA); 00...0 => C(Q)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
Zero	If C(A) = 0, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 0, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON	
Exp. Underflow	If exponent below -128, then ON	

NOTES: 1. This division is executed as follows:

The dividend mantissa $C(Y)_{8-35}$ is shifted right and the dividend exponent $C(Y)_{0-7}$ increased accordingly until $|C(Y)_{8-35}| < |C(AQ)_{0-27}|$;

$C(Y)_{0-7}$ - C(E) => C(E);

$C(Y)_{8-35} \div$ C(AQ) => C(A);

00...0 => C(Q).

2. If mantissa of divisor = 0, then the division itself does not take place. Instead, a Divide-Check fault trap occurs; and all the registers remain unchanged.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFDV	Double-Precision Floating Divide	567

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

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

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
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
Exp. Overflow	If exponent above +127, then ON	
Exp. Underflow	If exponent below -128, then ON	

NOTES: 1. This division is executed as follows:

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

$C(E) - C(Y\text{-pair})_{0-7} \Rightarrow C(E)$;

$C(AQ) \div C(Y\text{-pair})_{8-71} \Rightarrow C(AQ)_{0-63}$;

$00\ldots0 \Rightarrow C(AQ)_{64-71}$

2. If mantissa of divisor = 0, then the division itself does not take place. Instead, a Divide-Check fault trap occurs. The divisor C(Y) remains unchanged, C(AQ) contains the dividend magnitude in absolute, and the negative indicator reflects the dividend sign.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFDI	Double-Precision Floating Divide Inverted.	527

SUMMARY: C(Y-pair) $\div$ C(EAQ) => C(EAQ)

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

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
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
Exp. Overflow	If exponent above +127, then ON	
Exp. Underflow	If exponent below -128, then ON	

NOTES: 1. This division is executed as follows:

The dividend mantissa $C(Y\text{-pair})_{8-71}$ is shifted right and the dividend exponent $C(Y\text{-pair})_{0-7}$ increased accordingly until $| C(Y\text{-pair})_{8-71} | < | C(AQ)_{0-63} |$;

$C(Y\text{-pair})_{0-7}$ - C(E) => C(E);

$C(Y\text{-pair})_{8-71} \div C(AQ)$ => $C(AQ)_{0-63}$;

00...0 => $C(AQ)_{64-71}$

2. If mantissa of divisor = 0, then the division itself does not take place. Instead, a Divide-Check fault trap occurs; and all the registers remain unchanged.

FLOATING POINT NEGATE

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FNEG	Floating Negate	513

SUMMARY: - C(AQ) normalized => C(AQ)

MODIFICATIONS: None

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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTES:

1. This instruction changes the number in 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).

2. Even if originally C(EAQ) were normalized, an exponent overflow can still occur, namely when originally C(AQ) = -1.00...0 and C(E) = +127.

FLOATING POINT NORMALIZE

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FNO	Floating Normalize	573

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

MODIFICATIONS: None

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
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Overflow	Set OFF

NOTES:

1. The instruction normalizes the number in EAQ. If the overflow indicator is ON, then the number in EAQ is normalized one place to the right; and then the sign bit $C(AQ)_0$ is inverted in order to reconstitute the actual sign. Furthermore, the overflow indicator is set OFF.

2. This instruction can be used to correct overflows that occurred with fixed-point numbers.

FLOATING POINT COMPARE

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FCMP	Floating Compare	515

SUMMARY: Algebraic comparison C $[(E)(AQ_{0-27})]$ :: C(Y)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$C\ [(E)(AQ_{0-27})] > C(Y)$
1	0	$C\ [(E)(AQ_{0-27})] = C(Y)$
0	1	$C\ [(E)(AQ_{0-27})] < C(Y)$

NOTE: This comparison is executed as follows:

Compare C(E) :: $C(Y)_{0-7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents.

Then compare the mantissas and set the indicators accordingly.

FLOATING POINT COMPARE

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FCMG	Floating Compare Magnitude	425

SUMMARY: Algebraic comparison $|C\ [(E)(AQ_{0-27})]|\ ::\ |C(Y)|$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$\|C\ [(E)(AQ_{0-27})]\| > \|C(Y)\|$
1	0	$\|C\ [(E)(AQ_{0-27})]\| = \|C(Y)\|$
0	1	$\|C\ [(E)(AQ_{0-27})]\| < \|C(Y)\|$

NOTE: This comparison is executed as follows:

Compare $C(E)\ ::\ C(Y)_{0-7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents.

Then compare the absolute value of the mantissas and set the indicators accordingly.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFCMP	Double-Precision Floating Compare	517

SUMMARY: Algebraic comparison $C[(E)(AQ_{0-63})]$ :: $C(Y\text{-pair})$

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

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$C\ [(E)(AQ_{0-63})] > C(Y\text{-pair})$
1	0	$C\ [(E)(AQ_{0-63})] = C(Y\text{-pair})$
0	1	$C\ [(E)(AQ_{0-63})] < C(Y\text{-pair})$

NOTE: This comparison is executed as follows:

Compare $C(E)$ :: $C(Y)_{0-7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents.

Then compare the mantissas and set the indicators accordingly.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFCMG	Double-Precision Floating Compare Magnitude	427

SUMMARY: Algebraic comparison $|C\ [(E)(AQ_{0-63})]|\ ::\ |C(Y\text{-pair})|$

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

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$|C\ [(E)(AQ_{0-63})]| > |C(Y\text{-pair})|$
1	0	$|C\ [(E)(AQ_{0-63})]| = |C(Y\text{-pair})|$
0	1	$|C\ [(E)(AQ_{0-63})]| < |C(Y\text{-pair})|$

NOTE: This comparison is executed as follows:

Compare C(E) :: $C(Y)_{0-7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents.

Then compare the absolute value of the mantissas and set the indicators accordingly.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FSZN	Floating Set Zero and Negative Indicators from Memory	430

SUMMARY: Test the Number C(Y)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	Mantissa $C(Y)_{8-35} > 0$
1	0	Mantissa $C(Y)_{8-35} = 0$
0	1	Mantissa $C(Y)_{8-35} < 0$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANA	AND to A	375

SUMMARY: $C(A)_i$ AND $C(Y)_i$ => $C(A)_i$ for all 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANQ	AND to Q	376

SUMMARY: $C(Q)_i$ AND $C(Y)_i$ => $C(Q)_i$ for all i = 0, 1, ..., 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANAQ	AND to AQ	377

SUMMARY: $C(AQ)_i$ AND $C(Y\text{-pair})_i$ => $C(AQ)_i$ for all 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANXn	AND to Xn (n = 0, 1, ..., 7)	36n

SUMMARY: $C(Xn)_i$ AND $C(Y)_i$ => $C(Xn)_i$ for all 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANSA	AND to Storage A	355

SUMMARY: $C(A)_i$ AND $C(Y)_i$ => $C(Y)_i$ for all i = 0, 1, ..., 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANSQ	AND to Storage Q	356

SUMMARY: $C(Q)_i$ AND $C(Y)_i$ => $C(Y)i$ for all i = 0, 1, ..., 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

Mnemonic:	Name of the Instruction:		Op Code (Octal)
ANSXn	AND to Storage Xn	(n = 0, 1, ..., 7)	34n

SUMMARY: $C(Xn)_i$ AND $C(Y)_i$ => $C(Y)_i$ for all 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
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORA	OR to A	275

SUMMARY: $C(A)_i$ OR $C(Y)_i$ => $C(A)_i$ for all 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORQ	OR to Q	276

SUMMARY: $C(Q)_i$ OR $C(Y)_i$ => $C(Q)_i$ for all i = 0, 1, ..., 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORAQ	OR to AQ	277

SUMMARY: C(AQ)i OR $C(Y\text{-pair})_i$ => $C(AQ)_i$ for all i = 0, 1, ..., 71

MODIFICATIONS: All except DU, DL, CI, SC

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORXn	OR to Xn (n = 0, 1,..., 7)	26n

SUMMARY: $C(Xn)_i$ OR $C(Y)_i$ => $C(Xn)_i$ for all 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORSA	OR to Storage A	255

SUMMARY: $C(A)_i$ OR $C(Y)_i$ => $C(Y)_i$ for all i = 0, 1,...,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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORSQ	OR to Storage Q	256

SUMMARY: $C(Q)_i$ OR $C(Y)_i$ => $C(Y)_i$ for all i = 0, 1, ..., 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORSXn	OR to Storage Xn (n = 0, 1, ..., 7)	24n

SUMMARY: $C(Xn)_i$ OR $C(Y)_i$ => $C(Y)_i$ for all 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
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERA	EXCLUSIVE OR to A	675

SUMMARY: $C(A)_i \not\equiv C(Y)_i \Rightarrow C(A)_i$ 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERQ	EXCLUSIVE OR to Q	676

SUMMARY: $C(Q)_i \not\equiv C(Y)_i \Rightarrow C(Q)_i$ for i = 0, 1,...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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERAQ	EXCLUSIVE OR to AQ	677

SUMMARY: $C(AQ)_i \not\equiv C(Y\text{-pair})_i \Rightarrow C(AQ)_i$ for all 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

Mnemonic:	Name of the Instruction:		Op Code (Octal)
ERXn	EXCLUSIVE OR to Xn	(n = 0, 1,...,17)	66n

SUMMARY: $C(Xn)_i \not\equiv C(Y)_i \Rightarrow C(Xn)_i$ 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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERSA	EXCLUSIVE OR to Storage A	655

SUMMARY: $C(A)_i \not\equiv C(Y)_i \Rightarrow C(Y)_i$ for i = 0, 1,...,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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERSQ	EXCLUSIVE OR to Storage Q	656

SUMMARY: $C(Q)_i \not\equiv C(Y)_i \Rightarrow C(Y)_i$ for i = 0, 1,...,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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERSXn	EXCLUSIVE OR to Storage Xn (n = 0, 1,...,7)	64n

SUMMARY: $C(Xn)_i \not\equiv C(Y)_i \Rightarrow C(Y)_i$ 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
Negative	If $C(Y)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMPA	Compare with A	115

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

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Carry	Algebraic (Signed Fixed-Point) Comparison Relation	Sign
0	0	0	C(A) > C(Y)	$C(A)_0 = 0, C(Y)_0 = 1$
0	0	1	C(A) > C(Y)	$C(A)_0 = C(Y)_0$
1	0	1	C(A) = C(Y)	$C(A)_0 = C(Y)_0$
0	1	0	C(A) < C(Y)	$C(A)_0 = C(Y)_0$
0	1	1	C(A) < C(Y)	$C(A)_0 = 1, C(Y)_0 = 0$

Zero	Carry	Logic (Unsigned Fixed-Point) Comparison Relation
0	0	C(A) < C(Y)
1	1	C(A) = C(Y)
0	1	C(A) > C(Y)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMPQ	Compare with Q	116

SUMMARY: Comparison C(Q) :: C(Y)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Carry	Algebraic (Signed Fixed-Point) Comparison Relation	Sign
0	0	0	$C(Q) > C(Y)$	$C(Q)_0 = 0,\ C(Y)_0 = 1$
0	0	1	$C(Q) > C(Y)$	$C(Q)_0 = C(Y)_0$
1	0	1	$C(Q) = C(Y)$	$C(Q)_0 = C(Y)_0$
0	1	0	$C(Q) < C(Y)$	$C(Q)_0 = C(Y)_0$
0	1	1	$C(Q) < C(Y)$	$C(Q)_0 = 1,\ C(Y)_0 = 0$

Zero	Carry	Logic (Unsigned Fixed-Point) Comparison Relation
0	0	$C(Q) < C(Y)$
1	1	$C(Q) = C(Y)$
0	1	$C(Q) > C(Y)$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMPAQ	Compare with AQ	117

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

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

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Carry	Algebraic (Signed Fixed-Point) Comparison Relation	Sign
0	0	0	C(AQ) > C(Y-pair)	$C(AQ)_0 = 0, C(Y\text{-pair})_0 = 1$
0	0	1	C(AQ) > C(Y-pair)	$C(AQ)_0 = C(Y\text{-pair})_0$
1	0	1	C(AQ) = C(Y-pair)	$C(AQ)_0 = C(Y\text{-pair})_0$
0	1	0	C(AQ) < C(Y-pair)	$C(AQ)_0 = C(Y\text{-pair})_0$
0	1	1	C(AQ) < C(Y-pair)	$C(AQ)_0 = 1, C(Y\text{-pair})_0 = 0$

Zero	Carry	Logic (Unsigned Fixed-Point) Comparison Relation
0	0	C(AQ) < C(Y-pair)
1	1	C(AQ) = C(Y-pair)
0	1	C(AQ) > C(Y-pair)

Mnemonic:	Name of the Instruction:		Op Code (Octal)
CMPXn	Compare with Xn	(n = 0, 1,...,7)	10n

SUMMARY: Comparison $C(Xn) :: C(Y)_{0-17}$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Carry	Algebraic (Signed Fixed-Point) Comparison Relation	Sign
0	0	0	$C(Xn) > C(Y)_{0-17}$	$C(Xn)_0 = 0, C(Y)_0 = 1$
0	0	1	$C(Xn) > C(Y)_{0-17}$	$\left.\right\}$ $C(Xn)_0 = C(Y)_0$
1	0	1	$C(Xn) = C(Y)_{0-17}$	
0	1	0	$C(Xn) < C(Y)_{0-17}$	
0	1	1	$C(Xn) < C(Y)_{0-17}$	$C(Xn)_0 = 1, C(Y)_0 = 0$

Zero	Carry	Logic (Unsigned Fixed-Point) Comparison Relation
0	0	$C(Xn) < C(Y)_{0-17}$
1	1	$C(Xn) = C(Y)_{0-17}$
0	1	$C(Xn) > C(Y)_{0-17}$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CWL	Compare with Limits	111

SUMMARY: Algebraic comparison of C(Y) with the closed interval [C(A); C(Q)] and also with the number C(Q)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Y) is contained in the closed interval [C(A); C(Q)], i.e., either $C(A) \leqq C(Y) \leqq C(Q)$ or $C(A) \geqq C(Y) \geqq C(Q)$, then ON; otherwise OFF

Negative	Carry	Relation between C(Q) and C(Y)	Signs of C(Q) and C(Y)
0	0	$C(Q) > C(Y)$	$C(Q)_0 = 0$, $C(Y)_0 = 1$
0	1	$C(Q) \geqq C(Y)$	$C(Q)_0 = C(Y)_0$
1	0	$C(Q) \leqq C(Y)$	
1	1	$C(Q) < C(Y)$	$C(Q)_0 = 1$, $C(Y)_0 = 0$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMG	Compare Magnitude	405

SUMMARY: Algebraic comparison $|C(A)| :: |C(Y)|$

MODIFICATION: All

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$\|C(A)\| > \|C(Y)\|$
1	0	$\|C(A)\| = \|C(Y)\|$
0	1	$\|C(A)\| < \|C(Y)\|$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SZN	Set Zero and Negative Indicators from Memory	234

SUMMARY: Test the number C(Y)

MODIFICATION: All

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	Number $C(Y) > 0$
1	0	Number $C(Y) = 0$
0	1	Number $C(Y) < 0$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMK	Compare Masked	211

SUMMARY: $Z_i = \overline{C(Q)_i}$ AND $[C(A)_i \not\equiv C(Y)_i]$ for all i = 0, 1,..., 35

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If Z = 0, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

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

2. The zero indicator is set ON, if the comparison is successful for all bit positions; i.e., if for all i = 0, 1,..., 35 there is

either $C(A)_i \equiv C(Y)_i$ or $C(Q)_i = 1$

(identical) (masked)

Otherwise Zero indicator is set OFF.

3. The negative indicator is set ON if the comparison is unsuccessful for bit position 0, i.e., if

$C(A)_0 \not\equiv C(Y)_0$ as well as $C(Q)_0 = 0$

(nonidentical) (nonmasked)

Otherwise negative indicator is set OFF.

(Revised February 26, 1971)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CANA	Comparative AND with A	315

SUMMARY: $Z_i = C(A)_i$ AND $C(Y)_i$ for all $i = 0, 1, \ldots, 35$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CANQ	Comparative AND with Q	316

SUMMARY: $Z_i = C(Q)_i$ AND $C(Y)_i$ for all $i = 0, 1, \ldots, 35$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CANAQ	Comparative AND with AQ	317

SUMMARY: $Z_i = C(AQ)_i$ AND $C(Y\text{-pair})_i$ for all $i = 0, 1, \ldots, 71$

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

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CANXn	Comparative AND with Xn (n = 0, 1,...,7)	30n

SUMMARY: $Z_i = C(Xn)_i$ AND $C(Y)_i$ for all $i = 0, 1, \ldots, 17$

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CNAA	Comparative NOT with A	215

SUMMARY: $Z_i = C(A)_i$ AND $\overline{C(Y)_i}$ for all $i = 0, 1, \ldots, 35$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CNAQ	Comparative NOT with Q	216

SUMMARY: $Z_i = C(Q)_i$ AND $\overline{C(Y)_i}$ for all $i = 0,1,\ldots,35$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CNAAQ	Comparative NOT with AQ	217

SUMMARY: $Z_i = C(AQ)_i$ AND $\overline{C(Y\text{-pair})_i}$ for all i = 0, 1,...,71

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

INDICATORS: (Indicators not listed are not affected)

Zero	If Z = 0, then ON; otherwise OFF
Negative	If Z_0 = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CNAXn	Comparative NOT with Xn	20n

SUMMARY: $Z_i = C(Xn)_i$ AND $\overline{C(Y)_i}$ for all i = 0, 1,...,17

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	If Z = 0, then ON; otherwise OFF
Negative	If Z_0 = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TRA	Transfer Unconditionally	710

SUMMARY: EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTES:

1. The new effective address replaces the C(ICTC), and the new pointer in TBR formed during the appending process for the transfer address replaces C(PBR). Pointers in TBR may come from the following sources:

   a. C(PBR) when bit 29 of instruction word = 0

   b. C(ABRm) when bit 29 of instruction word = 1

   c. C(ABRm) when designated by an ITB modifier in the indirect word (m is an external ABR)

   d. C(TBR) when brought in as a result of an ITS modifier in the indirect word

2. All successful transfers depend upon normal access restrictions, and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TSBn	Transfer and Set Base n (n=0, 1,...,7)	270-273 670-673

SUMMARY: $C(ICTC) + 00...01 \Rightarrow C(ABRn)_{0-17}$; $C(PBR) \Rightarrow C(ABRm)_{0-17}$; $EA \Rightarrow C(ICTC)$; $P \Rightarrow C(PBR)$ where n and m are the designated internal and the linked external ABR's respectively.

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

INDICATORS: None affected

NOTES:

1. If the ABR SPECIFIED BY THE TSBn instruction is external, that is, $C(ABRn)_{21} = 1$; then $C(ICTC) + 00...01 \Rightarrow C(ABRn)_{0-17}$ does not take place. The pointer formed during the appending process for the transfer address replaces C(PBR). Pointers in TBR may come from the following sources:

   a. C(PBR) when bit 29 of instruction word = 0

   b. C(ABRm) when bit 29 of instruction word = 0

   c. C(ABRm) when designated by an ITB modifier in the direct word (m is an external ABR)

   d. C(TBR) when brought in as a result of an ITS modifier in the indirect word

2. This instruction may be executed in Master or Slave mode. If attempted in Slave mode an illegal procedure fault will be generated unless $C(ABRn,m)_{22} = 0$. The ABR's will be affected in the following manner:

   a. If $C(ABRn)_{22} = 1$, the fault is generated and ABRn, m are not changed

   b. If $C(ABRn)_{22} = 0$ and $C(ABRm)_{22} = 1$, then ABRn is loaded, and the fault is generated for ABRm

3. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TSXn	Transfer and Set Index Register n n (n=0, 1,...,7)	70n

SUMMARY: C(ICTC) + 0...01 => C(Xn) EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTES: 1. The new effective address replaces the C(ICTC), and the new pointer in TBR formed during the appending process for the transfer address replaces C(PBR). Pointers in TBR may come from the following sources:

a. C(PBR) when bit 29 of instruction word = 0

b. C(ABRm) when bit 29 of instruction word = 1 (m is an external ABR)

c. C(ABRm) when designated by an ITB modifier in the indirect word)

d. C(TBR) when brought in as a result of an ITS modifier in the indirect word

2. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TSS	Transfer and Set Slave	715

SUMMARY: EA => C(ICTC), C(TBR) => C(PBR), Reset Absolute indicator

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

INDICATORS: None affected

NOTES: 1. The new effective address replaces the C(ICTC), and the new pointer in TBR formed during the appending process for the transfer address replaces C(PBR). Pointers in TBR may come from the following sources:

a. C(PBR) when bit 29 of instruction word = 0

b. C(ABRm) when bit 29 of instruction word = 1

c. C(ABRm) when designated by an ITB modifier in the indirect word (m is an external ABR)

d. C(TBR) when brought in as a result of an ITS modifier in the indirect word

2. If this instruction is attempted in Slave mode a 635/645 compatibility fault will occur. When this instruction is executed in Master mode, the absolute indicator is reset just before the fetch of the new (transferred) instruction unless bit 29 of TSS is ON, or ITB or ITS indirection is specified. Therefore, it is recommended that TSS not be used to get out of Absolute mode.

3. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RET	Return	630

SUMMARY: $C(Y)_{0-17}$ => C(ICTC); $C(Y)_{18-28}$ => C(IR); $C(Y)_{29-35}$ are not used

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

INDICATORS: (Indicators not listed are not affected)

Absolute Mode	If corresponding bit in C(Y) is 1, and processor is in Procedure-Master mode, then ON; otherwise OFF.
All other Indicators	If corresponding bit in C(Y) is 1, then ON; otherwise OFF.

NOTES: 1. The contents of the location specified by Y replaces the contents of the instruction counter and indicator register. A possible change in status of the Absolute mode indicator takes place as the last part of the instruction execution. The relationship between $C(Y)_{18-28}$ and the indicators is as follows:

Bit Position	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Absolute Mode

2. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RTCD	Return Double	610

SUMMARY: $C(Y)_{0-17}$ => C(PBR); $C(Y)_{18-35}$ is ignored
$C(Y+1)_{0-17}$ => C(ICTC); $C(Y+1)_{18-28}$ => C(IR); $C(Y)_{29-35}$ are not used

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

INDICATORS: (Indicators not listed are not affected)

Absolute Mode	If corresponding bit in C(Y+1) is 1, and processor is in Procedure-Master mode, then ON; otherwise OFF.
All other Indicators	If corresponding bit in C(Y+1) is 1, then ON; otherwise OFF.

NOTES: 1. The contents of the location specified by Y+1 replaces the contents of the instruction counter and indicator register. A possible change in status of the Absolute mode indicator takes place as the last part of the instruction execution. The relationship between $C(Y+1)_{18-28}$ and the indicators is as follows:

Bit Position	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Absolute Mode

2. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TZE	Transfer on Zero	600

SUMMARY: If zero indicator ON, then EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTES:

1. The new effective address replaces the C(ICTC), and the new pointer in TBR formed during the appending process for the transfer address replaces C(PBR). Pointers in TBR may come from the following sources:

   a. C(PBR) when bit 29 of instruction word = 0
   b. C(ABRm) when bit 29 of instruction word = 1 (m is an external ABR)
   c. C(ABRm) when designated by an ITB modifier in the indirect word (m is an external ABR)
   d. C(TBR) when brought in as a result of an ITS modifier in the indirect word

2. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TNZ	Transfer on Not Zero	601

SUMMARY: If zero indicator OFF, then EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTES: Same as for TZE instruction, above.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TMI	Transfer on Minus	604

SUMMARY: If negative indicator ON, then EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTES: Same as for TZE instruction, above.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TPL	Transfer on Plus	605

SUMMARY: If negative indicator OFF, then EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTES: 1. The new effective address replaces the C(ICTC), and the new pointer in TBR formed during the appending process for the transfer address replaces C(PBR). Pointers in TBR may come from the following sources:

a. C(PBR) when bit 29 of instruction word = 0

b. C(ABRm) when bit 29 of instruction word = 1 (m is an external ABR)

c. C(ABRm) when designated by an ITB modifier in the indirect word (m is an external ABR)

d. C(TBR) when brought in as a result of an ITS modifier in the indirect word

2. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TRC	Transfer on Carry	603

SUMMARY: If Carry indicator ON, then EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTE: Same as for the TPL instruction on page 2-99.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TNC	Transfer on No Carry	602

SUMMARY: If Carry indicator OFF, then EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTE: Same as for the TPL instruction on page 2-99.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TOV	Transfer on Overflow	617

SUMMARY: If overflow indicator ON, then EA => C(ICTC), C(TBR) => C(PBR)

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

INDICATORS: (Indicators not listed are not affected)

Overflow	Set OFF

NOTES: 1. The new effective address replaces the C(ICTC), and the new pointer in TBR formed during the appending process for the transfer address replaces C(PBR). Pointers in TBR may come from the following sources:

a. C(PBR) when bit 29 of instruction word = 0

b. C(ABRm) when bit 29 of instruction word = 1 (m is an external ABR)

c. C(ABRm) when designated by an ITB modifier in the indirect word (m is an external ABR)

d. C(TBR) when brought in as a result of an ITS modifier in the indirect word

2. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TEO	Transfer on Exponent Overflow	614

SUMMARY: If exponent overflow indicator ON, then EA => C(ICTC); C(TBR) => C(PBR)

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

INDICATORS: (Indicators not listed are not affected)

Exponent Overflow	Set OFF

NOTES: Same as for the TOV instruction above.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TEU	Transfer on Exponent Underflow	615

SUMMARY: If exponent underflow indicator ON, then EA => C(ICTC); C(TBR) => C(PBR)

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

INDICATORS: (Indicators not listed are not affected)

Exponent Underflow	Set OFF

NOTE: Same as for the TOV instruction on preceding page.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TTF	Transfer on Tally Runout Indicator Off	607

SUMMARY: If Tally Runout indicator OFF, then EA => C(ICTC); C(TBR) => C(PBR)

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

INDICATORS: None affected

NOTES:

1. The new effective address replaces the C(ICTC), and the new pointer in TBR formed during the appending process for the transfer address replaces C(PBR). Pointers in TBR may come from the following sources:

   a. C(PBR) when bit 29 of instruction word = 0

   b. C(ABRm) when bit 29 of instruction word = 1 (m is an external ABR)

   c. C(ABRm) when designated by an ITB modifier in the indirect word (m is an external ABR)

   d. C(TBR) when brought in as a result of an ITS modifier in the indirect word

2. All successful transfers depend upon normal access restrictions and are subject to linkage faults and directed faults.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
BCD	Binary to Binary-Coded-Decimal	505

SUMMARY: C(A) divided by C(Y) => 4-bit quotient and remainder. Shift C(Q) left six positions; 4-bit quotient => $C(Q)_{32-35}$ and remainder => C(A). Shift C(A) left three positions.

MODIFICATIONS: All except IT categories CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON
Negative	If before execution $C(A)_0 = 1$, then ON; otherwise OFF

NOTES:

1. This instruction carries out one step in an algorithm for the conversion of a number from the binary to the decimal system of notation, which requires the repeated short division of the binary number or last remainder by certain constants.
   $C_i = 8^i \times 10^{n-1}$ (for i=1,2...)
   with n being defined by: $10^{n-1} \leq |\text{number}| \leq (10^n) -1$

2. The values in the table that follows are the conversion constants to be used with the Binary to 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 to its decimal equivalent. The instruction is executed once per digit, using the constant appropriate to the conversion step wtth each execution.

3. 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 conversion, the contents of the accumulator are shifted right 3 positions, the constants in the conversion step 1 row may be used one at a time in order of decreasing value until the conversion is complete.

4. See diagram on the following page.

BINARY TO BCD CONVERSION CONSTANTS

Conversion Step \ Starting Range of C(AR)	$-10^{10}+1 \rightarrow 10^{10}-1$	$-10^{9}+1 \rightarrow 10^{9}-1$	$-10^{8}+1 \rightarrow 10^{8}-1$	$-10^{7}+1 \rightarrow 10^{7}-1$	$-10^{6}+1 \rightarrow 10^{6}-1$	$-10^{5}+1 \rightarrow 10^{5}-1$	$-10^{4}+1 \rightarrow 10^{4}-1$	$-10^{3}+1 \rightarrow 10^{3}-1$	$-10^{2}+1 \rightarrow 10^{2}-1$	$-10^{1}+1 \rightarrow 10^{1}-1$
1	8^1x10^9	$8\ x10^8$	$8\ x10^7$	$8\ x10^6$	$8\ x10^5$	$8\ x10^4$	$8\ x10^3$	$8\ x10^2$	$8\ x10^1$	8
2	8^2x10^8	8^2x10^7	8^2x10^6	8^2x10^5	8^2x10^4	8^2x10^3	8^2x10^2	8^2x10^1	8^2	
3	8^3x10^7	8^3x10^6	8^3x10^5	8^3x10^4	8^3x10^3	8^3x10^2	8^3x10^1	8^3		
4	8^4x10^6	8^4x10^5	8^4x10^4	8^4x10^3	8^4x10^2	8^4x10^1	8^4			
5	8^5x10^5	8^5x10^4	8^5x10^3	8^5x10^2	8^5x10^1	8^5				
6	8^6x10^4	8^6x10^3	8^6x10^2	8^6x10^1	8^6					
7	8^7x10^3	8^7x10^2	8^7x10^1	8^7						
8	8^8x10^2	8^8x10^1	8^8							
9	8^9x10^1	8^9								
10	8^{10}									

Example:

```
01      LDX2    0,DU                            PLACE ZEROS IN X2
02      LDA     X                               LOAD ACCUMULATOR WITH VALUE TO BE CONVERTED
03      RPT     6,1                             REPEAT 6 TIMES, INCREMENT BY 1
04      BCD     TAB,2                           DIVIDE BY TAB, TAB+1, ETC.
05      STQ     Y                               STORE CONVERTED NUMBER IN Y
         .
         .
         .
         .
06TAB   DEC     800000, 640000, 512000, 409600, 327680
        DEC     262144
```

5. Because there is a limit on range, a maximum of 10 digits may be converted correctly.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CAM	Clear Associative Memory	532

SUMMARY: 0 => $C(AR)_{54}$; usage value (0000-1111) => $C(AR)_{56-59}$ for all 16 AR's

MODIFICATIONS: None

INDICATORS: None affected

NOTE: Execution of the CAM instruction sets the empty/full bit in each AR to zero; that is, empty, and initializes the usage value of each AR by arbitrarily assigning a unique number in the range 0-15 to each register. If this instruction is attempted in Slave mode, an illegal procedure fault will be generated.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
GTB	Gray to Binary	774

SUMMARY: C(A) converted from Gray Code to binary representation => C(A)

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

NOTE: This conversion is defined by the following algorithm where R_i and S_i denote the contents of bit positions i of the A-register before and after conversion.

$S_0 = R_0 \quad S_i = (R_i \text{ AND } \overline{S_{i-1}}) \text{ OR } (\overline{R_i} \text{ AND } S_{i-1})$
for i = 1,2,...,35.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CIOC	Connect I/O Channel	015

SUMMARY: C(Y) are transferred from the memory module via the memory module port that is specified by C(Y).

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

INDICATORS: None affected

NOTES:

1. The absolute address Y is used to access the memory location as usual. However, the memory module does not transmit the contents of this location to the processor that submitted the absolute address; it uses $C(Y)_{33-35}$ to select one of its eight ports, sends a connect pulse to the unit on this port, and then transmits C(Y). C(Y) must be a 0 modulo 8 to be independent of interlace.
2. This instruction can be used in the Master mode only. If the use of this instruction is attempted by a processor that is in the Slave mode an illegal memory command fault will occur.
3. A connect command is sent out the processor port selected by the 24-bit absolute address formed by this instruction.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DIS	Delay Until Interrupt Signal	616

SUMMARY: No operation takes place, and the processor does not continue with the next instruction but waits for a program interrupt signal.

MODIFICATIONS: None

INDICATORS None affected

NOTES:

1. This instruction can be used in the Master mode only or a 635/645 compatibility fault will occur.
2. The inhibit bit (bit 28 of the instruction word) will <u>not</u> inhibit interrupts for this instruction.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DRL	Derail	002

SUMMARY: Causes a fault which obtains and executes in the Absolute mode the two instructions stored at memory locations 4 + C and 5 + C decimal; the constant C being set up in the maintenance panel FAULT VECTOR switches.

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: The DRL instruction itself does not affect any indicator. However, the execution of the two instructions from 4 + C and 5 + C may affect indicators; each one in turn will affect the absolute mode indicator as follows:

Absolute Mode	If the instruction obtained actually results in a transfer of control and is not the TSS instruction, then ON. If the instruction obtained is either the RET or RTCD instruction with bit 29 = zero, or the TSS instruction, then OFF.

NOTES: Execution of the DRL instruction implies the following conditions:

1. During the execution of this DRL instruction and the two instructions obtained, the processor is in the Absolute mode, independent of the value of its Absolute indicator. The processor will stay in the Absolute mode if the Absolute indicator is ON after the execution of these three instructions.

2. The instruction from 4 + C must not alter the memory location 5 + C, and must not be an XED instruction.

3. If the instruction from 4 + C alters the contents of the instruction counter, then this transfer of control is effective immediately; and the instruction from 5 + C is not executed.

4. After the execution of the two instructions obtained from Y-pair, 4 + C and 5 + C, the next instruction to be executed is obtained from C(ICTC) + 1. This is the instruction stored in the memory right after this DRL instruction unless the contents of the instruction counter have been changed by the execution of the two instructions obtained from 4 + C and 5 + C.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MME	Master Mode Entry 1	001

SUMMARY: Causes a fault which obtains and executes in the Absolute mode the two instructions stored at the memory locations 2 + C and 3 + C decimal; the constant C is set up on the maintenance Panel FAULT VECTOR switches.

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: The MME instruction itself does not affect any indicator. However, the execution of the two instructions from 2 + C and 3 + C may affect indicators; each one in turn will affect the Absolute mode indicator as shown.

Absolute Mode	If the instruction obtained actually results in a transfer of control and is not the TSS instruction, then ON. If the instruction obtained is either the RET or RTCD instruction with bit 29 = zero, or the TSS instruction, then OFF.

NOTES: Execution of the MME instruction implies the following conditions:

1. During the execution of this MME instruction and the two instructions obtained, the processor is in the Absolute mode independent of the value of its Absolute mode indicator. The processor will stay in Absolute mode if the Absolute mode indicator is set ON after the execution of these three instructions.

2. The instruction from 2 + C must not alter the memory location 3 + C, and must not be an XED instruction.

3. If the instruction from 2 + C alters the contents of the instruction counter, then this transfer or control is effective immediately and the instruction from 3 + C is not executed.

4. After the execution of the two instructions obtained from Y-pair, 2 + C and 3 + C, the next instruction to be executed is obtained from C(ICTC)+1. This is the instruction stored in memory right after this MME instruction unless the contents of the instruction counter have been changed by the execution of the two instructions obtained from 2 + C and 3 + C.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MME2	Master Mode Entry 2	004

SUMMARY: Causes a fault which obtains and executes in the Absolute mode the two instructions stored at the memory locations 8 + C and 9 + C decimal; the constant C is set up on the maintenance panel FAULT VECTOR switches.

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: The MME2 instruction itself does not affect any indicator. However, the execution of the two instructions from 8 + C, and 9 + C may affect indicators, particularly each one in turn will affect the Absolute mode indicator as shown.

Absolute Mode	If the instruction obtained actually results in a transfer of control and is not the TSS instruction, then ON. If the instruction obtained is either the RET or RTCD instruction with bit 29 = zero, or the TSS instruction, then OFF.

NOTES: Execution of the MME2 instruction implies the following conditions:

1. During the execution of this MME2 instruction and the two instructions obtained, the processor is in the Absolute mode independent of the value of its Absolute mode indicator. The processor will stay in Absolute mode if the absolute mode indicator is set ON after the execution of these three instructions.

2. The instruction from 8 + C must not alter the memory location 9 + C, and must not be an XED instruction.

3. If the instruction from 8 + C alters the contents of the instruction counter, then this transfer of control is effective immediately and the instruction from 9 + C is not executed.

4. After the execution of the two instructions obtained from Y-pair, 8 + C and 9 + C, the next instruction to be executed is obtained from C(ICTC)+1. This is the instruction stored in memory right after this MME2 instruction unless the contents of the instruction counter have been changed by the execution of the two instructions obtained from 8 + C and 9 + C.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MME3	Master Mode Entry 3	005

SUMMARY: Causes a fault which obtains and executes in the Absolute mode the two instructions stored at memory locations 10 + C and 11 + C decimal; the constant C is set up on the maintenance panel FAULT VECTOR switches.

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: The MME3 instruction itself does not affect any indicator. However, the execution of the two instructions from 10 + C and 11 + C may affect indicators; each one in turn will affect the Absolute mode indicator as shown.

Absolute Mode	If the instruction obtained actually results in a transfer of control and is not the TSS instruction, then ON. If the instruction obtained is either the RET or RTCD instruction with bit 29 = zero, or the TSS instruction, then OFF.

NOTES: Execution of the MME3 instruction implies the following conditions:

1. During the execution of this MME3 instruction and the two instructions obtained, the processor is in the Absolute mode independent of the value of its Absolute mode indicator. The processor will stay in Absolute mode if the Absolute mode indicator is set ON after the execution of these three instructions.

2. The instruction from 10 + C must not alter the memory location 11 + C and must not be an XED instruction.

3. If the instruction from 10 + C alters the contents of the instruction counter, then this transfer of control is effective immediately and the instruction from 11 + C is not executed.

4. After the execution of the two instructions obtained from Y-pair, 10 + C and 11 + C, the next instruction to be executed is obtained from C(ICTC)+1. This is the instruction stored in memory right after this MME3 instruction unless the contents of the instruction counter have been changed by the execution of the two instructions obtained from 10 + C and 11 + C.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MME4	Master Mode Entry 4	007

SUMMARY: Causes a fault which obtains and executes in the Master mode the two instructions stored at the memory locations 14 + C and 15 + C decimal; the constant C is set up on the maintenance panel FAULT VECTOR switches.

MODIFICATIONS: All Except CI, SC, SCR

INDICATORS: The MME4 instruction itself does not affect any indicator. However, the execution of the two instructions from 14 + C and 15 + C may affect indicators, each one in turn will affect the Absolute mode indicator as shown.

Absolute Mode	If the instruction obtained actually results in a transfer of control and is not the TSS instruction, then ON. If the instruction obtained is either the RET or RTCD instruction with bit 29 = zero, or the TSS instruction, then OFF.

NOTES: Execution of the MME4 instruction implies the following conditions:

1. During the execution of this MME4 instruction and the two instructions obtained, the processor is in the Absolute mode independent of the value of its Absolute mode indicator. The processor will stay in Absolute mode if the Absolute mode indicator is set ON after the execution of these three instructions.

2. The instruction from 14 + C must not alter the memory location 15 + C and must not be an XED instruction.

3. If the instruction from 14 + C alters the contents of the instruction counter, then this transfer of control is effective immediately and the instruction from 15 + C is not executed.

4. After the execution of the two instructions obtained from Y-pair, 14 + C and 15 + C, the next instruction to be executed is obtained from C(ICTC) + 1. This is the instruction stored in memory right after this MME4 instruction unless the contents of the instruction counter have been changed by the execution of the two instructions obtained from 14 + C and 15 + C.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
XEC	Execute	716

SUMMARY: Obtain and execute the instruction stored at the memory location Y

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

INDICATORS: Indicators not listed are not affected

NOTES:

1. The XEC instruction itself does not affect any indicator. However, the execution of the instruction from Y may affect indicators.

2. After the execution of the instruction obtained from location Y, the next instruction to be executed is obtained from C(ICTC)+1; the one stored in memory right after this XEC instruction, unless the contents of the instruction counter have been changed by the execution of the instruction obtained from memory location Y.

3. To Execute (XEC) a Repeat Double (RPD) instruction, the XEC instruction must be in an odd location.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
XED	Execute Double	717

SUMMARY: Obtain and execute the two instructions stored at the memory Y-pair locations

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

INDICATORS: Indicators not listed are not affected

NOTES:

1. The XED instruction itself does not affect any indicator. However, the execution of the two instructions from Y-pair may affect indicators.

2. The first instruction obtained from Y-pair MUST NOT alter the memory location from which the second instruction is obtained, and _must_ _not_ be another XED instruction.

3. If the first instruction obtained from Y-pair alters the contents of the instruction counter, then this transfer of control is effective immediately; and the second instruction of the pair is not executed.

4. After the execution of the two instructions obtained from Y-pair, the next instruction to be executed is obtained from C(ICTC)+1. This is the instruction stored in memory right after this XED instruction unless the contents of the instruction counter have been changed by the execution of the two instructions obtained from the memory locations Y-pair.

5. To Execute Double (XED) a pair which has Repeat Double (RPD) as the odd instruction of the pair, XED must be located at the odd address.

6. If RPD is specified anywhere within a sequence of XED's, the _original_ and _all_ _subsequent_ XED's in the sequence must be in odd locations.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RMCM	Read Memory Controller Mask Register	233

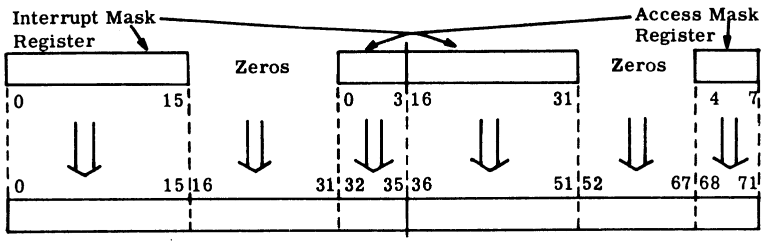
SUMMARY: C(Memory Controller Interrupt Mask Register)
C(Memory Controller Access Mask Register)
of Memory Unit Specified
=> C(AQ)

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: 1. The absolute address Y generated for this instruction is used in selecting a processor port as with a normal memory access request. However, the selected module does not transmit the contents of an addressed memory location, but the contents of its Memory Controller Interrupt Mask Register and Memory Controller Access Mask Register.



Combined AQ-register

2. When selecting a memory module; memory size, instruction word address field, PBR, TBR, internal and external ABR's, and memory interlace (if the final address is not 0 mod 8) must be considered.

3. This instruction can be used in the Master mode only. If the use of this instruction is attempted by a processor that is in the Slave mode a 635/645 compatibility fault will occur.

4. The memory command actually sent out the processor port is RMSK.

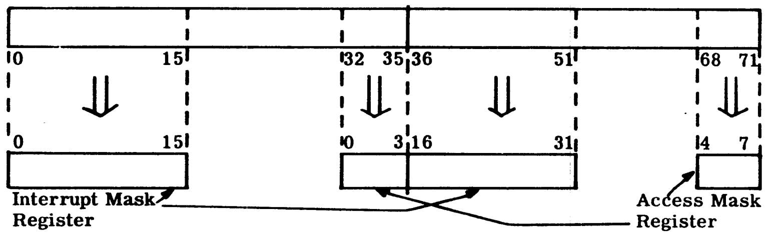
Mnemonic:	Name of the Instruction:	Op Code (Octal)
SMCM	Set Memory Controller Mask Register	553

SUMMARY: C(AQ) => { C(Memory Controller Interrupt Mask Register) C(Memory Controller Access Mask Register) of Memory Unit specified

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

INDICATORS: None affected

NOTES: 1. The absolute address Y generated for this instruction is used in selecting a processor port as with a normal memory access request. However, the selected module does not store the data received in a memory location but in its Memory Controller Interrupt Mask Register.



Combined AQ-register

2. When selecting a memory module; memory size, instruction word address field, PBR, TBR, internal and external ABR's, and memory interlace (if the final address is not 0 mod 8) must be considered.

3. This instruction can be used in the Master mode only. If the use of this instruction is attempted by a processor that is in the Slave mode a 635/645 compatibility fault will occur.

4. The command actually sent out the processor port is SMSK.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SMIC	Set Memory Controller Interrupt Cells	451

SUMMARY: C(A) is used to set selected Interrupt Cells ON in the System Controller of the memory unit selected by Y_{0-2}

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

INDICATORS: None affected

NOTES:

1. The absolute address Y generated for this instruction is used in selecting a processor port as with a normal memory access request. However, the selected module does not store the data received in a memory location, but uses it to set selected Interrupt Cells ON.

   For $i = 0, 1, \ldots, 15$ AND $C(A)_{35} = 0$:
   if $C(A)_i = 1$, then set Interrupt Cell (i) ON

   For $i = 0, 1, \ldots, 15$ AND $C(A)_{35} = 1$:
   if $C(A)_i = 1$, then set Interrupt Cell (16+i) ON.

2. When selecting a memory module; memory size, instruction word address field, PBR, TBR, internal and external ABR's and memory interlace (if the final address is not 0 modulo 8) must be considered.

3. This instruction can be used in the Master mode only or a 635/645 compatibility fault will occur.

4. The command actually sent out the processor port is SXC.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RPT	Repeat	520

SUMMARY: Execute the next instruction a specified number of times or until a specified Terminate condition is met.

MODIFICATIONS: None

INDICATORS:

Tally Runout	If termination because of Tally = 0, then ON If because Terminate condition is met, then OFF
All other indicators	The RPT instruction itself does not affect any of the other indicators. However, the execution of the repeated instruction may affect indicators.

NOTES: 1. This RPT instruction has the following format:

0 7	8	9	10	11 17	18 26	27	28	29	30 35
Tally			C	Term.Cond.	Op Code	0	1	0	Delta

2. If C = 1, then bits 0-17 of the RPT instruction => X0. If C = 0, then X0 contains whatever was left from the previous instruction.

3. In any case, the Terminate condition and Tally from X0 will control the repetition loop for the instruction following this RPT instruction; initial Tally = 0 will be interpreted as 256.

4. The repetition loop consists of the following steps:

   a. Execute the repeated instruction,
   b. $C(X0)_{0-7}$ -1 => $C(X0)_{0-7}$
   c. If Termination condition met then terminate,
   d. If $C(X0)_{0-7}$ = 0, then set Tally Runout indicator ON and terminate;
   e. Go to 4a.

5. All instructions can be used as repeated instructions except the following:

   a. All control instructions
   b. All special instruction operations except BCD
   c. All general base instructions--LBRm, EABm, ADBm, SBRm, LDBR,

(Continued)

CAM, EAPm, LDCF, RTCD, STCD, STPm, ZAM, TSBm, SCU, RCU, LDB

6. Address modification for the repeated instruction:

   For the repeated instruction, only the modifiers R and RI are permitted, and only the designators specifying X1,...,X7. The effective address EA (in the case of R) or the address EA of the indirect word to be referenced (in the case of RI) will be:

   a. For the first execution of the repeated instruction:

      Y + C(R) => EA, EA => C(R)

   b. For any successive execution:

      C(R) + Delta => EA, EA => C(R)

   In the case of RI, only one indirect reference will be made per repeated execution. The tag portion of the indirect word will not be interpreted as usual but will be ignored; and instead the modifier R and the designator R = N will be applied.

7. The Terminate conditions:

   The possible terminate conditions are the same for all three repeat instructions -- RPT, RPD, RPL, except that RPL can terminate on a link address = 0.

   The bit configuration in bit positions 11-17 of the RPT instruction defines the terminate conditions for which the repetition loop will be terminated immediately. If more than one condition is specified, the repeat will terminate if any of the specified conditions are met.

   Bit 17 = 0: any overflow is completely ignored, that is, neither the respective overflow indicator is set ON, nor an overflow trap occurs.

   = 1: any overflow terminates the repetition loop, and it is treated as usual; that is, the respective overflow indicator is set ON, and if the Overflow Mask indicator is OFF, then an overflow fault occurs.

   Bit 16 = 1: if Carry indicator is OFF, then terminate the repetition loop.

   Bit 15 = 1: if Carry indicator is ON, then terminate the repetition loop.

   Bit 14 = 1: if negative indicator is OFF, then terminate the repetition loop.

   Bit 13 = 1: if negative indicator is ON, then terminate the repetition loop.

   Bit 12 = 1: if zero indicator is OFF, then terminate the repetition loop.

(Continued)

Bit 11 = 1: if zero indicator is ON, then terminate the repetition loop.

A "0" in both bit positions for one indicator will cause this indicator to be ignored as a terminate condition; a "1" in both bit positions will cause a termination after the first execution of the repeated instruction.

8. At the time of termination:

$X0_{0-7}$ will contain the Tally Residue; that is, the number of repeats remaining until a Tally Runout would have occurred.

If the RPT instruction is interrupted before termination, the Tally Runout indicator will be OFF.

The Xn specified by the designator of the repeated instruction will contain the effective address of the next operand or indirect word that would have been secured (this is because of the overlap between an execution of the repeated instruction and the address modification for the next execution of the repeated instruction).

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RPD	Repeat Double	560

SUMMARY: Execute the pair of instructions from the next location Y-pair a specified number of times or until a specified terminate condition is met.

MODIFICATIONS: None

INDICATORS:

Tally Runout	If termination of Tally = 0, then ON If terminate condition is met, then OFF
All other indicators	The RPD instruction itself does not affect any of the other indicators. However, the execution of the repeated instructions may affect indicators.

NOTES: 1. The RPD instruction must be stored in an odd memory location except when accessed via the XEC instruction in which case the RPD instruction can be either even or odd.

2. This RPD instruction has the following format:

0 7	8	9	10	11 17	18 26	27	28	29	30 35
Tally	A	B	C	Term.Cond.	Op Code	0	1	0	Delta

3. If C = 1, then bits 0-17 of the RPD instruction => X0. If C = 0, then X0 contains whatever was left from the previous instruction.

4. The terminate condition and Tally from X0 will control the repetition loop for the instruction following this RPD instruction; initial Tally = 0 will be interpreted as 256.

5. The repetition cycle consists of the following steps:

a. Execute the pair of repeated instructions
b. $C(X0)_{0-7}-1 \Rightarrow C(X0)_{0-7}$
c. If termination condition met after termination of odd instruction then terminate.
d. If $C(X0)_{0-7} = 0$, then set Tally Runout indicator ON and terminate.
e. Go to 5a.

(Continued)

6. Note that if an overflow fault occurs on the even instruction, this precludes execution of the odd instruction.

7. All instructions can be used as repeated instructions except the following:

   a. All control instructions.
   b. All special operations instructions except BCD.
   c. All general base instructions--LBRm, EABm, ADBm, SBRm, LDBR, SDBR, CAM, EAPm, LDCF, RTCD, STCD, STPm, ZAM, TSBm, SCU, RCU, LDB, STB, SAM.

8. Address Modification for the pair of repeated instructions:

   For each of the two repeated instructions, only the modifiers R and RI are permitted, and only the designators specifying X1,...,X7.

   The effective address EA (in the case of R) or the address EA of the indirect word to be referenced (in the case of RI) will be:

   a. For the first execution of each of the two repeated instructions

      Y + C(R) => EA, EA => C(R)

   b. For any successive execution of

      The first of the two repeated instructions

      if A=1, then C(R) + Delta => EA, EA => C(R) or
      if A=1, then C(R) => EA

      The second of the two repeated instructions

      if B=1, then C(R) + Delta => EA, EA => C(R) or
      if B=0, then C(R) => EA

      (A and B are the contents of bit positions 8 and 9 of the RPD instruction)

      In the case of RI, only one indirect reference will be made per repeated execution. The tag portion of the indirect word will not be interpreted as usual, but will be ignored; and instead the modifier R and the designator R=N will be applied.

9. The terminate conditions:

   The possible terminate conditions are the same for all three repeat instructions - RPT, RPD, RPL except that RPL may be terminated on a link address = 0.

   The bit configuration in bit positions 11-17 of the RPD instruction defines the terminate conditions for which the repetition loop will be terminated upon completion of the odd instruction. If more than one condition is specified, the repeat will terminate if any of the specified conditions are met.

(Continued)

Bit 17 = 0: any overflow is completely ignored; that is, neither the respective overflow indicator is set ON, nor an overflow fault occurs.

= 1: any overflow terminates the repetition loop, and it is treated as usual; that is, the respective overflow indicator is set ON, and if the overflow mask is OFF, then also an overflow fault occurs on the even instruction, the odd instruction is not executed.

Bit 16 = 1: if Carry indicator is OFF, then terminate the repetition loop.

Bit 15 = 1: if Carry indicator is ON, then terminate the repetition loop.

Bit 14 = 1: if negative indicator is OFF, then terminate the repetition loop.

Bit 13 = 1: if negative indicator is ON, then terminate the repetition loop.

Bit 12 = 1: if zero indicator is OFF, then terminate the repetition loop.

Bit 11 = 1: if zero indicator is ON, then terminate the repetition loop.

10. At the time of termination:

$X0_{0-7}$ will contain the Tally Residue, that is, the number of repeats remaining until a Tally Runout would have occurred.

If the RPD instruction is interrupted before termination, the Tally Runout indicator will be OFF in any case.

The Xn specified by the designator of each two repeated instructions will contain the effective address of the next operand or indirect word that would have been secured.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RPL	Repeat Link	500

SUMMARY: Execute the next instruction a specified number of times, until a specified terminate condition is met or a link address = 0 is found.

MODIFICATIONS: None

INDICATORS:

Tally Runout	If termination because of Tally = 0 or link address = 0, then ON. If because terminate condition is met, then OFF.
All other indicators	The RPL instruction itself does not affect any of the other indicators. However, the execution of the repeated instruction may affect indicators.

NOTES: 1. This RPL instruction has the following format:

0 – 7	8	9	10	11 – 17	18 – 26	27	28	29	30 – 35
Tally			C	Term.Cond.	Op. Code	0	1	0	

2. If C = 1, then bits 0-17 of the RPL instruction => X0. If C = 0, the X0 contains whatever was left from the previous instruction.

3. The terminate condition and Tally X0 will control the repetition loop for the instruction following this RPL instruction; initial Tally = 0 will be interpreted as 256.

4. The repetition loop consists of the following steps:

   a. Execute the repeated instruction
   b. $C(X0)_{0-7}$ -1 => $C(X0)_{0-7}$
   c. If termination condition met then set Tally Runout indicator OFF and terminate
   d. If the Tally $C(Xn)_{0-17}$ = 0 or the link address $C(Y)_{0-17}$ = 0, then set Tally Runout indicator ON and terminate

      If first address is zero, termination will result upon completion of the first operation.

   e. Go to 4a.

5. All instructions can be used as repeated instructions except the

(Continued)

following:

a. Instructions that could alter the Link Address $C(Y)_{0-17}$
b. EAA, EAQ, EAX, NEG, NEGL
c. All special operations instructions
d. All shift instructions
e. All control instructions
f. All general base instructions--LBRn, EABn, ADBn, SBRn, LDBR, SDBR, CAM, EAPn, LDCF, RTCD, STCD, STPn, ZAM, TSBn, SCU, RCU, LDB, STB, SAM.

6. Address modification for the repeated instruction:

For the repeated instruction, only the modifier R is permitted, and only the designators specifying R = X1,...X7.

The effective address EA will be

a. For the first execution of the repeated instruction

$Y + C(R) = EA$; Y of word fetches => C(R)

b. For any successive execution of the repeated instruction

C(R) => EA; Y of word fetches => C(R)

The effective address EA is the address of the next list word.

The upper half of the list word contains the Link Address, that is, the address of the next successive list word, and thus the effective address for the next successive execution of the repeated instruction.

The lower half of this list word contains the operand to be used for this execution of the repeated instruction:

$C(Y)_{18-35}$

For double-precision instructions that are being repeated, the operand is

$C(Y)_{0-17} = \underbrace{00...0}_{18\text{ times}},\quad C(Y)_{18-35},\ C(Y+1)_{0-35}$

7. The terminate conditions:

The possible terminate conditions are the same for all three repeat instructions - RPT, RPD, RPL except that RPL can terminate on a Link Address = 0.

The bit configuration in bit positions 11-17 of the RPL instruction defines the terminate conditions for which the repetition loop will be terminated immediately. If more than one condition is specified, the repeat will terminate if any of the specified conditions are met.

Bit 17 = 0: any overflow is completely ignored; that is, neither the respective overflow indicator is set ON,

(Continued)

nor an overflow trap occurs;

Bit 17 = 1: any overflow terminates the repetition loop, and it is treated as usual; that is, the respective overflow indicator is set ON, and if the Overflow Mask indicator is OFF, then also an overflow fault trap occurs.

Bit 16 = 1: if Carry indicator is OFF, then terminate the repetition loop.

Bit 15 = 1: if Carry indicator is OFF, then terminate the repetition loop.

Bit 14 = 1: if negative indicator is OFF, then terminate the repetition loop.

Bit 13 = 1: if negative indicator is ON, then terminate the repetition loop.

Bit 12 = 1: if zero indicator is OFF, then terminate the repetition loop.

Bit 11 = 1: if zero indicator is ON, then terminate the repetition loop.

8. At the time of termination:

$X0_{0...7}$ will contain the Tally residue, that is, the number of repeats remaining until a Tally runout would have occurred. If the RPL instruction is interrupted before termination, the Tally Runout indicator will be OFF in any case.

The Xn specified by the designator of this repeated instruction will contain the address of the list word that contains

In its upper half:	the address of the next list word
In its lower half:	the operand used in the last execution of the repeated instruction, for single-precision instructions. For double-precision instructions, this half word and the next full word are the operand last used.

(This is because there is no overlap between an execution of the repeated instruction and the address modification for the next execution of the repeated instruction.)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RSW	Read Switches	231

SUMMARY: C(Data Switches on maintenance panel) => $C(A)_{0-35}$

MODIFICATIONS: All types except DU, DL, CI, SC, SCR are allowed but none affect the operation of RSW.

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

Mnemonic:	Name of the Instruction:	Op Code (Octal)
NOP	No Operation	011

SUMMARY: No operation takes place

MODIFICATIONS: Generally the modification N, DU or DL should be used.*

INDICATORS: None affected

NOTE: No operation takes place but address modification may take place. If modification other than DU or DL is used, the effective address will be used in a memory access request which could lead to memory faults. The use of IT modification categories ID, DI, IDC, DIC causes the respective changes in the address and tally.

* This reminder that address formation will be performed should also serve as a warning.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LACL	Load Alarm Clock	453

SUMMARY: $C(AQ)_{20-65} \Rightarrow C(\text{Alarm Clock})_{51-6}$

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

INDICATORS: None affected

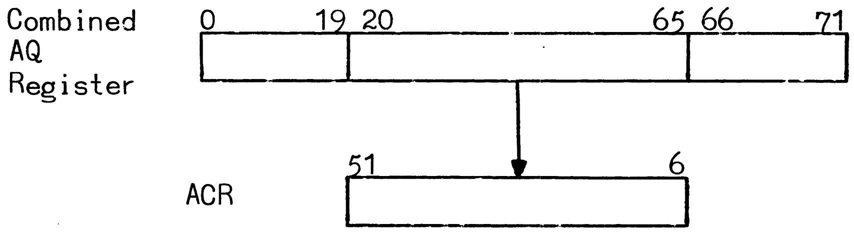
NOTES: 1. The absolute address generated by this instruction is used in selecting a processor port as with a normal memory access request but execution requires only an address for port selection since this instruction involves a non-core memory command like the RCCL, RMCM, and SMIC instructions.

2. When selecting a memory module; memory size, instruction word address field, PBR, TBR, internal and external ABR's, and memory interlace (if the final address is not 0 modulo 8) must be considered.

3. This instruction may be executed in Master mode only or an illegal procedure fault is generated.

4. The command actually sent out the processor port is SAC.

5. Refer to figure.



6. Note that, in contrast to normal system terminology (in which the most significant bit is bit 0), in the terminology used with the alarm clock bit 51 is the most significant bit.

(Revised March 19, 1971)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RCCL	Read Calendar Clock	633

SUMMARY: 0...0 => $C(AQ)_{0-19}$; $C(\text{Calendar Clock})_{51-0}$ => $C(AQ)_{20-71}$

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

INDICATORS: None affected

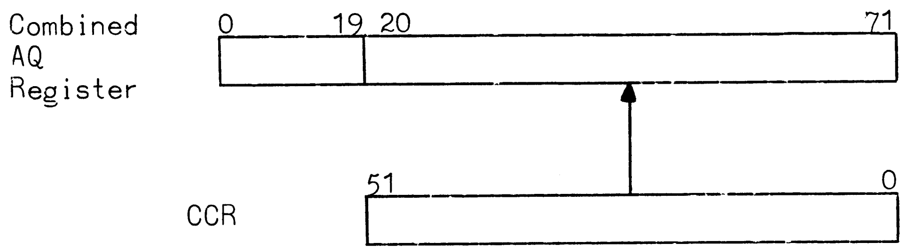
NOTES:

1. The absolute address generated for this instruction is used in selecting a processor port as with a normal memory access request but execution requires only an address for port selection, since this instruction involves a non-core memory command like the LACL, RMCM, SMCM and SMIC instructions.

2. When selecting a memory module; memory size, instruction word address field, PBR, TBR, internal and external ABR's, and memory interlace (if the final address is not 0 modulo 8) must be considered.

3. The command actually sent out the processor port is RCC.

4. Refer to figure.



5. Note that, in contrast to normal system terminology (in which the most significant bit is bit 0), in the terminology used with the calendar clock bit 51 is the most significant bit.

(Revised March 19, 1971)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LAM	Load Associative Memory	257

SUMMARY: C(Y,...Y+31) => C(AM)

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

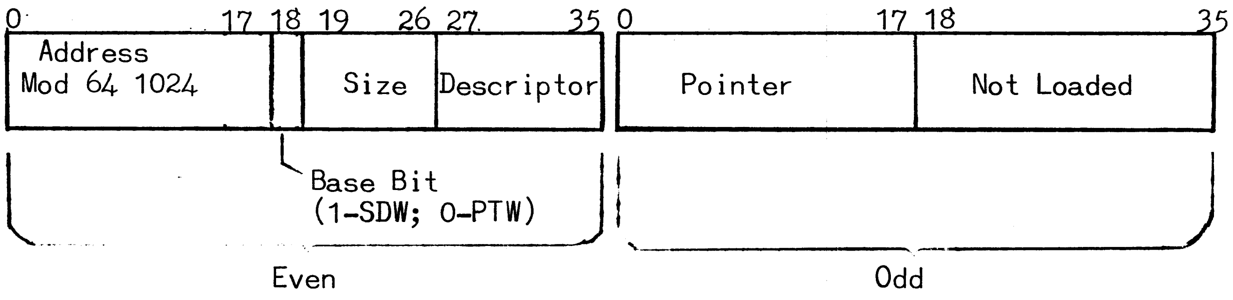
INDICATORS: None affected

NOTES: 1. The contents of even-odd word pairs are loaded into the associative memory as follows:

$(Y, Y+1)_{0-53}$ => AR (Usage = 0)

$(Y+2, Y+3)_{0-53}$ => AR (Usage = 1)

$(Y+30, Y+31)_{0-53}$ => AR (Usage = 15)



2. Unless the associative memory is first cleared with a CAM, the registers will be loaded in the order of the usage counts. If it is first cleared with a CAM instructions, the registers will be loaded "A" first on through "S".

3. Bit 54 through 59, while not loaded, will be set to a specific state as follows:

   Bit 54 set to 1
   Bit 55 set to 0
   Bit 56-59 remain the same as they were before the LAM instruction was executed.

4. This instruction may be executed in Master mode only or an illegal procedure fault will occur.

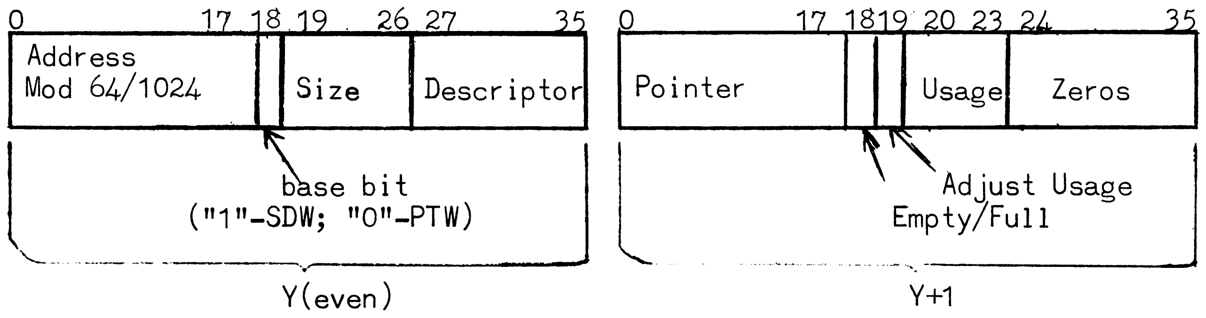
Mnemonic:	Name of the Instruction:	Op Code (Octal)
SAM	Store Associative Memory	557

SUMMARY: $C(AR)_{0-15}$ => C(Y,...,Y+31)
where Y must be 0 modulo 32.

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

INDICATORS: None affected

NOTE: The contents of the associative memory (16 words) are stored in sequence as even-odd word pairs in double-word stores. The initial value is assigned by the hardware (that is, associative memory register 0 stored in first word pair...associative memory register 15 stored in 16th word pair). If this instruction is attempted in Slave mode an illegal procedure fault will be generated. The format of the word pair stored follows:



Mnemonic:	Name of the Instruction:	Op Code (Octal)
ZAM	Store Associative Memory Zero	157

SUMMARY: C(AR) => C(Y,Y+1)
where $C(AR)_{56-59}$ = 0 and Y is an even location.

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

INDICATORS: None affected

NOTE: This instruction stores the C(AR) that has a current zero usage value into an even-odd word pair. The format is the same as that of the SAM instruction. If this instruction is attempted in Slave mode an illegal procedure fault is generated.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SCU	Store Control Unit	657

SUMMARY: C(TBR) => $C(Y)_{0-17}$, Appending Unit Status => $C(Y)_{18-35}$
Computed Address => $C(Y+1)_{0-17}$, Control Unit Status => $C(Y+1)_{18-35}$,
C(PBR) => $C(Y+2)_{0-17}$, Fault Data => $C(Y+2)_{18-35}$
C(ICTC) => $C(Y+3)_{0-17}$, C(IR) => $C(Y+3)_{18-28}$
Control Unit Status => $C(Y+3)_{30-35}$;
Even Instruction => $C(Y+4)_{0-35}$
Odd Instruction => $C(Y+5)_{0-35}$

MODIFICATIONS: All except DU, DL, CI, SC, SCR, F1, F2, F3

INDICATORS: None Affected

NOTES:
1. Detection of a fault or external interrupt condition will cause the taking of a snapshot of the CU status, storing it in temporary buffer registers, aborting the current sequence, and forcing an XED instruction pointing to an entry in a vector table corresponding to the particular fault or interrupt.

2. The SCU instruction is used specifically to store the buffered snapshot of the Control Unit immediately following interrupts or faults where eventual return to an exact point in the sequence is expected. Therefore, SCU must be used as the even instruction of an Execute Interrupt- or Fault Vector-pair.

3. If an attempt is made to issue SCU as a part of an instruction sequence, rather than in a vector following a "snapshot", the results described will not be obtained.

4. The first double word is stored directly from the data out register and contains whatever the last STORE+READ-ALTER-REWRITE cycle left there. The second double word stores the PBR, ICTC and IR. The third double word stores the instruction buffer registers and it may contain the next pair of instructions in the sequence, or any pair of instructions.

5. If external interrupts are honored during address preparation cycles for SCU, the current "snapshot" will be destroyed. Therefore, it is required that bit 28 of SCU be set to 1.

6. SCU is a privileged instruction to be executed in Master mode only or an illegal procedure fault will occur.

7. The effective address of an SCU instruction must be 0 modulo 8.

(Continued)

8. The execution of the SCU instruction involves the following actions:

   a. $C(TBR) => C(Y)_{0-17}$
   The contents of the temporary base register, whose snapshot now resides in the data out register (bits 0-17), stored in the upper half of Y.

   b. Appending Unit Status $=> C(Y)_{18-35}$
   The appending unit status, whose snapshot now resides in the data out register (bits 18-35), is stored in the lower half of Y as follows:

Bit Position	Name	Definition
18-21	$OSTR_{0-3}$	Odd Segment Tag Register and Use Flag
22-25	$ESTR_{0-3}$	Even Segment Tag Register and Use Flag
26	ITS	ITS Tag
27	ITB	ITB Tag
28		Zero (Not Used)
29	PEO	Parity Error, Operand
30	ITR	Indirect Tally Not Equal to Tally Runout Indicator
31		Zero (Not Used)
32		Zero (Not Used)
33	DS PTW	Descriptor Segment PTW Fetch
34	SDW	Segment Descriptor Word Fetch
35	PTW	Page Table Word Fetch

   c. Computed Address $=> C(Y+1)_{0-17}$
   The address, generated during the address preparation cycle, and whose snapshot is in the data out register (bits 36-53), is stored in the upper half of Y+1.

   This may be the address of an operand, indirect word, or an instruction. Applicable registers (index, external base) are included.

(Continued)

(SCU Instruction, Continued)

d. Control unit status => $C(Y+1)_{18-35}$
The control unit status whose snapshot now resides in the data out register (bits 54-71), is stored in the lower half of Y+1 as follows:

Bit Position	Name	Definition
18	PI	Instruction Fetch (1) Address Modifications (0)
19	PN	Indirect Address-Forced (no address modification)
20		Not Used
21	XDE	Execute Double Even
22	XDO	Execute Double Odd
23	IC	Even (0) or Odd (1) Instruction
24	MASF	Temporary Absolute Mode
25	EA	Operand (1) Indirect Fetch (0)
26	M/S	Master (1) Slave (0)
27	PA	Initial Address Preparation
28	PZ	Indirect Address Preparation (IR+RI)
29	PT	Indirect Address Preparation (IT)
30-35	CT_{0-5}	Control Tag

e. C(PBR) => $C(Y+2)_{0-17}$
The procedure base register is stored in the upper half of Y+2.

f. Fault Data => $C(Y+2)_{18-35}$

Information on the fault is stored in the lower half of Y+2 as follows:

Bit Position	Name	
18-20	Processor Number (000-111)	
21	Illegal Procedure type 1	
22	Illegal Procedure type 2	
23	Illegal Procedure type 3	[See Chapter 7
24	Illegal Procedure type 4	on Faults]
25	Illegal Procedure type 5	
26-30	Fault Code (00000-11111)	
31-35	Zero (Not Used)	

(Continued)

g. $C(ICTC) \Rightarrow C(Y+3)_{0-17}$
The instruction counter is stored in the upper half or Y+3.

In the case of an interrupt during an operand, or an indirect word address preparation, the ICTC specifies the location of the active instruction. In the case of an interrupt prior to an instruction fetch, the ICTC specifies the address of the last successfully executed instruction.

It should be noted that in the case of a fault on an XEC'ed or XED'ed instruction, the ICTC points to the location of the original XEC or XED instruction, rather than the location of the faulting instruction.

h. $C(IR) \Rightarrow C(Y+3)_{18-28}$, $0 \Rightarrow C(Y+3)_{29}$
The contents of the indicator register will be stored in Y+3 as follows:

Bit Position	Indicators
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Absolute Mode

The $C(Y+3)_{25}$ will contain the state of the Tally Runout indicator <u>prior</u> to address modification of the instruction (for tally operations).

i. Control unit status $\Rightarrow C(Y+3)_{30-33}$, $00 \Rightarrow C(Y+3)_{34,35}$
Control unit status will be stored in Y+3 as follows:

Bit Position	Definition
30	Initial Repeated Instruction
31	Repeat
32	Repeat Link
33	Repeat Double

j. EVEN Instruction $\Rightarrow C(Y+4)_{0-35}$

k. ODD Instruction $\Rightarrow C(Y+5)_{0-35}$

The active pair of instructions is stored in Y+4 and Y+5.

(Continued)

9. If the interrupt occurred prior to an instruction fetch (PI cycle), then these instructions have already been executed. If the interrupt occurred during address preparation for an indirect word or an operand, then if IC = 0 the faulting instruction is the even one, or if IC = 1 the faulting instruction is the odd.

10. The address field of the faulting instruction, $C(Y+4)_{0-17}$, contains the address field of the instructions, or the last indirect word, or the last indirect word minus one or delta. The tag field of the faulting instruction, $C(Y+4 \text{ or } 5)_{0-35}$, contains the tag of the original instruction or the last indirect word.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RCU	Restore Control Unit	613

SUMMARY: $C(Y)_{0-17}$ => C(TBR), $C(Y)_{18-35}$ => Appending Unit Status

$C(Y+1)_{18-35}$ => Control Unit Status

$C(Y+2)_{0-17}$ => C(PBR)

$C(Y+3)_{0-17}$ => C(ICTC), $C(Y+3)_{18-28}$ => C(IR), $C(Y+3)_{30-35}$ => Control Unit Status

C(Y+4) => Even Instruction

C(Y+5) => Odd Instruction

MODIFICATIONS: All types except DU, DL, CI, SC, SCR, F1, F2, and F3, are recognized by the hardware but defeat the purpose of the RCU.

INDICATORS: The RCU instruction of itself does not affect the indicators; however, the contents of Y+3 bits 0-28 will be placed in the indicator register.

NOTES:

1. This instruction can only be used in Master mode. If attempted in Slave mode an illegal procedure fault will occur.

2. The execution of the RCU instruction involves the following actions:

   a. $C(Y)_{0-17}$ => C(TBR)
   The contents of the upper half of Y replaces the contents of the temporary base register.

   b. $C(Y)_{18-35}$ => Appending Unit Status
   The contents of the lower half of Y will affect the appending unit registers and flags as follows:

Bit Position	Action
18-21	Replaces the contents of OSTR
22-25	Replaces the contents of ESTR
26	If "1" set ITS Flag, if "0", reset ITS Flag
27	If "1" set ITB Flag, if "0", reset ITB Flag
28-29	No effect
30	If "1" set ITR, if "0", reset Tally Runout.
31-35	No effect

(Continued)

c. $C(Y+1)_{18-35}$ => Control Unit Status
The contents of the lower half of Y+1 will affect the control unit registers and flags as follows:

Bit Position	Action
18	If "1" set PI, if "0" reset PI
19	If "1" set PN, if "0" reset PN
20	No effect
21	If "1" set XDE, if "0" reset XDE
22	If "1" set XDO, if "0" reset XDO
23	If "1" set IC, if "0" reset IC
24	If "1" set MASF, if "0" reset MASF
25	If "1" set EA, if "0" reset EA
26	If "1" set MS, if "0" reset MS
27	If "1" set PA, if "0" reset PA
28	If "1" set PZ, if "0" reset PZ
29	If "1" set PT, if "0" reset PT
30	Replaces the contents of CT_{0-5}

d. $C(Y+2)_{0-17}$ => C(PBR)
The contents of the upper half of Y+2 replaces the contents of the procedure base register.

e. $C(Y+3)_{0-17}$ => C(ICTC)
The contents of the upper half of Y+3 replaces the contents of the instruction counter.

f. $C(Y+3)_{18-28}$ => C(IR)
The contents of bits 18-28 of Y+3 affect the indicator register as follows:

Bit Position	Action
18	If "1" set Zero, if "0" reset Zero
19	If "1" set Negative, If "0" reset Negative
20	If "1" set Carry, if "0" reset Carry
21	If "1" set Overflow, if "0" reset Overflow
22	If "1" set Exponent Overflow, if "0" reset Exponent Overflow
23	If "1" set Exponent Underflow, if "0" reset Exponent Underflow
24	If "1" set Overflow Mask, if "0" reset Overflow Mask
25	If "1" set Tally Runout, if "0" reset Tally Runout
26	If "1" set Parity Error, if "0" reset Parity Error
27	If "1" set Parity Mask, If "0" reset Parity Mask
28	If "1" set Absolute Mode, if "0" reset Absolute Mode

(Continued)

g. $C(Y+3)_{30-35}$ => Control Unit Status
The contents of Y+3, bits 30-35 affect the Control Unit Status as follows:

Bit Position	Action
30	If "1" set RF, if "0" reset RF
31	If "1" set FT, if "0" reset FT
32	If "1" set FL, if "0" reset FL
33	If "1" set FD, if "0" reset FD
34	No effect
35	No effect

h. $C(Y+4)_{0-35}$ => Even Instruction
The contents of Y+4 replaces the contents of the Even Instruction.

i. $C(Y+5)_{0-35}$ => Odd Instruction
The contents of Y+5 replaces the contents of the Odd Instruction.

CHAPTER 3 DATA REPRESENTATION

INFORMATION REPRESENTATION

The 645 processor is organized to deal with 36-bit groupings of information. In addition, 6-bit, 9-bit, and 18-bit groups plus 72-bit double precision groups can be manipulated via the instruction set. These bit groupings are used by the hardware and software to represent a variety of forms of information. All notation used throughout the processor is in binary. The way information is represented for instruction, indirect, appending and associative memory words is included in these paragraphs.

POSITION NUMBERING

The numbering of bit positions, character positions, words, etc., increases in the direction of conventional reading and writing: from the most- to the least-significant digit of a number, and from left to right in conventional alphanumeric text.

Graphical presentations in this manual show registers and data with position numbers increasing from left to right.

NUMBER SYSTEM

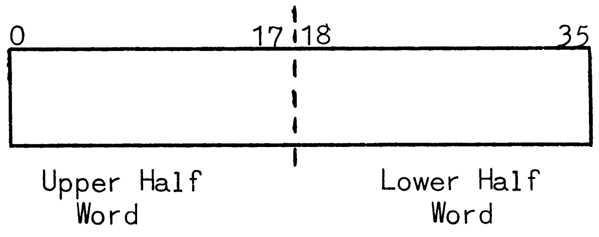
With the binary system of notation used throughout the processor, many of the instructions (mainly additions, subtractions, and comparisons) can be used in two ways: either operands and results are regarded as signed binary numbers in the two's complement form (the "arithmetic" case), or they are regarded as unsigned positive binary numbers (the "logic" case). Hardware actions within the processor are the same in either case; interpretation of the data by the programmer is the only difference. The zero and negative indicators facilitate the general interpretation of the results in the arithmetic case; the zero and carry indicators in the logic case. The overflow indicator reflects the occurrence of overflow for instructions involving the "arithmetic" and "logic" cases, i.e., logical add, logical subtract, arithmetic add and arithmetic subtract. The instruction set contains "add logic" instructions which particularly facilitate arithmetic of the logic type with half-word, single-word, and double-word precision.

Subtractions are carried out internally by adding the two's complement of the subtrahend. (Note that when the subtrahend is zero the algorithm for forming the two's complement is still carried out. Thus, each bit of the subtrahend is complemented and a "1" is added into the least-significant bit position of the parallel adder, yielding zeros). It is a characteristic feature of the two's complement representation that a "no borrow" condition in the case of true subtraction corresponds to a "carry" condition in the case of the two's complement and vice versa.

A statement on the assumed location of the binary point has significance only for multiplications and divisions. These two operations are implemented for integer arithmetic as well as for fractional arithmetic with numbers in two's complement form. "Integer" means that the position of the binary point may be assumed to the right of the least-significant bit position (that is, to the right of bit position 35 or 71, depending on the precision of the respective number) and "fractional" means that the position of the binary point may be assumed to the left of the most-significant bit position (that is, between the bit positions 0 and 1).

PROCESSOR MACHINE WORD

The machine word consists of 36 bits arranged as shown in the following figure.

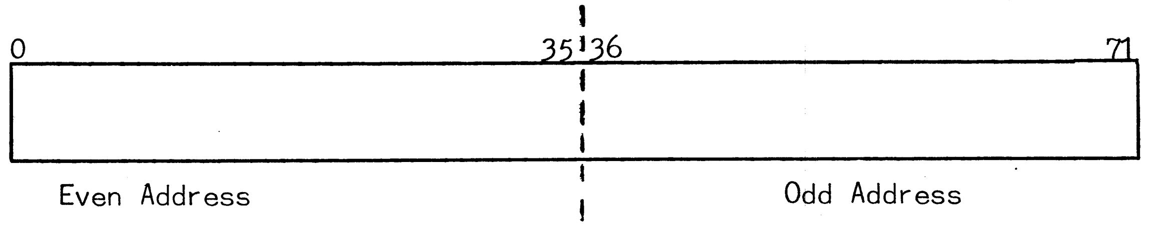


One Machine Word

Numbering of bit positions and character positions within the machine word increases from left to right. In a single word the bit positions are numbered from 0 to 35; if two machine words make a double word, positions are numbered from 0 to 71, where 0 is the leftmost bit and 71 the rightmost bit.

Data transfers between processor and memory are word oriented; 36 bits are transferred at a time for single-precision data and two successive 36-bit words are transferred for double-precision data. Parity on words transferred to, or read from the memory module are handled completely within that module. The processor is notified only if a parity error exists. The memory module adds a parity bit to each 36-bit word before storing it. When words are requested from memory, the memory verifies the parity bit read from the memory and removes it from the word transferred prior to sending each word to the processor.

The processor has features for transferring and processing pairs of words. In transferring a pair of words to or from memory, a pair of memory locations is accessed; these addresses are an even and the next-higher odd number as shown in the following figure:



A Pair of Machine Words

In addressing such a pair of memory locations in an instruction that is intended for handling pairs of machine words, either of the two addresses may be used as the absolute address (Y) except as noted under the instruction descriptions. Thus, if Y is even, the pair of locations (Y, Y+1) is accessed. If Y is odd, the pair of locations (Y-1, Y) is accessed. The term "Y-pair" is used for each such pair of addresses.

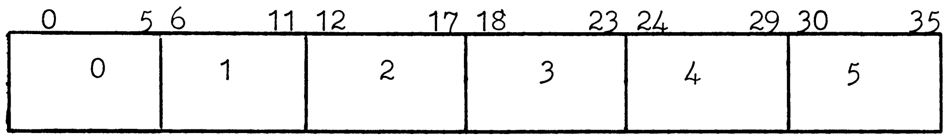
REPRESENTATION OF DATA

Data is represented in two forms: alphanumeric or numeric. Both forms may be handled by the hardware as indicated by the instruction repertoire.

Alphanumeric Data

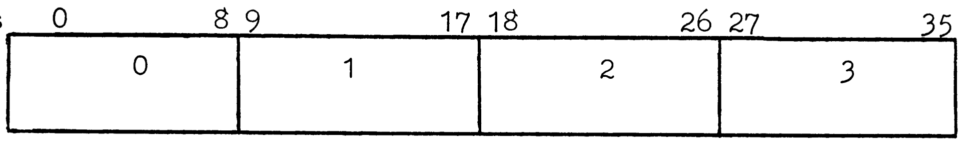
Alphanumeric data are represented by 6-bit or 9-bit characters. A machine word contains either six of four characters as shown on the next page:

Character Positions within a word



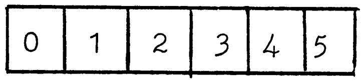
6-bit

Character Positions within a word

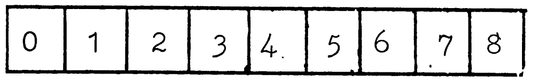


9-bit

Bit positions within a character



6-bit



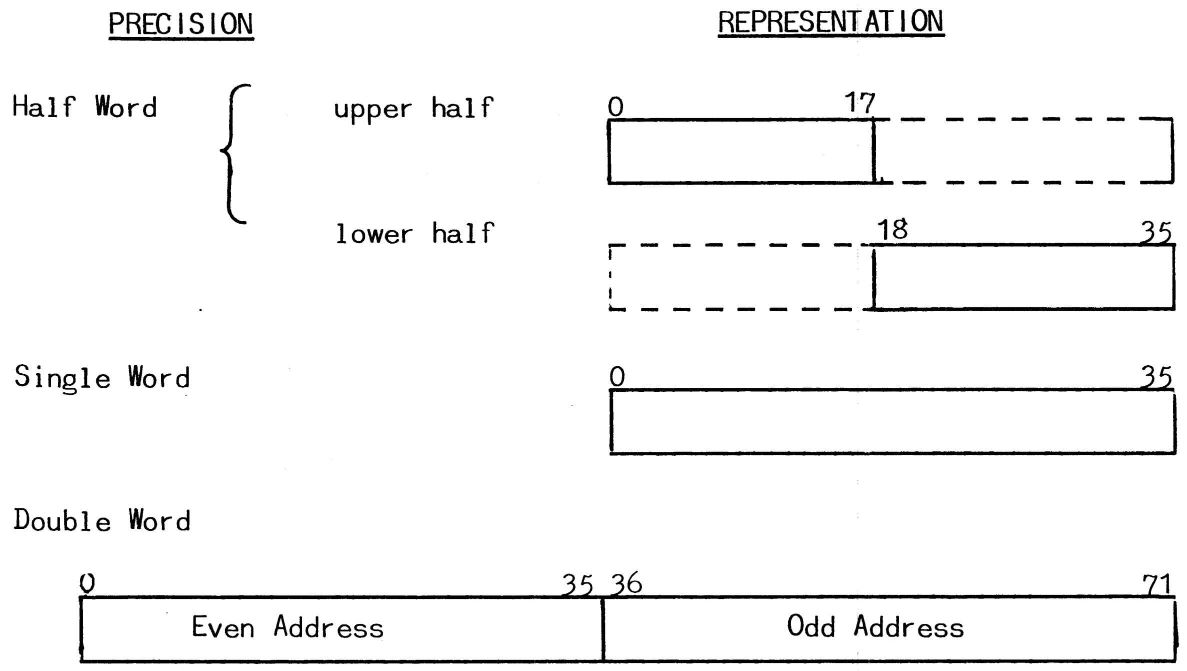
9-bit

Numeric Data

Numeric data is represented in two forms; binary fixed-point, or binary floating-point. Decimal data is handled by software.

Binary Fixed-point Numbers

The instruction set comprises instructions for binary fixed-point arithmetic with half-word, single-word, and double-word precision as shown in the following figure.



Instructions can be divided into two groups according to the way in which the operand is interpreted: the "logic" group and the "algebraic" group.

For the "logic" group, operands and results are regarded as unsigned, positive binary numbers. In the case of addition and subtraction, the occurrence of any overflow is reflected by the carry out of the most-significant (leftmost) bit position.

Addition - If the carry out of the leftmost bit position equals 0, then the result is below the range.

Subtraction - If the carry out of the leftmost bit position equals 0, then the result is below the range.

In the case of comparisons, the Zero and Carry indicators show the relation.

For the "algebraic" group, operands and results are regarded as signed, binary numbers, the leftmost bit being used as a sign bit, (a 0 being plus and 1 minus). When the sign is positive all the bits represent the absolute value of the number; and when the sign is negative, the bits represent the two's complement of the absolute value of the number.

In the case of addition and subtraction the occurrence of an overflow is reflected by the carries into and out of the leftmost bit position (the sign position). If the carry into the leftmost bit position does not equal the carry out of that position, then overflow has occurred. If the overflow has been detected and if the sign bit equals 0, the resultant is below range; if with overflow, the sign bit equals 1, the resultant is above range.

An explicit statement about the assumed location of the binary point is necessary only for multiplication and division. When performing addition, subtraction, and comparison it is sufficient to assume that the binary points are "lined up". In the 645 processor, multiplication and division are implemented in two forms for two's complement numbers: integer and fractional.

1. In integer arithmetic, the location of the binary point is assumed to the right of the least-significant bit position, that is, depending on the precision, to the right of bit position 35 or 71. The general representation of a fixed-point integer is then:

   $$- a_n 2^n + a_{n-1} 2^{n-1} + a_{n-2} 2^{n-2} + \dots + a_1 2^1 + a_0 2^0 \underset{\text{binary point}}{\wedge}$$

   where a_n is the sign bit.

2. In fractional arithmetic, the location of the binary point is assumed to the left of the most-significant bit position, that is, to the left of bit position 1. The general representation of a fixed-point fraction is then:

   $$- a_0 2^0 \underset{\text{binary point}}{\wedge} + a_1 2^{-1} + a_2 2^{-2} + \dots + a_{n-1} 2^{-(n-1)} + a_n 2^{-n}$$

The number ranges for the various cases of interpretation, precision, and arithmetic are listed in the table below.

Interpretation	Arithmetic	Precision		
		Half-Word (X_n, Y_{0-17})	Single-Word (A,Q,Y)	Double-Word (AQ,Y-pair)
Algebraic	Integral	$-2^{17}\leq N\leq(2^{17}-1)$	$-2^{35}\leq N\leq(2^{35}-1)$	$-2^{71}\leq N\leq(2^{71}-1)$
	Fractional	$-1\leq N\leq(1-2^{-17})$	$-1\leq N\leq(1-2^{-35})$	$-1\leq N\leq(1-2^{-71})$
Logic	Integral	$0\leq N\leq(2^{18}-1)$	$0\leq N\leq(2^{36}-1)$	$0\leq N\leq(2^{72}-1)$
	Fractional	$0\leq N\leq(1-2^{-18})$	$0\leq N\leq(1-2^{-36})$	$0\leq N\leq(1-2^{-72})$

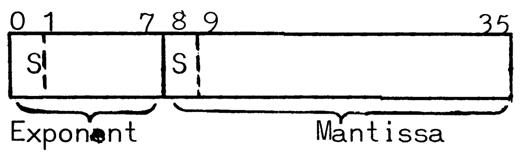
Ranges of Fixed-Point Numbers

Binary Floating-point Numbers

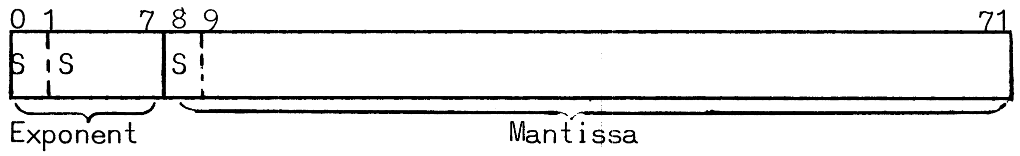
The instruction set contains instructions for binary floating-point arithmetic with numbers of single-word and double-word precision. The upper 8 bits represent the integral exponent E in two's complement form, and the lower 28 or 64 bits represent the fractional mantissa M in two's complement form. The notation for a floating-point number Z is:

$$Z = M \cdot 2^E$$

Single-Word Precision



Double-Word Precision



COMPARISON RELATIONS

In the case of comparisons, the Zero and Negative and Carry indicators show the relative results of the comparison.

ALGEBRAIC (SIGNED FIXED-POINT COMPARISON)				
INDICATORS			RELATION	SIGNS
ZERO	NEGATIVE	CARRY		C(R), C(Y)
0	0	0	C(R) > C(Y)	+ -
0	0	1	C(R) > C(Y)	+ -
1	0	1	C(R) = C(Y)	+ +
0	1	0	C(R) < C(Y)	+ +
0	1	1	C(R) < C(Y)	- +

LOGIC (UNSIGNED) FIXED POINT		
INDICATORS		RELATION
ZERO	CARRY	
0	0	C(R) < C(Y)
1	1	C(R) = C(Y)
0	1	C(R) > C(Y)

FLOATING POINT COMPARE		
INDICATORS		RELATION
ZERO	NEGATIVE	
0	0	C(R) > C(Y)
1	0	C(R) = C(Y)
0	1	C(R) < C(Y)

Alignment and Representation

Before doing floating-point additions or subtractions, the processor aligns the number which has the smaller positive exponent. To maintain accuracy, the lowest permissible exponent of -128 together with the mantissa equal to 0000 ...0 shall be defined as the machine representation of the number zero (which has no unique floating-point representation). Whenever a floating-point operation yields a result whose untruncated mantissa is equal to zero (71 bits plus sign because of extended precision), the exponent is automatically set to -128. The general representation of the exponent for single and double precision is:

$$-e_7 2^7 + e_6 2^6 + \ldots + e_1 2^1 + e_0 2^0$$

where e_7 is the sign.

The general representations of single- and double-precision mantissas are:

Single Precision: $-m_0 2^0 + m_1 2^{-1} + m_2 2^{-2} + \ldots + m_{26} 2^{-26} + m_{27} 2^{-27}$

Double Precision: $-m_0 2^0 + m_1 2^{-1} + m_2 2^{-2} + \ldots + m_{62} 2^{-62} + m_{63} 2^{-63}$

where m_0 is the sign in both cases.

Normalized Floating-point Numbers

For normalized floating-point numbers, the binary point is placed at the most-significant bit of the mantissa (to the right of the sign bit). Numbers are normalized by shifting the mantissa (and correspondingly adjusting the exponent) until no leading zeros are present in the mantissa for positive numbers, or until no leading ones are present for the negative numbers. Zeros fill in the vacated bit positions. With the exception of the number zero (represented as 0×2^{-128}), all normalized floating-point numbers will contain a binary 1 in the most-significant bit position for positive numbers and a binary 0 in the most-significant bit position for negative numbers. Some examples are:

Unnormalized positive number	$(\underset{S}{0}\ 0001101) \times 2^7$
Same number normalized	$(\underset{S}{0}\ 1101000) \times 2^4$
Unnormalized negative number	$(\underset{S}{1}\ 11010111) \times 2^{-4}$
Same number normalized	$(\underset{S}{1}\ 01011100) \times 2^{-6}$

Number Ranges

The number ranges resulting from the various cases of precision normalization, and sign are listed in the table below:

	Sign	Single Precision	Double Precision
Normalized	Positive	$2^{-129} \leq N \leq (1-2^{-27})2^{-127}$	$2^{-129} \leq N \leq (1-2^{-63})2^{127}$
	Negative	$-(1+2^{-26})2^{-129} \geq N \geq -2^{127}$	$-(1+2^{-62})2^{-129} \geq N \geq -2^{127}$
Unnormalized	Positive	$2^{-155} \leq N \leq (1-2^{-27})2^{127}$	$2^{-191} \leq N \leq (1-2^{-63})2^{127}$
	Negative	$-2^{-155} \geq N \geq -2^{127}$	$-2^{-191} \geq N \geq -2^{127}$

Number Ranges in the 645

Note: The floating-point number zero is not included in the table.

CHAPTER 4 PROGRAM ACCESSIBLE REGISTERS

There are a number of processor registers. Some are explicitly referenced by particular instructions.* Others are implicitly referenced during the course of execution of instructions. Still others are used in both ways. These registers are listed in the table below. See Chapter 2 for a discussion of each instruction to determine the way in which registers are used.

Name	Mnemonic	Bit Length	Quantity
Accumulator Register	A	36	1
Quotient Register	Q	36	1
Exponent Register	E	8	1
Accumulator Quotient Register**	AQ	72	1
Exponent Accumulator Quotient Register**	EAQ	72***	1
Address Base Registers	ABRn	24	8
Descriptor Base Register	DBR	29	1
Instruction Counter	IC or ICTC	18	1
Index Register	Xn	18	8
Procedure Base Register	PBR	18	1
Timer Register	TR	24	1
Indicator Register	IR	18	1
Associative Memory Register	AR7	60	16

Processor Registers

* There are two external registers which are also explicitly referenced by particular instructions. These two registers are part of the Calendar Clock and are called the Calendar Clock Register and the Alarm Clock register. The Calendar Clock register provides the calendar time upon request and the Alarm Clock register provides program interrupts at predetermined times. The Calendar Clock is a 52-bit register which increments each 1.0 microsecond. The Alarm Clock is a 0 mod 128 register which initiates a program interrupt when the calendar time becomes equal to or greater than the time stored in the Alarm Clock register.

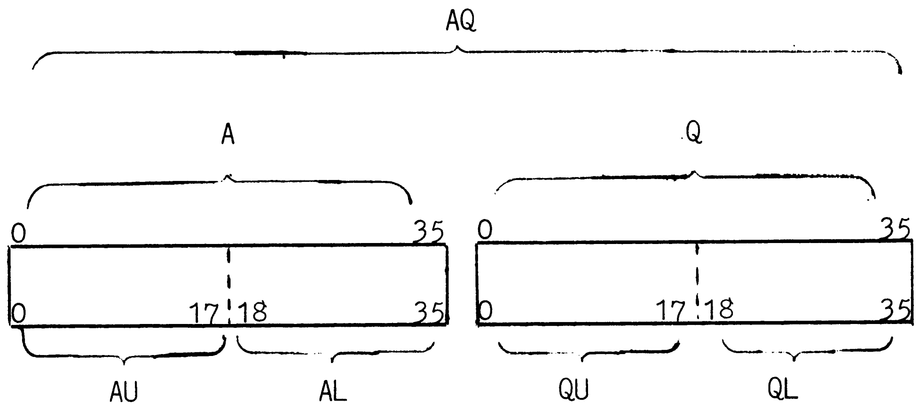
** The AQ and EAQ registers are not separate registers. They are a logical combination of the A and Q registers; and the E, A, and Q registers.

*** The exponent register is used instead of the least significant 8 bits of the AQ register.

(Revised March 19, 1971)

ACCUMULATOR (A), QUOTIENT (Q), AND ACCUMULATOR-QUOTIENT (AQ) REGISTERS

Formats:



Function:

In floating-point operations, the AQ register serves as a mantissa register for single- and double-precision.

In fixed-point double-precision operations, the AQ serves as an operand register.

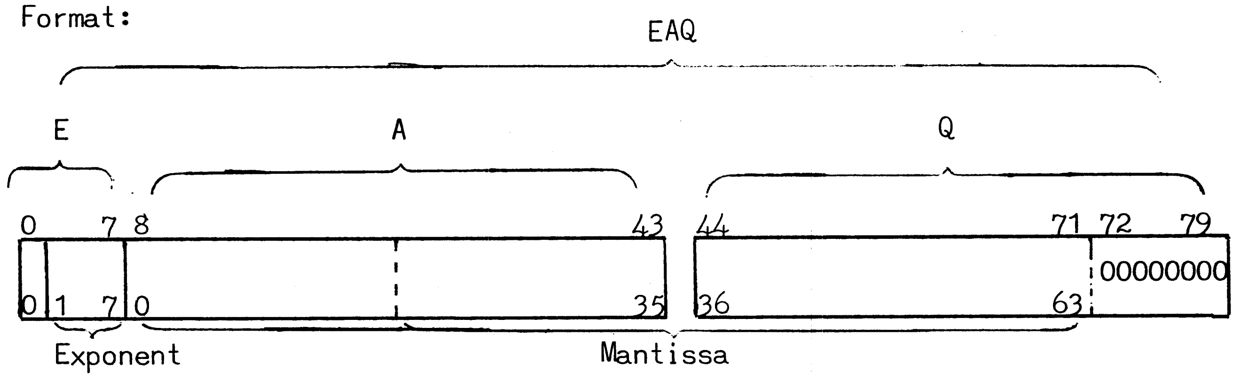
In fixed-point single-precision operations, the A register and Q register are independent of each other and serve as accumulator and quotient registers. Each register holds an operand. The description of each instruction explains how these registers are used.

In address modification, each half of the A register and each half of the Q register holds an index value. The halves of the registers are designated as follows:

A_0 - 17	A upper	(AU)
A_{18} - 35	A lower	(AL)
Q_0 - 17	Q upper	(QU)
Q_{18} - 35	Q lower	(QL)

EXPONENT (E) AND EXPONENT ACCUMULATOR QUOTIENT (EAQ) REGISTERS

Format:



Function:

The E register supplements the AQ register in floating-point operations, serving as exponent register for the floating-point number.

Bit position 0 indicates whether the sign of the exponent is plus (0) or minus (1).

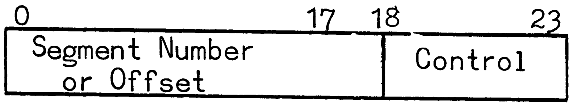
Bit positions 1-7 contain a number (from 0 to 127) representing the exponent of a floating-point number residing in the AQ register.

Bit positions 8-71 of the EAQ (0 to 63 of the AQ) hold the mantissa. When the floating-point number is loaded from Y-pair locations, the last eight bit positions (72-79) of the EAQ register are reset to zero.

When the floating-point number is placed in store, only bit positions 0-71 of the EAQ register are transferred. However, 72 bits of mantissa precision are maintained internal to the processor before storing.

ADDRESS BASE REGISTERS (ABR_0 to ABR_7)

Format:



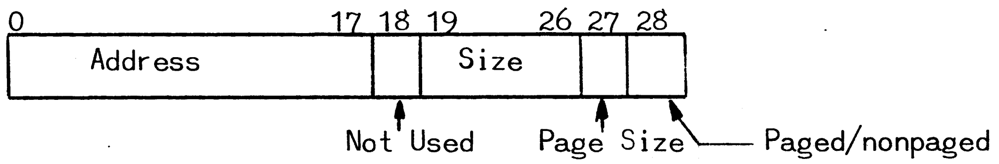
Function:

An address base register contains the segment number or offset of a particular segment and the control information required during the address appending process. The registers are used for operand fetches, indirect word fetches, and for transfers. ABR's can be used either singly or in pairs. An internal ABR contains an offset relative to the base of a segment and is "linked" by its control field to another ABR containing a segment number, whose control field indicates that it is an external ABR. When the internal ABR is addressed by an instruction the CPU directs the operation called for to the segment defined by the "linked" external register at the location specified by the sum of the internal register's offset field and the instrution address field. If the external ABR is addressed directly, the operation is directed to the segment defined in the register at the location specified by the instruction address field. Use of the ABR's is described in greater detail in Chapter 6.

Segment Number	-A segment number is an 18-bit number used to specify a segment.
Offset	-An offset is an 18-bit number used to specify a location relative to the beginning of a segment.
Control	-Bits 18-20 designate the second ABR of a pair when the ABR is internal. The bits are not used when the ABR is external. Bit 21 designates internal and external ABR's: 0 = an internal ABR 1 = an external ABR Bit 22 designates whether the ABR can be changed in Slave mode: 0 = can be changed 1 = cannot be changed Bit 23 is unassigned

DESCRIPTOR BASE REGISTER (DBR)

Format:



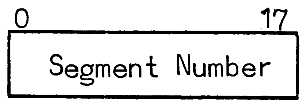
Function:

The descriptor base register contains the absolute address of the descriptor segment and the control information required during the address appending process.

Address	- bits 0-17 hold the high-order 18 bits of the 24-bit absolute address of the descriptor segment (or the descriptor segment page table if the descriptor segment is paged). The address is 0 modulo(64).
Size	- bits 19 to 26 contain a value that indicates the number of 64- or 1024-word pages or blocks in the descriptor segment.
Page Size	- bit 27 indicates whether the descriptor segment contains 64-word or 1024-word pages: 0 = 1024 words 1 = 64 words
Paged/Nonpaged	- bit 28 indicates whether the descriptor segment is paged or not: 0 = paged 1 = nonpaged

PROCEDURE BASE REGISTER (PBR)

Format:

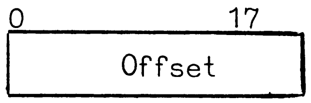


Function:

The procedure base register contains the segment number of the procedure segment currently in execution when the processor is not in Absolute mode.

INSTRUCTION COUNTER (IC or ICTC)

Format:



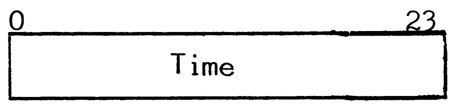
Function:

The instruction counter contains the offset relative to the beginning of the segment containing the instruction being processed by the control unit whenever the processor is in Append mode. When in Absolute mode, the IC or ICTC contains the absolute address of the instruction being executed by the control unit. When used in address modification the IC or ICTC contains an index value.

(Note that IC is used here as equivalent to ICTC, which is used in Appendix D.)

TIMER REGISTER (TR)

Format:



Function:

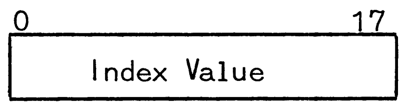
The timer register provides a program interrupt at the end of a variable item; the maximum total elapsed time is four minutes.

The contents of the timer register is decremented by one either upon each memory access or every 15.625 microseconds; the setting, under control of a manual switch, controls which way this register is decremented.

In Slave mode, a timer runout fault occurs when the count reaches zero. When the processor is in Master mode, no fault occurs and the counter starts decrementing from a maximum count after zero is reached. When the processor is returned to Slave mode after zero has been reached in Master mode, the timer runout fault occurs.

INDEX REGISTERS (X_0 to X_7)

Format:

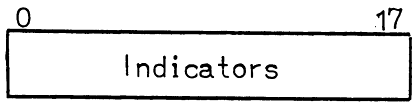


Function:

When used for address modification, these registers contain index values. In fixed-point operations, the index registers may be used as operand registers.

INDICATOR REGISTER (IR)

Format:



Function:

The indicator register contains all program accessible processor indicators which show the processor's states.

Each indicator occupies a specific bit position, where 1 = indicator ON and 0 = indicator OFF.

These indicators are set by the processor, either as the primary result of execution of instructions that set indicators, or as the secondary result of instructions that produce other results.

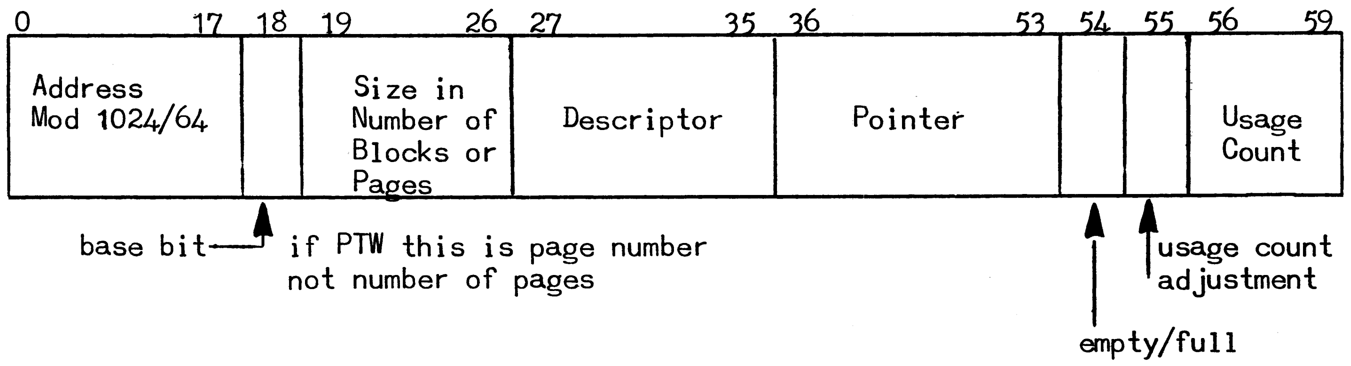
Bit	Indicator	Action
0	Zero	Set ON when contents of a processor register (A, Q, AQ, X_n, IR, TR), or an adder involved in arithmetic operation or a comparison is set to all 0's.
1	Negative	Set ON when contents of bit position 0 of a processor register (A, Q, AQ, Xn), or an adder after an arithmetic operation or comparison is set to 1.
2	Carry	Set ON when a carry is generated out of bit position 0 as a result of a left shift, addition, subtraction or comparison.
3	Overflow	Set ON when overflow occurs after execution of an arithmetic instruction other than logical add, logical subtract, compare, or add to base n instruction.

4	Exponent Overflow	Set ON if there is overflow from the exponent register (exponent > +127) resulting from floating-point arithmetic operation. Indicator remains ON until turned OFF by an LDI, RET, RTCD, RCU, or TEO.
5	Exponent Underflow	Set ON when there is underflow from the exponent register (exponent < -128) from a floating-point arithmetic operation. Indicator remains ON until turned OFF by one of the following: LDI, RET, RTCD, RCU, or TEU.
6	Overflow Mask	Set ON to prevent occurrence of an overflow fault when there is an overflow, exponent overflow, or exponent underflow (bit 3, 4, or 5 ON). When the overflow mask is cleared to 0 (unmasked), no fault is generated from previously set overflow or underflow indicators. The status of the overflow mask indicator does not affect the setting, testing, or storing of the overflow or underflow indicators. The overflow mask indicator is turned ON or OFF by the following: LDI, RCU, RET, or RTCD.
7	Tally Runout	Indicator shows whether a repeat instruction terminated because a specified termination condition was met or whether a tally count reached 0. The indicator is set ON by: 1. A repeat RPT or repeat double RPD instruction terminating because a tally count reached 0. 2. A repeat link RPL instruction terminating when a 0 link address or a zero tally is encountered.

	3. A tally count reaching 0 in an indirect and tally address modification (DI, DIC, ID, IDC, AD, SD, SC, SCR). The indicator is not affected by an indirect-to-segment, indirect-to-base, or fault modification. The indicator is set OFF when a specified termination condition is met in a RPT, RPD, or RPL instruction.
8 Parity Error	Set ON when a parity error is detected during access of one or both memory locations of a Y-pair. Indicator is turned OFF by: LDI, RET, RTCD, or RCU.
9 Parity Mask	Set ON to prevent occurrence of a parity error fault trap (bit 8 ON). The indicator is turned ON or OFF by: LDI, RCU, RET, or RTCD. When the parity mask indicator is cleared to 0 (unmasked), no fault is 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.
10 Absolute Mode	Set ON when the processor is in Absolute (Direct) mode of addressing and in Master mode of execution. The indicator is set to 0 (OFF) when the processor is in Append mode of addressing and either Master or Slave mode of execution.
11-17	Unused.

ASSOCIATIVE MEMORY REGISTERS (AR)

Format: (for each register)



Function:

The 16 associative memory registers constitute high-speed storage for recently referenced segment or page locations. Use of the associative memory registers precludes the need to access regular memory during repeated accesses to the same segment or page. Contents of the associative memory register differ--either the segment descriptor word (SDW) or the page table word (PTW).

SDW IN REGISTER	BITS	PTW IN REGISTER
High order 18 bits of absolute address of segment or its page table if paged	0-17	High order 18 bits of absolute address of page
Bit 18 = 1	18	Bit 18 = 0
Number of blocks in unpaged segment or number of pages in paged segment (taken from size field of SDW). Maximum = 256.	19-26	Page number; the increment added to the origin (bits 0-17) to locate the PTW. Maximum = 256.
Control bits (from SDW)	27-35	Control bits (from SDW and control field of PTW)
0 = 1024; 1 = 64 word block or page	27	0 = 1024; 1 = 64 word page

SDW IN REGISTER	BITS	
0 = paged; 1 = unpaged	28	0 = PTW in associative store
not used	29	0 = page written into; 1 = page not written.
Bit positions of the PTW and corresponding positions of the associated segment descriptor word are combined in positions 30-35 to reflect the most restrictive access and class control bits.	30-35	Same as SDW.
0 = not written in Slave; 1 = can be written in Slave	30	Same as SDW.
0 = access in Master; 1 = access in Master or in Slave mode	31	
Not used	32	
An alternate use of bits 30-32 is as one of the codes for a directed fault (see Faults).	30-32	
Class bits: 000 - directed fault in 30-32 001 - data segment 010 - procedure - Slave 011 - procedure - execute only 100 = procedure - Master	33-35	
Segment number of the segment	36-53	Same as SDW.
0 = contents no longer valid (a CAM places a 0 in this bit position) 1 = contents still valid (full)	54	Same as SDW.
0 = usage count current; 1 = usage count needs adjustment	55	Same as SDW.
Usage count; see below.	56-59	Usage count; see below.

The usage count is a number from 0 to 15, showing relative use of each register. When a new word is brought into associative memory, it is given a usage count of 15 and it replaces the word that has count 0. Hardware decrements by

1 the count in the other 15 registers. Each time a word already in an associative memory register is referenced, its usage count increases to 15 and all bypassed registers are decremented by 1. In this way, the most recently referenced associative memory word has the largest usage count and all words below it are in order depending upon use.

CHAPTER 5 ADDRESSING -- SEGMENTATION AND PAGING

INTRODUCTION

The 645 processor generates an effective address (EA) for the instruction which it is executing, the operand it is dealing with, or the indirect word it is fetching. The various means of effective address formation are explained in Chapter 6.

An effective address consists of a segment number and an offset within that segment. (A segment may be defined as an array of words, each of which is directly addressable by the processor). The processor uses the EA to compute an absolute memory address. The computation of this absolute memory address can be done in one of two ways, depending on whether the processor is in Absolute or Append mode. The processor then uses the computed absolute memory address to access the information required to perform the particular operation requested.

SEGMENTATION

Any address in the 645 consists of a pair of integers (segno,offset). The range of segno and offset is 0 to $2^{18}-1$. segno is called the segment number; offset, the offset within the segment. Word(segno,offset) is accessed through a hardware register which is the segno'th word in a table called a descriptor segment. The address of the descriptor segment is recorded in a processor register called a descriptor segment base register (DBR). Each word of a descriptor segment is called a segment descriptor word (SDW).

The following is a simplified description of the appending process for the case in which segments are not paged.

The DBR contains the following values:

1. the absolute memory address of the descriptor segment
2. the length of the descriptor segment

The detailed format of the DBR is described in Chapter 4.

The SDW for a given segment segno contains the following values:

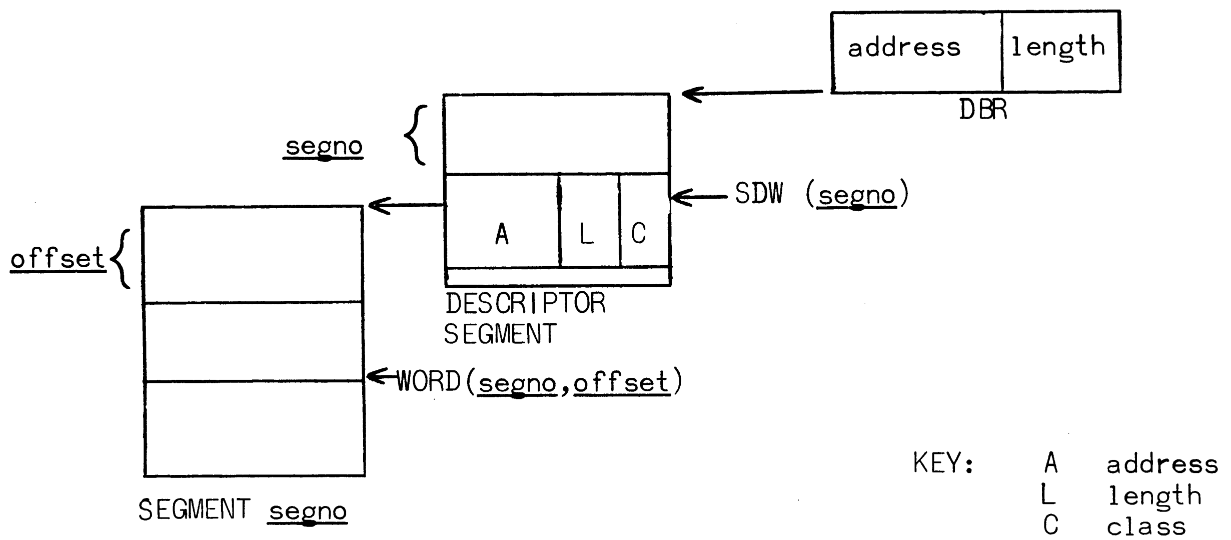
1. the absolute memory address of segment segno
2. the length of the segment segno
3. class bits which indicate the type of a segment, or which indicate a directed fault.

The detailed format of an SDW is given in Appendix E.

The algorithm used by the processor for referencing word(segno,offset) is as follows:

1. If the length of the descriptor segment is less than segno, generate an illegal procedure fault.
2. Access SDW(segno) at the absolute address of the descriptor segment plus segno.
3. If the class bits indicate a directed fault, generate a directed fault (used by the software to signify a "missing segment").
4. If the length of segment segno is less than offset generate an illegal procedure fault.
5. If the class bits of segment segno are incompatible with the reference, generate a fault.
6. Reference the word at the absolute address of the segment segno plus offset.

The following figure depicts information used to reference word(segno,offset) in segment segno.



Hardware Segmentation in the 645

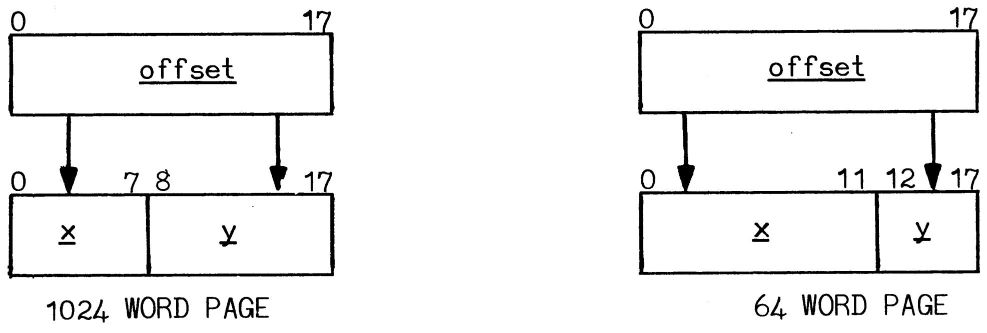
PAGING

A bit in an SDW indicates whether the corresponding segment is paged or not paged. Another bit in the SDW indicates whether the page size is 64 or 1024 words. Analogous bits in the DBR serve the same purpose for the descriptor segment.

An element of a paged segment is the y th word of the x th page of the segment, where x and y are defined as:

y	offset modulo(page size)
x	(offset - y)/page size

Since page size is either 1024(2^{10}) or 64(2^6), the processor can compute x and y from the 18-bit binary representation by merely dividing offset into two parts. The right-hand part, which consists of 10 (for 1024 word pages) or 6 (for 64 word pages) least significant bits of offset, represents the binary value of y. The left-hand part, which consists of the 8 (for 1024 word pages) or 12 (for 64 word pages) most significant bits of offset, represents the binary value of x. The following figure illustrates the division of offset.



Hardware Interpretation of the Word Number

The page table of a segment is an array of physically contiguous words in core memory. Each element of this array is called a page table word (PTW).

A given page table word contains the following items:

1. the absolute memory address of a page
2. class bits which indicate the type of segment to which the page belongs, or indicate a directed fault.

The detailed format of a PTW is given in Appendix E.

With paging, the address field in the DBR contains the absolute address of the page table of the descriptor segment; the address field in the SDW contains the absolute address of the page table of a segment.

The full algorithm used by the processor to access word(_segno_,_offset_) is as follows: (Referring to the figure following this algorithm will be helpful.)

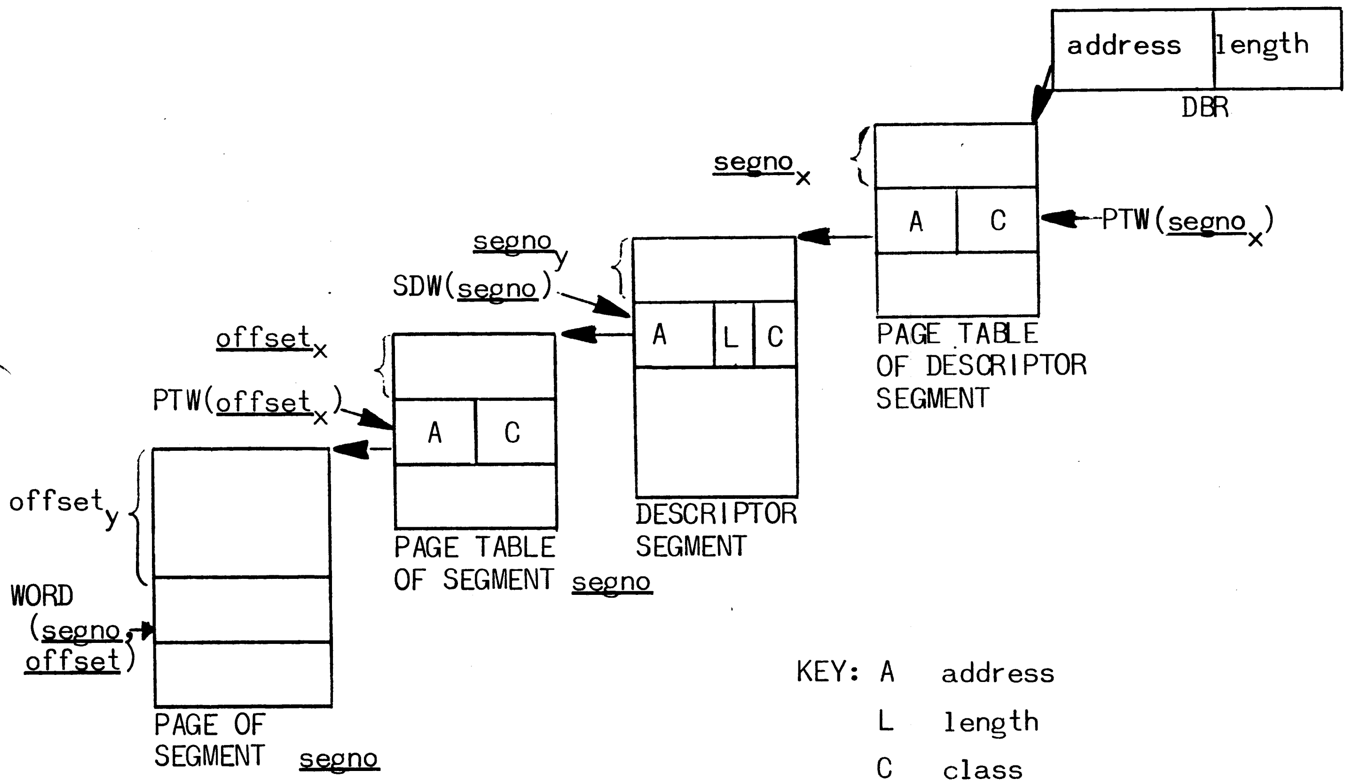
1. If the length of the descriptor segment is less than _segno_, generate an illegal procedure fault.
2. Split _segno_ into $\underline{segno}_x$ and $\underline{segno}_y$ such that:
   $\underline{segno}_y$ = _segno_ modulo(_page size_) and
   $\underline{segno}_x$ = (_segno_ - $\underline{segno}_y$)/_page size_.
   (This corresponds to the division of an offset into fields _x_ and _y_ described above.)
3. Access PTW($\underline{segno}_x$) at the absolute address of the descriptor segment page table plus $\underline{segno}_x$.
4. If the class bits of PTW($\underline{segno}_x$) indicate a directed fault, generate a directed fault (used by the software to signify a "missing page" of a descriptor segment).
5. Access SDW(_segno_) at the absolute address defined by the absolute address in PTW($\underline{segno}_x$) plus $\underline{segno}_y$.
6. If the class bits of SDW(_segno_) indicate a directed fault, generate a directed fault (used by the software to signify a "missing segment").
7. If the length of segment _segno_ is less than _offset_, generate an illegal procedure fault.
8. Split _offset_ into $\underline{offset}_x$ and $\underline{offset}_y$ such that:
   $\underline{offset}_y$ = _offset_ modulo(_page size_) and
   $\underline{offset}_x$ = (_offset_ - $\underline{offset}_y$)/_page size_.
9. Access PTW($\underline{offset}_x$) at the absolute address of the page table for segment _segno_ plus $\underline{offset}_x$.
10. If the class bits of PTW($\underline{offset}_x$) indicate a directed fault, generate a directed fault (used by the software to signify a "missing page").
11. Combine the class bits of PTW($\underline{offset}_x$) with the class bits of SDW(_segno_)

to produce the most restrictive access. If the resulting access rights are incompatible with the reference, generate a fault.

12. Reference the word at the absolute address contained in PTW(<u>offset</u>$_x$) plus <u>offset</u>$_y$.

See Appendix F for a detailed interpretation of "class bits".

The following illustration depicts the ference to a word(<u>segno</u>,<u>offset</u>)in segment <u>segno</u>. Both the descriptor segment and the referenced segment are paged.



<u>Hardware Segmentation and Paging in the 645</u>

MODE OF ADDRESSING (ABSOLUTE/APPEND)

In the Absolute mode, the effective address becomes the absolute core memory address. The appending mechanism is bypassed. Addresses are limited to the lower 256K of memory. Absolute addressing may be conceptualized as the creation of a segment with a base address of all zeroes where all effective addresses are offsets to the base of this segment. The intersegment fetch of indirect words and operands is optional through the use of bit 29 of the instruction word, or ITB, or ITS modifiers. In other words, indirect words and operands may be addressed in Absolute mode or via the appending mechanism.

In Append mode, the appending mechanism is employed for all instructions, indirect words, and operand fetches. In the Append mode, the processor generates an effective address which consists of a segment (or page) number and a word number (offset) within the segment (or page). The 18-bit effective address is either added to a base address (unpaged), or the word number field of the EA is concatenated to the base address (paged). More complete details of how the processor performs appending have already been discussed in the sections titled "SEGMENTATION" and "PAGING".

CHANGING ADDRESS MODES

The control unit is normally in the Append mode but the Absolute mode is entered temporarily when an XED instruction pointing to a fault or interrupt vector is "forced". The mode becomes Absolute if either instruction is a satisfied transfer, other than TSS, without bit 29 = 1, or ITB, or ITS indirection. The control unit remains in Absolute until TSS is executed or until a transfer with bit 29 = 1 takes place.

The Absolute mode may also be entered or exited if in Master mode one of the following instructions brings in a different state in the bit position corresponding to the absolute indicator (bit 10 of the Indicator register): RET, RTCD, RCU.

CHAPTER 6 EFFECTIVE ADDRESS FORMATION

INTRODUCTION

The effective address on the 645 consists of two parts, a segment number and an offset. During the formation of an effective address these two portions are stored in temporary registers used as working registers by the processor. A tentative segment number is stored in the temporary base register (TBR); a temporary offset within the indicated segment is stored in a computed address register (CAR). When each effective address computation has been completed, the contents of the TBR and CAR are presented to the appending unit of the processor for conversion to an absolute address (see Chapter 5).

In this chapter the description of effective address formation is divided into two parts. The first part describes the type of effective address formation involving only the offset, that is, the contents of the CAR. The contents of the TBR remain constant and are obtained at the beginning of the effective address formation by copying the contents of the PBR into the TBR. In this type of effective address formation, references remain local to the segment specified by the constant value contained in the TBR.

The second part of this chapter describes the type of effective address formation for which the contents of the TBR can be modified as well as the contents of the CAR. This feature is necessary in order to permit intersegment referencing on the 645.

The first type of effective address formation is analogous to the type of address modification on the 635 and does not make explicit use of segment numbers. The second type of effective address formation is unique to the 645 and makes use of segment numbers stored either in word pairs in core memory or in address base registers (ABR's).

In actual practice the two types of effective address formation can be intermixed. Note that in cases where effective address calculations are chained together via registers or indirect words, the processor must convert intermediate effective addresses to absolute addresses in order to fetch the next item in the chain.

EFFECTIVE ADDRESS FORMATION INVOLVING SEGMENT OFFSET ONLY

For the following types of effective address formation, the segment number is assumed to remain constant.

Modifier Field of an Instruction Word

Bits 30-35 of an instruction or indirect word constitute the modifier field for address modification. If bits 30-35 are all zeros (no modification), the leftmost 18 bits of the instruction word constitute the offset portion of the effective address and are loaded into the CAR.

If bits 30-35 of an instruction or indirect word are not all zeros, then one of four possible address modification types is specified. Bits 0-17, possibly subject to further modification, form a tentative offset in the CAR. Then depending on the type of modification, the contents of the CAR specify the offset of either the operand or an indirect word.

The modifier field has the format:

Modifier Designator M_d		Register (or Tally) Designator R_d or T_d			
30	31	32	33	34	35

The modifier designator (M_d) portion of the modifier field specifies the type of modification and the register designator (R_d) portion specifies the variation within the specified type.

General Types of Modification

There are four general types of modification: Register, Register Then Indirect, Indirect Then Register, and Indirect Then Tally:

TYPE	MODIFIER	MODIFICATION
Register R	00	Index according to the register designator R_d.
Register Then Indirect RI	01	Index according to register designator R_d. The result of the register modification is the offset of an indirect word or pair of words. The indirect word is retrieved and modification continues as specified by the indirect word. If the indirect word or pair specifies indirection, the indirect sequence continues.
Indirect Then Register IR	11	The indirect word or pair is first retrieved. If address modification is specified by the indirect word or pair, it is carried out. If the modification is again IR, retrieval and modification continue until an R, IT, or RI modification is encountered. The safe-stored contents of the register are then added and modification is concluded in the code of an R or IT. Modification continues in accordance with the indirect word in the case of an RI.
Indirect Then Tally IT	10	The address and modifier fields of the indirect word are substituted according to R_d, which is used as the tally designator. There are 13 variations of IT modification. IT modification allows both indirect addressing and register modification and permits automatic incrementing and decrementing of the fields of the indirect word.

Register, Register Then Indirect, and Indirect Then Register

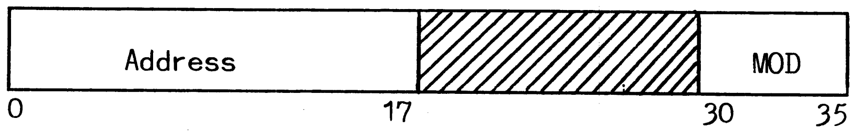
Bits 32-35 are used as a register designator R_d when bits 30 and 31 specify Register, Register Then Indirect, or Indirect Then Register (R, RI, IR) modification:

REGISTER		BITS 32-35	MNEMONIC	MODIFICATION y = address field of instruction word
none		0000	N	y becomes the offset.
INDEX	X0	1000	0	y + $C(X_n)$ becomes the offset.
	X1	1001	1	
	.	.	.	
	.	.	.	
	X7	1111	7	
ACCUM.	A (upper)	0001	AU	y + $C(A)_{0-17}$ becomes the offset.
	A (lower)	0101	AL	y + $C(A)_{18-35}$ becomes the offset.
QUOT.	Q (upper)	0010	QU	y + $C(Q)_{0-17}$ becomes the offset.
	Q (lower)	0110	QL	y + $C(Q)_{18-35}$ becomes the offset.
INST. CNTR.	IC	0100	IC	y + C(IC) becomes the offset.
DIRECT OP.	none	0011	DU	y, 00...0 becomes the operand itself. (DU means direct upper. Contents of the address field followed by 18 zeros become the operand.) See note below.
	none	0111	DL	00...0, y becomes the operand itself. (DL means direct lower. 18 zeros followed by the address field become the operand.) See note below.

Note: There is a contradiction between the DU and DL variations and the RI modification. DU and DL specify use of the instruction address field without modification and the RI modification specifies modification of the address field. If used together, the results are unreliable.

Indirect Word

The format of the indirect word used with Register, Register Then Indirect, or Indirect Then Register modifiers is:

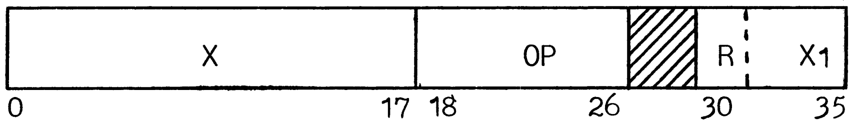


Register Modification (R)

The registers that can be used in R modification are the DU, DL, AU, AL, QU, QL, IC, and eight index registers (X_0-X_7). Register modification can be indicated in the instruction word or in an indirect word. Wherever it is indicated, R modification terminates the address modification sequence.

Example:

Instruction Word



The contents of index register X1 are added to X in order to reference the operand.

Register Then Indirect Modification (RI)

In RI modification, the following sequence of events occurs:

1. A register modification of any variation except DU or DL is performed. (This step is not performed when the instruction or indirect word specifies "no modification" by having N in the modifier field. This permits an indirect word to be retrieved without performing a register modification.)

2. The indirect word is retrieved from memory, using the tentative address resulting from the R modification.

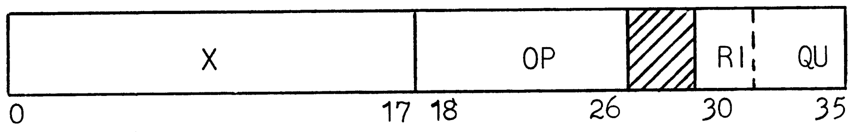
3. The processor examines M_d and R_d of the retrieved word and if a modification is specified therein, it is performed on the indirect word's address field.

4. Modification continues from one indirect word to the next as long as RI, IR, or certain variations of IT modification are specified.

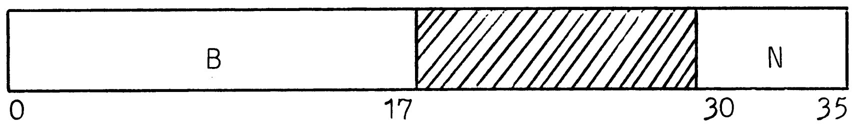
5. When an indirect word specifying R modification or certain variations of IT modification is found, the modification is performed on the last indirect word retrieved. The result forms the offset portion of the effective address, and the indirect sequence is terminated.

Example:

Instruction Word



Indirect Word at X + QU



The operand at B is referenced.

Indirect Then Register Modification (IR)

The sequence of steps in IR modification is:

1. The modifier field of the instruction word or preceding indirect word is saved for use in making the final modification.

2. An indirect word is obtained from memory using: a) the address field of the instruction word, or b) the address field of the preceding indirect word as the offset of the indirect word to be retrieved.

3. Any modification specified in the modifier field of the first indirect word retrieved is treated as follows:

   a. If the modification is another IR, the modifier field of this indirect word is saved (replacing the modifier field previously saved). Another indirect word is obtained, using the address field of the current indirect word. As long as IR modification continues, each saved modifier field replaces the modifier previously saved until a modification other than IR is encountered.

   b. If the modification is RI, then RI is performed as it always is, ignoring the fact that it follows an IR modification. The saved modifier field of the IR instruction word or indirect word is not used and it is not destroyed.

   c. If a series of modifications has IR modification followed by RI modification and then returns to more IR modification, the following occurs:

      (1) The modifier field of the first IR modification is saved and each new modifier field replaces the saved modifier field.

      (2) The modification of the RI type proceeds as usual, having no effect on the saved modifier.

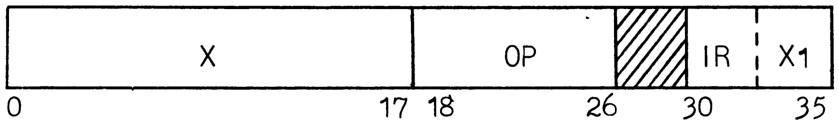
      (3) The return to the IR type of modification causes the modifier field to again replace the modifier previously saved (before the RI modification).

4. The chain of modifications is terminated as follows: If the indirect word is an R or certain variations of IT modification, the saved modifier is used to obtain the effective operand address. The contents of the register specified by the saved modifier are added to the address field of the current R or IT indirect word to obtain the offset portion of the effective address.

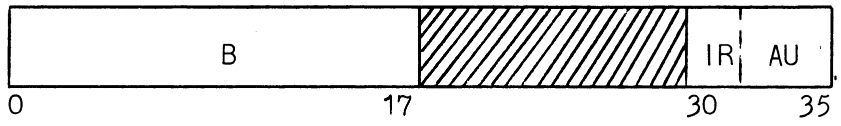
It is possible for the <u>last saved modifier</u> to have an N variation. In this case the address field of the current R or IT indirect word is used without modification.

Example of IR Modification:

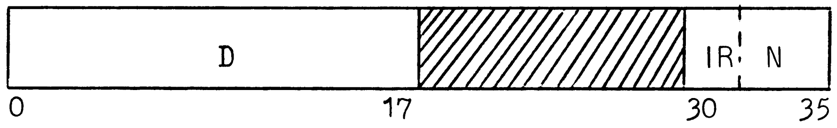
Instruction Word



Indirect Word at X



Indirect Word at B



The modification (X1) of the instruction word is saved and X is retrieved. Modification field of X (AU) replaces the instruction word modification field and B is retrieved. The operand at D and the saved register (AU) is referenced.

Indirect Then Tally Modification (IT)

When bits 30-31 specify IT modification, bits 18-29 of the indirect word hold the tally count. The tally count indicates how many times the access field is to be used for access. A maximum of 4096 accesses can be made. The tally count is always decremented or incremented by 1. Bits 32-35 indicate the 13 possible variations of IT modification that can occur.

MNEMONIC	BITS 32-35	VARIATION
I	1001	Indirect only.
ID	1110	Increment address, Decrement tally.
DI	1100	Decrement address, Increment tally.
AD	1011	Add Delta to address field.
SD	0100	Subtract Delta from address field.
DIC	1101	Decrement address, Increment tally and Continue.
IDC	1111	Increment address, Decrement tally and Continue.
SC	1010	Sequence Character.
SCR	0101	Sequence Character Reversed.
CI	1000	Character from Indirect.
ITS	0011	Indirect To Segment. ITS recognized only if it appears in an indirect word of an RI or IR modification series and only if the referring offset is even.
ITB	0001	Indirect To Base. ITB recognized only if it appears in an indirect word of an RI or IR modification series and only if the referring offset is even.
FT1	0000	Fault Tag 1
FT2	0110	Fault Tag 2
FT3	0111	Fault Tag 3

Tally Word

The format of tally words used in I, ID, DI, AD, SD, DIC, IDC, SC, SCR, and CI modification is:

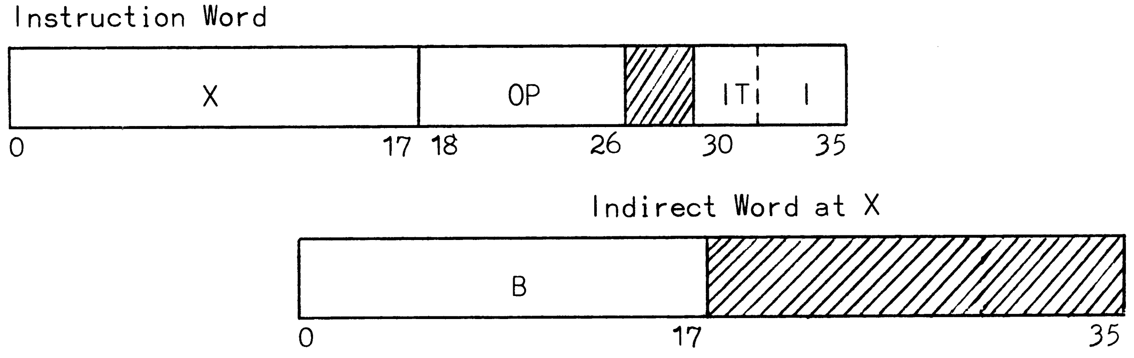
Address	Tally	Control
0 17	18 29	30 35

The meaning of control field depends upon the specific type of modification.

Indirect Only (I)

The processor retrieves only one indirect word, which contains the effective address in bits 0-17. The tally and control fields are not used.

Example:



The operand at B is referenced.

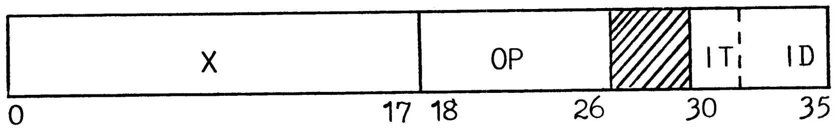
Increment Address, Decrement Tally (ID)

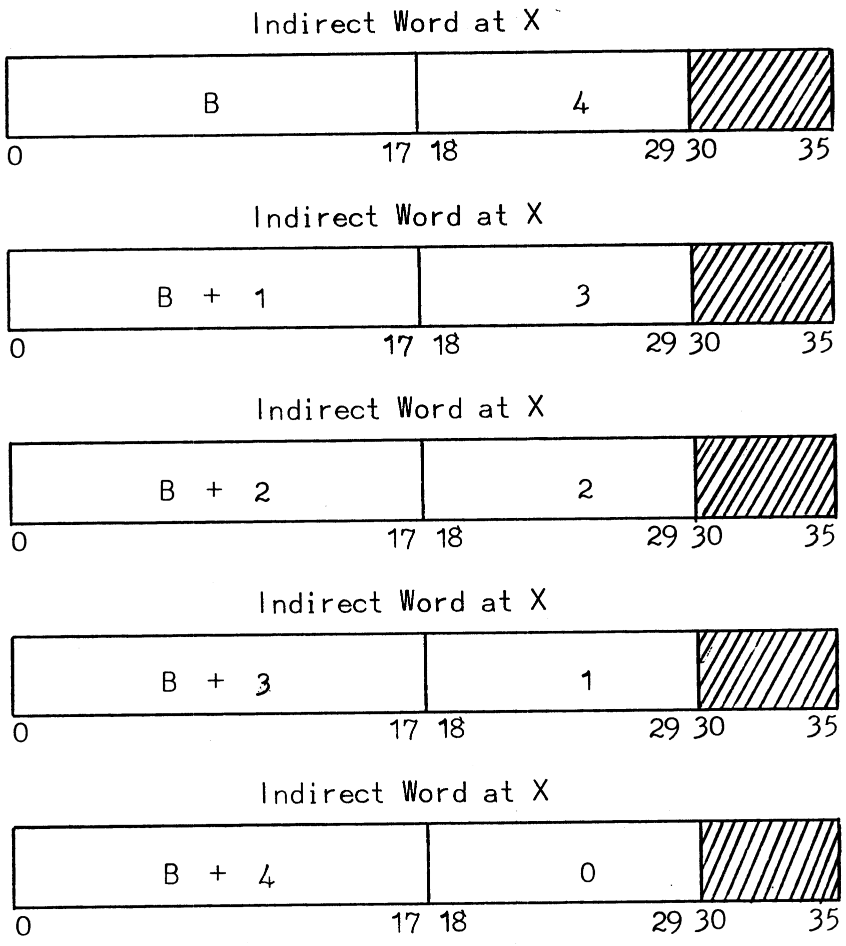
The indirect word contains a tally count. Each time the instruction is executed, the tally is decremented and the address field of the indirect word is incremented, causing consecutive memory locations to be accessed. In each case, the operand is referenced before the tally and address fields are altered.

Increment Address, Decrement Tally (ID), Continued

Example: (4 instruction executions)

Instruction Word





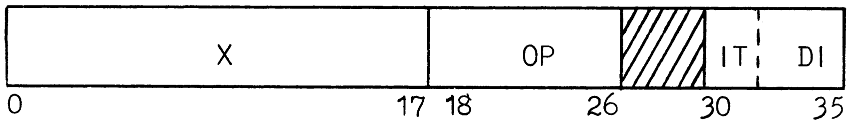
Operand at B is referenced; tally goes to 3.
Operand at B+1 is referenced; tally goes to 2.
Operand at B+2 is referenced; tally goes to 1.
Operand at B+3 is referenced; tally goes to 0
and sequence ends.

Decrement Address, Increment Tally (DI)

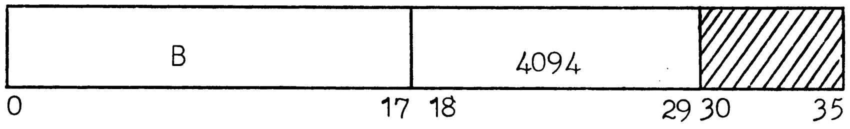
The indirect word contains a tally count and an address field. The control field is unused. Each time the instruction is executed, the tally is incremented and the address field of the indirect word is decremented, causing consecutive memory locations to be accessed. In each case, the tally and address field are altered before the operand is referenced.

Example: (2 instruction executions)

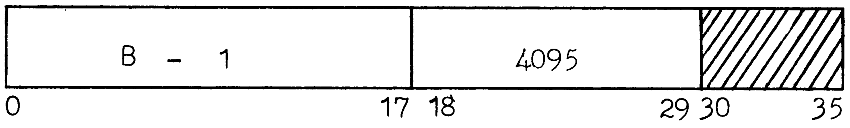
Instruction Word



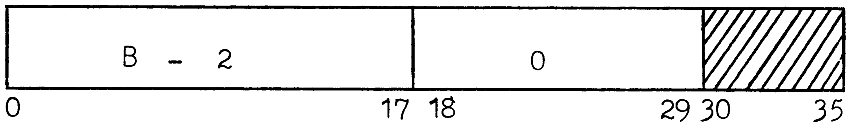
Indirect Word at X



Indirect Word at X



Indirect Word at X

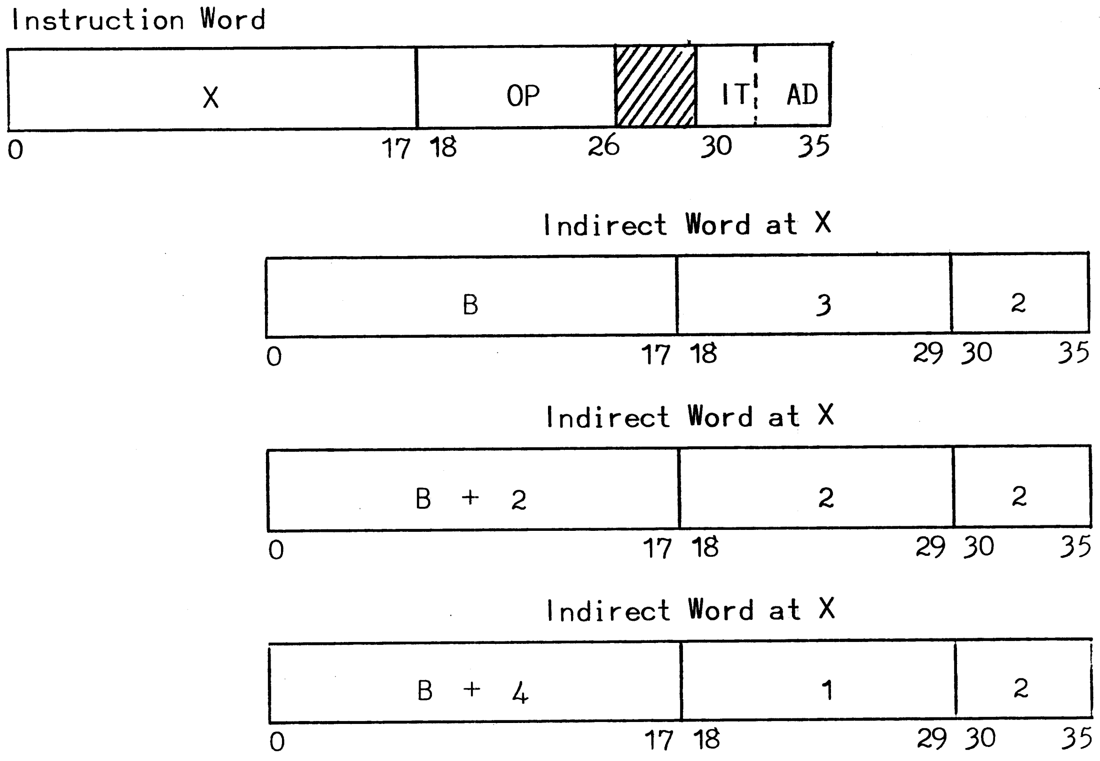


The tally at X is incremented and the operand at B - 1 is referenced. The tally is incremented and B - 2 is referenced. The tally goes to 0 and the sequence ends.

Add Delta to Address (AD)

The indirect word contains an address field, a tally count and a control field specifying the delta value to be added to the offset for each access. Each time the instruction is executed the operand is referenced before the tally is decremented.

Example: (2 instruction executions)



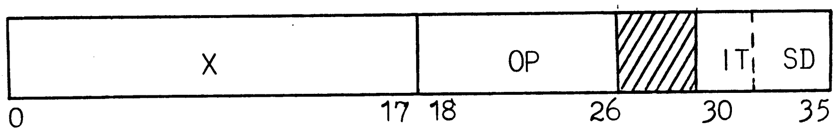
Operand at B is referenced; tally goes to 2. Operand at B + 2 is referenced; tally goes to 1. Operand at B + 4 is referenced and sequence ends.

Subtract Delta from Address (SD)

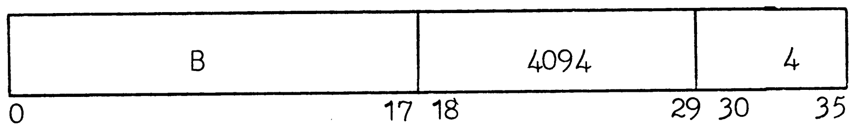
The indirect word contains an address field, a tally count and a control field specifying the delta value to be subtracted from the offset for each access. In each case, the tally is incremented before the operand is referenced.

Example: (2 instruction executions)

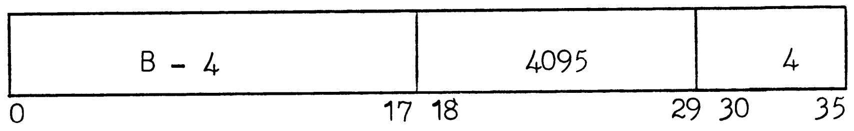
Instruction Word



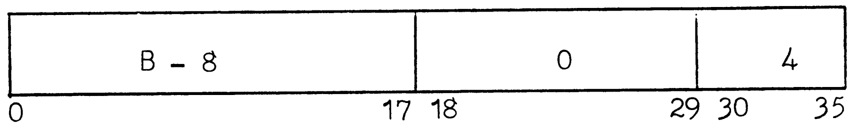
Indirect Word at X



Indirect Word at X



Indirect Word at X



Tally is incremented and the operand at B - 4 is referenced. Tally is incremented and B - 8 is referenced and sequence ends.

Character Handling in the SC, SCR, and CI Variations

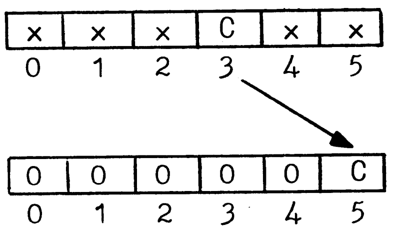
Following is a summary of character handling in the SC, SCR, and CI variations of the IT modification.

1. Each time a load instruction is executed an operand is taken from memory and loaded as a single word with the specified character right-justified in the register and the remaining character positions zero filled. In 6-bit operations, the character is in position 5. In 9-bit

character operations it is in position 3 as shown:

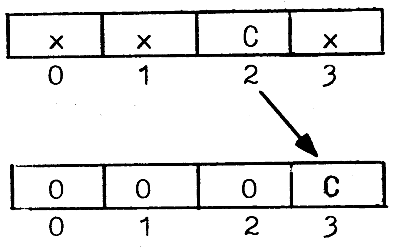
6-bit Characters

9-bit Characters



Memory Location

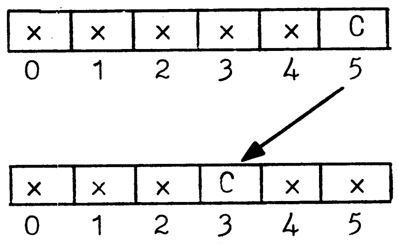
Register



2. When an operation places a result in memory, the right-justified character in the register replaces the specified character in the memory location of the operand. Other characters in the memory location remain unchanged. In 6-bit character operations, character 5 of the result replaces the specified character in the memory location. In 9-bit character operations, character 3 replaces the character in the memory location as shown:

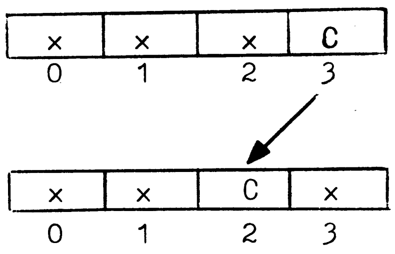
6-bit Characters

9-bit Characters



Register

Memory Location



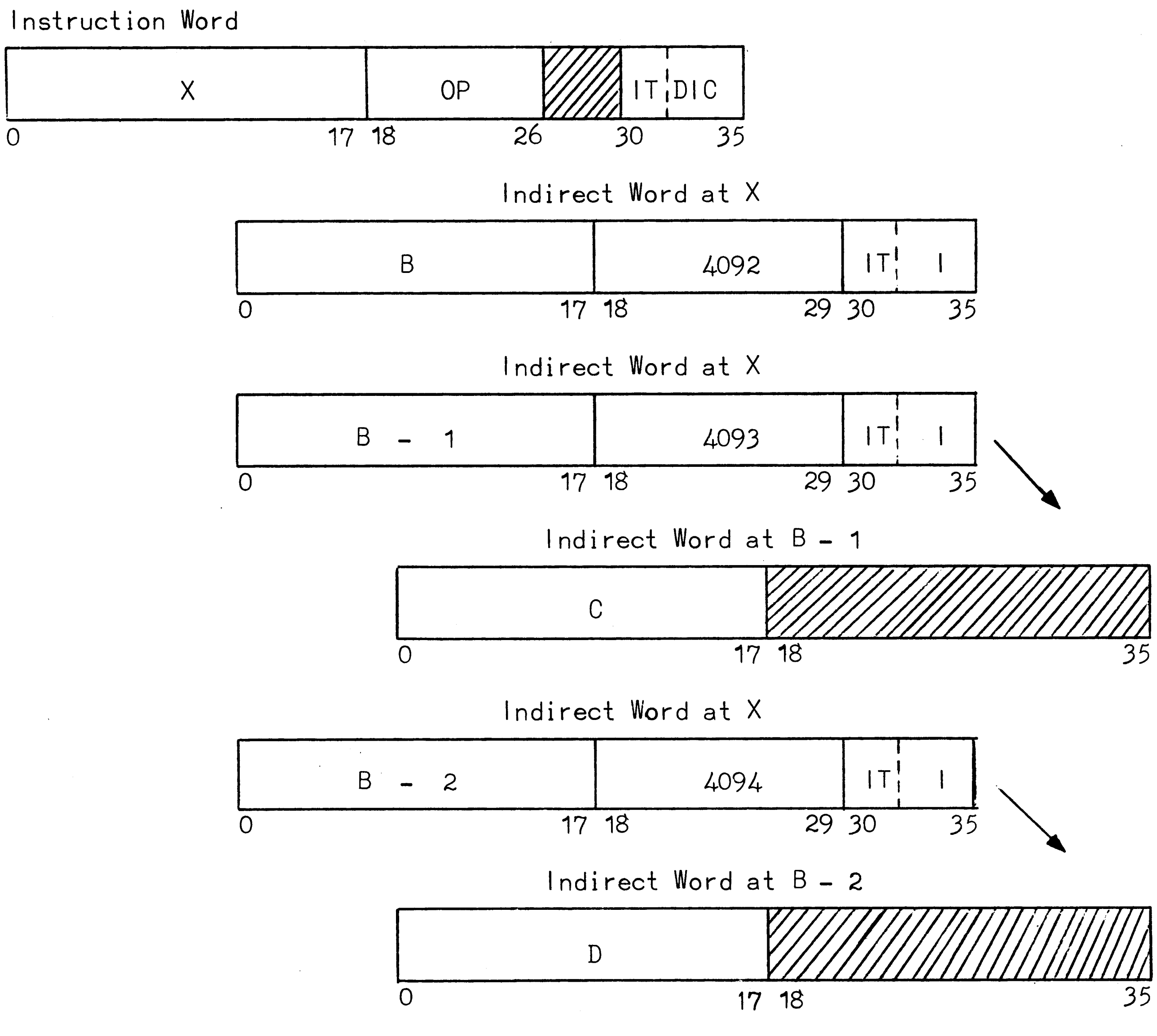
3. Character operations can only be used in operations involving the A-register or Q-register.

4. Tallying involves incrementing and decrementing of the character count as well as the tally count. The tally counts characters; that is, the number of character references. Address field modification takes place only after the character count goes to 3 or 5 for tally decrementing or goes to 0 for tally incrementing.

Decrement Address, Increment Tally, and Continue (DIC)

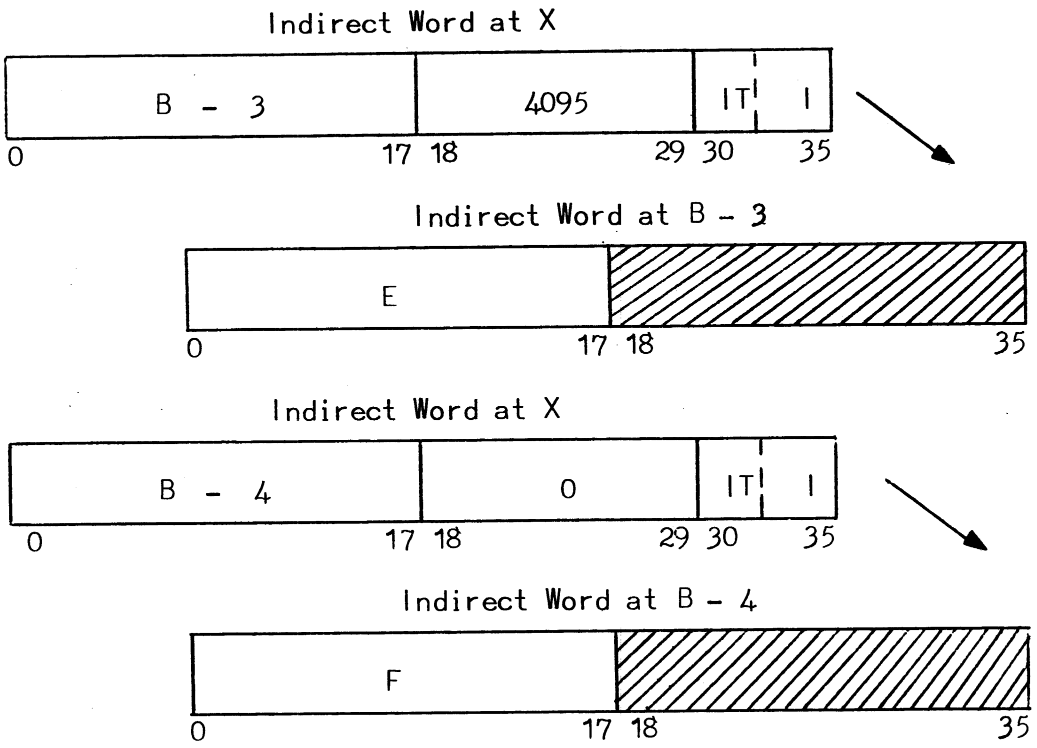
The DIC variation is similar to the DI variation except that the control field of the indirect word further modifies the offset. The modifier can be of any type as long as no index modification is involved, since the indexing adder is used by the tally phase of the modification. This means that when an R or RI modification is in the indirect word, the variation must be N (no index modification).

Example: (4 instruction executions)



(The example is continued on the next page.)

Example: Continued

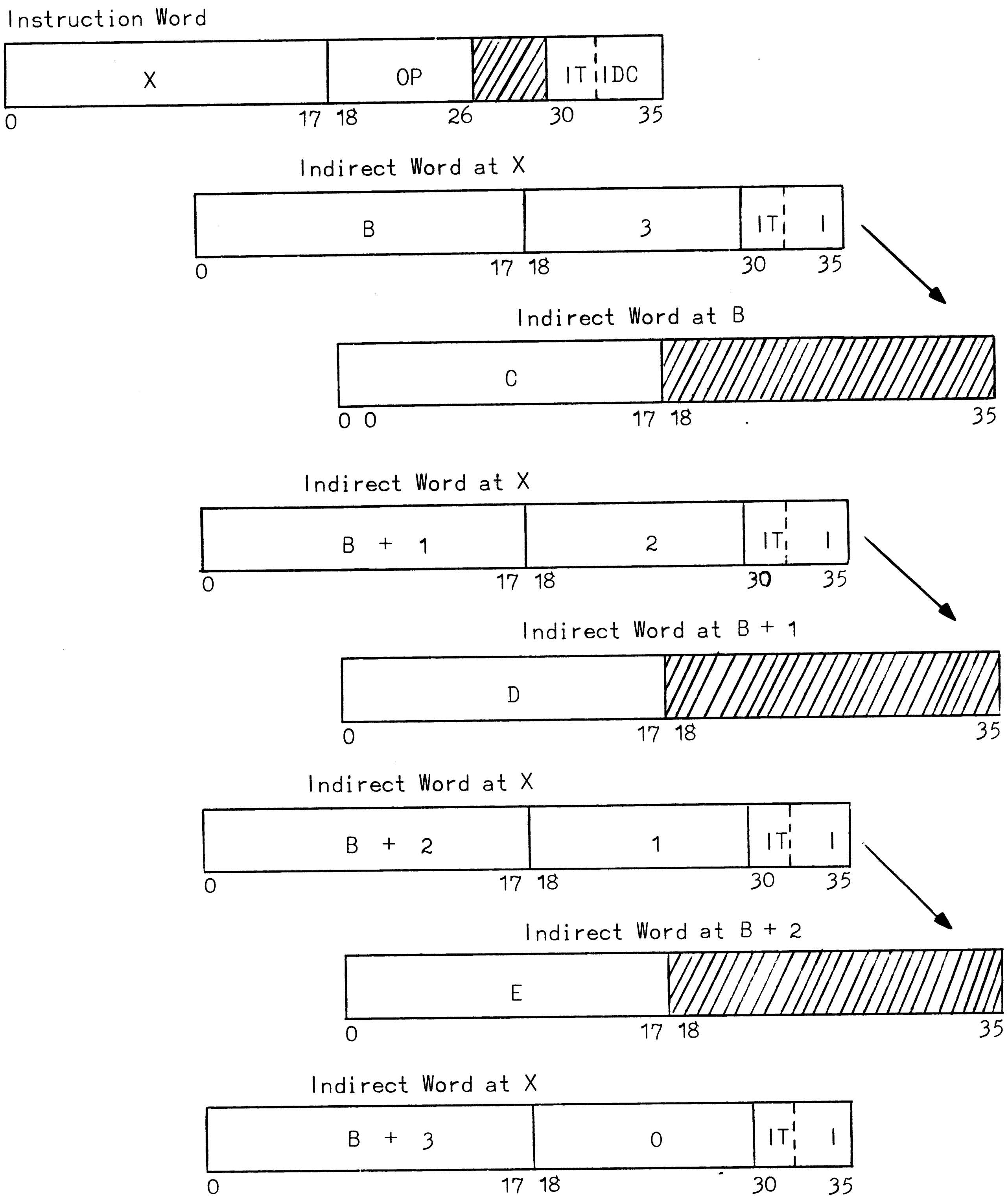


B is decremented; the tally goes to 4093
and operand at C is referenced, etc.

Increment Address, Decrement Tally, and Continue (IDC)

The IDC variation is similar to the ID variation except that the control field of the indirect word further modifies the offset. The modifier can be of any type as long as no index modification is involved, since the indexing adder is used by the tally phase of the modification. This means that when an R or RI modification is in the indirect word, the variation must be N (no index modification).

Example: (3 instruction executions)



The operand at C is referenced; the tally goes to 2; the operand at D is referenced, etc.

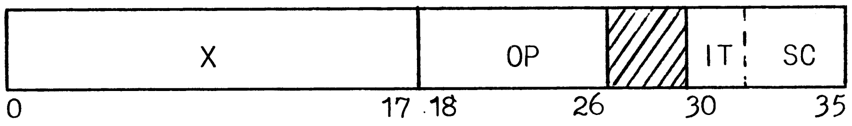
Sequence Character (SC)

The SC variation permits operations to be performed on 6-bit or 9-bit characters in sequential locations in memory. The operations involve only the A-register or Q-register. The indirect word contains an address field, a tally field and a flag in position 30 (0 for 6-bit and 1 for 9-bit). Bits 33-35 indicate which character position is being referenced (0-5 for 6-bit and 0-3 for 9-bit). Each time the instruction is executed the tally field is decremented and the character field is incremented after the operand is referenced. Each time the last character of the current word is referenced the address field is incremented and the character field is reset to zero.

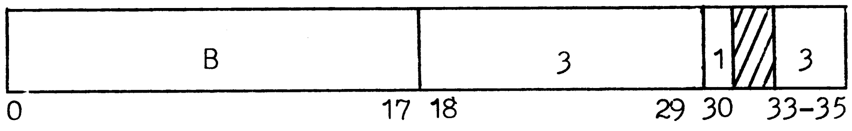
Example: (3 instruction executions)

Beginning at the last character of a word of 9-bit bytes:

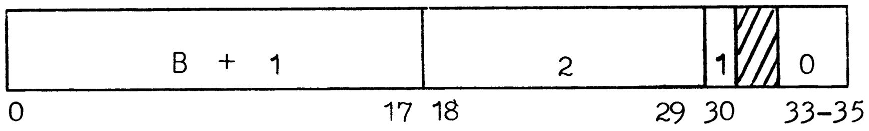
Instruction Word



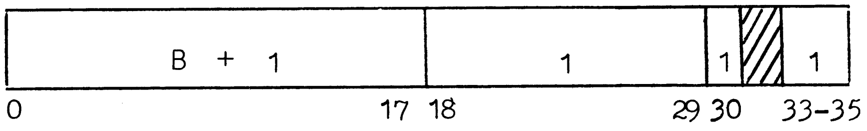
Indirect Word at X



Indirect Word at X



Indirect Word at X

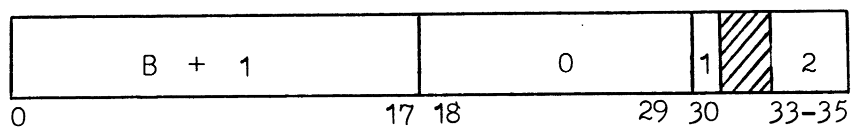


(Example is continued on the next page.)

Sequence Character (SC), Continued

Example: Continued

Indirect Word at X



Character #3 at B is referenced, then character #0 of B + 1, character #1 of B + 1, and then the sequence ends as the tally goes to 0.

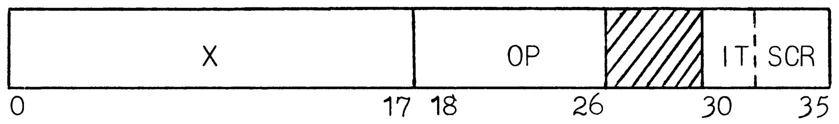
Sequence Character Reversed (SCR)

The SCR variation permits operations to be performed on 6-bit or 9-bit characters in sequential locations in memory. The operations involve only the A-register or Q-register. The indirect word contains an address field and a tally field and a flag in position 30 (0 for 6-bit and 1 for 9-bit). Bits 33-35 indicate which character position is being referenced (0-5 for 6-bit and 0-3 for for 9-bit). Each time the instruction is executed the tally field is incremented and the character field is decremented before the operand is referenced. Each time the first character (character 0) of the current word is referenced the address field is decremented and the character field is set to the number of the last character (character 3 or 5).

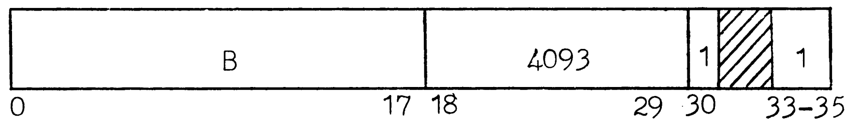
Example: (3 instruction executions)

Beginning at the second character of a word of 9-bit bytes:

Instruction Word



Indirect Word at X

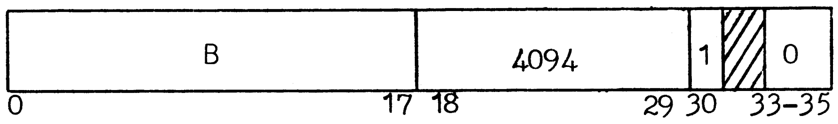


(Example is continued on the next page.)

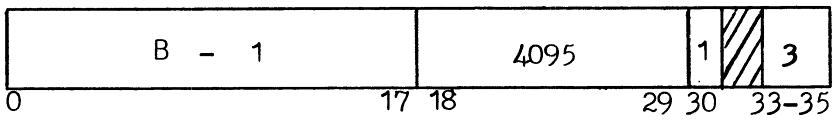
Sequence Character Reversed (SCR), Continued

Example: Continued

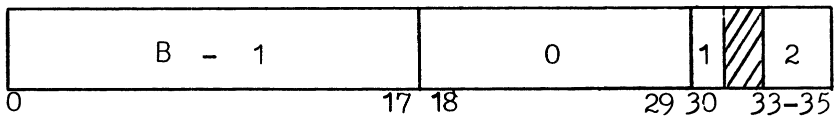
Indirect Word at X



Indirect Word at X



Indirect Word at X



The tally field of indirect X is incremented and the address and character fields decremented; character #0 of B is referenced, then character #3 of B-1, then character #2 of B-1, and then the sequence ends.

Character From Indirect (CI)

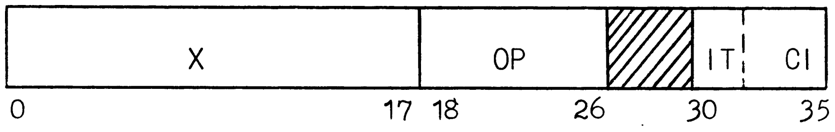
The CI variation permits repeated references to a single 6-bit or 9-bit character in memory. The operation involves only the A-register and Q-register. The indirect word contains an address field, a flag in position 30 (0 for 6-bit and 1 for 9-bit), and bits 33-35 indicate which character position is being referenced (0-5 for 6-bit and 0-3 for 9-bit). Each time the instruction is executed the tally field is decremented as the character is referenced. The address and character fields remain unchanged.

Character From Indirect (CI), Continued

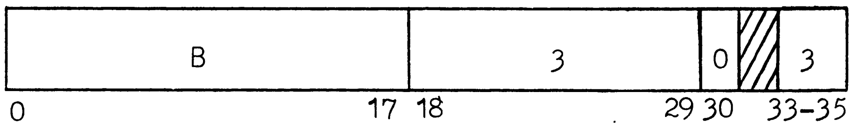
Example: (3 executions)

Beginning with the fourth character of a word of 6-bit characters:

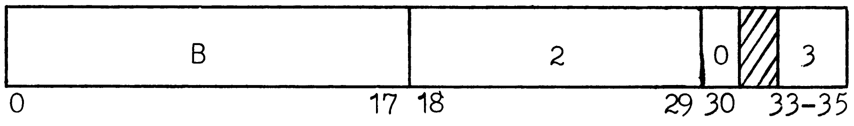
Instruction Word



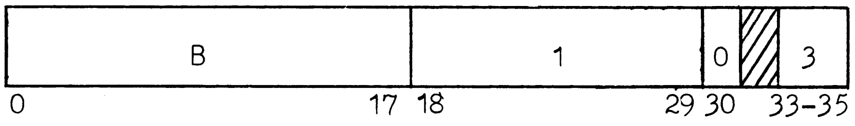
Indirect Word at X



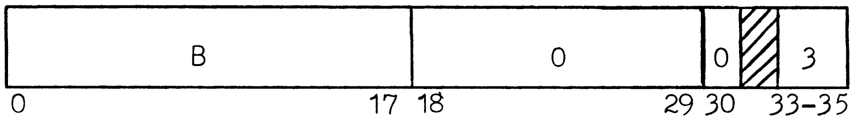
Indirect Word at X



Indirect Word at X



Indirect Word at X



Character #3 of B is referenced and tally goes to 2; character #3 of B is referenced and tally goes to 1; character #3 of B is referenced and sequence ends.

Other Types of IT modification

There are five types of IT modification which are different from the types of IT modification described previously. Two of these, Indirect To Segment (ITS) and Indirect To Base (ITB) involve selection of a segment number as well as an offset and are described in the following section of this chapter. The other three, Fault Tag 1, Fault Tag 2 and Fault Tag 3, do not produce effective addresses. These fault tags cause the processor to take a fault whenever they are used as address modifiers.

Fault Tag Modification

When bits 30-35 indicate one of the three fault tags (FT1, FT2, FT3) in the instruction word, the processor traps to the appropriate fault vector (see Chapter 7). The modifier field of an indirect word may also contain a fault tag. In this case, the processor stops the effective address formation and traps to the appropriate fault vector.

EFFECTIVE ADDRESS FORMATION INVOLVING BOTH SEGMENT NUMBER AND SEGEMENT OFFSET

The segment number can be changed for effective address formation using ITS or ITB modification, or using bit 29 of the instruction to specify the use of one of eight address base registers (ABR's) as part of the effective address.

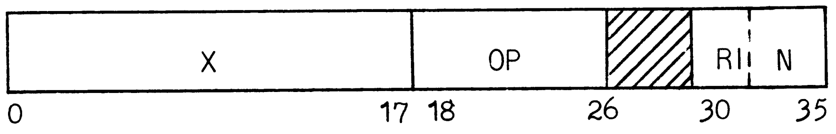
For the form of indirect addressing using ITS or ITB modification, a word pair is used to specify an intersegment reference. The segment to be referenced can be indicated in one of two ways. In ITS modification the segment number is in the first word of the word pair. In ITB modification the segment number is contained in an address base register (ABR) which is specified by the first word of the pair. Both ITS and ITB modification contain an offset in the second word of the word pair. They can specify further modification by means of the modifier field in the word pair using any one of the previously described modifications except for DU and DL. A special restriction requires that the first word of these word pairs occupy an even memory location.

Indirect To Segment (ITS)

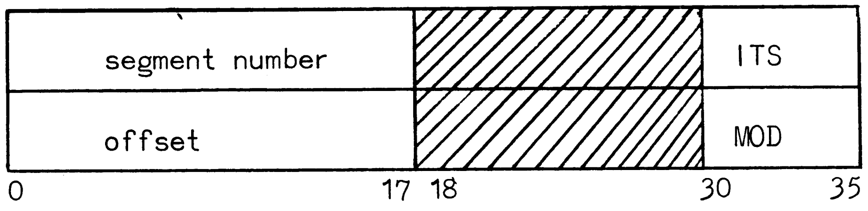
ITS modification must appear in the indirect word referenced via RI or IR modification (not in the instruction word). The offset in the referring word must be even.

Example:

Instruction Word



Even/Odd Indirect Word Pair at X and X + 1



1. The effective address referring to the indirect word indicates an even word and the next word.

2. The modifier field of the even indirect word is an ITS designator, i.e., bits 30-35 contain 100011.

3. The segment number field of the even word (bit positions 0-17) replaces the contents of the temporary base register (TBR).

4. The offset field of the odd word (bit positions 0-17) and any modification indicated in bit positions 30-35 of the odd word are used to compute a new offset which replaces the contents of the computed address register (CAR).

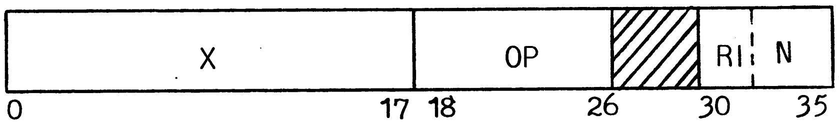
Indirect To Base (ITB)

The ITB modification must appear in the indirect word referenced via RI or IR modification (not in an instruction word). The offset in the referring word must be even.

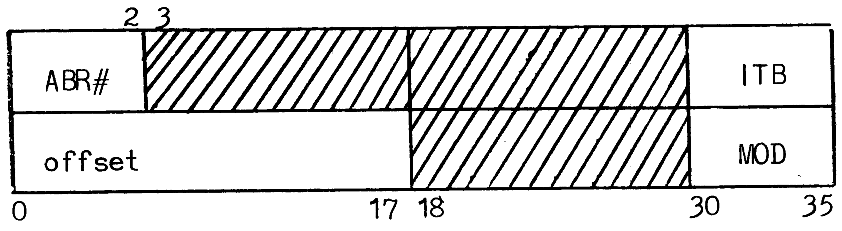
Indirect To Base (ITB), Continued

Example:

Instruction Word



Even/Odd Indirect Word Pair at X and X + 1



1. The effective address referring to the indirect word indicates an even word and the next word.

2. The offset field of the odd word (bit positions 0-17) and any modification indicated in bit positions 30-35 of the odd word are used to compute a new offset which replaces the contents of the CAR.

3. Bit positions 0-2 of the even indirect word select an address base register (ABR).

4. If bit position 21 of the selected ABR is 1, the ABR is external, and the contents of bit positions 0-17 are a segment number and are loaded into the TBR.

5. If bit position 21 of the selected ABR is 0, the ABR is internal. In this case, a pair of ABR's is used, one "internal", the other "external". The offset field (bit positions 0-17) of the selected ABR (internal) is added to the offset already contained in the CAR and the result replaces the contents of the CAR to form a new offset. Then bits 18-20 of the internal ABR are used to select another ABR (external). The segment number portion of the external ABR (bit positions 0-17) are loaded into the TBR to form a new tentative segment number.

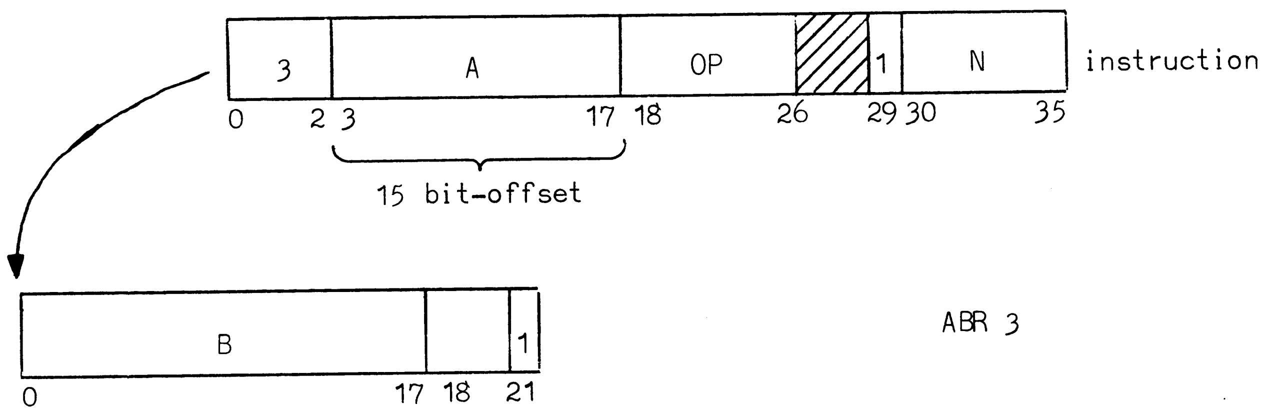
Use of Bit 29 to Specify an Address Base Register (ABR)

The final type of address modification involving both an offset and segment number uses bit 29 to indicate the use of one or a pair of eight **address base** registers (ABR's). For this type of modification bits 0-2 of the address field of the instruction specify which ABR to use.

If the designated ABR is external (i.e., bit number 21 = 1) the segment number field (bits 0-17) are loaded into the TBR. The remaining 15 bits of the instruction (bits 3-17) are used to produce an 18-bit offset. The offset is created by copying bits 3-17 of the instruction word into the rightmost 15 bits and extending bit position 3 to the left. This 18-bit offset, possibly subject to further modification, is loaded into the CAR.

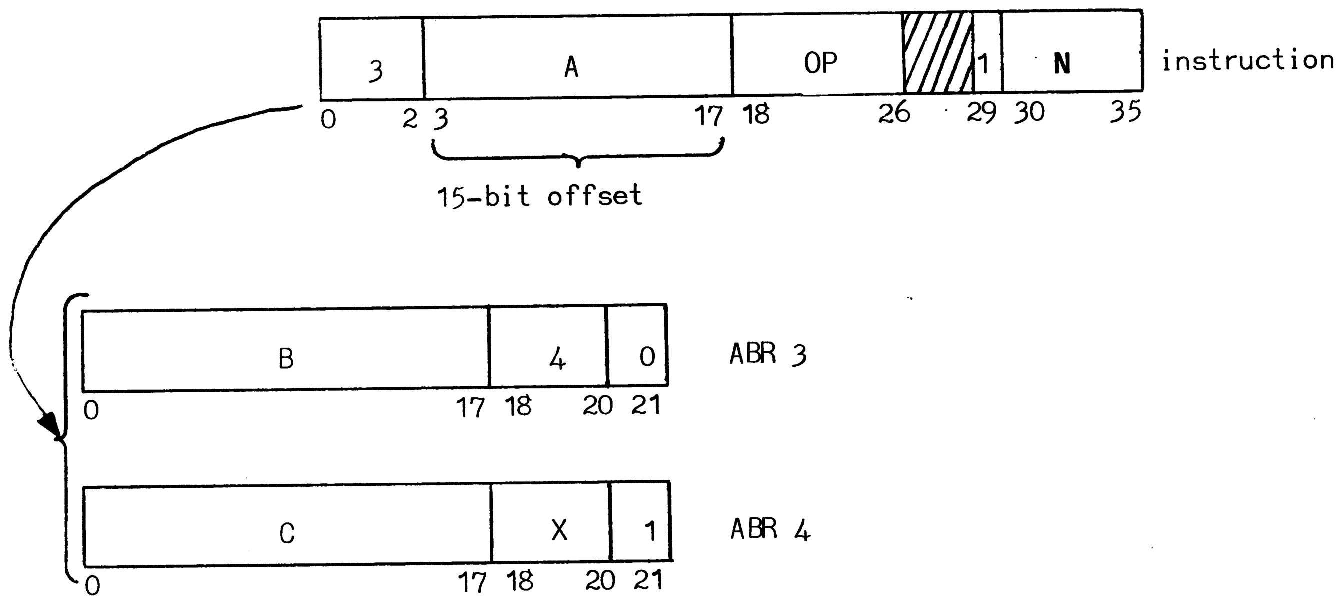
If the designated ABR is internal (i.e., bit number 21 =1) then bits 0-17 contain an offset. Bits 18-20 specify another ABR whose control field indicates it is external. Bit positions 0-17 of this external ABR contain a segment number which is loaded into the TBR. The offset from the instruction field, extended as above, is added to the offset of the internal ABR. This tentative offset, possibly subject to further modification, is loaded into the CAR.

The following examples illustrate the use of address appending, with bit 29 ON, both for paired and unpaired ABR's.



The operand at offset A in segment B is referenced.

Address Appending: Bit 29 ON, Unpaired ABR (External)



The operand at offset A + B in Segment C is referenced.

Address Appending: Bit 29 ON, Paired ABR
(Internal and External)

CHAPTER 7 FAULTS AND INTERRUPTS

Both faults and interrupts result in an interruption of normal processing, but there is a difference in how faults and interrupts originate. Generally, faults are caused by conditions which are internal to the processor; interrupts are caused by devices which are external to the processor. Faults and interrupts enable the 645 system to respond promptly when conditions occur that require system attention. A unique pair of locations which contain two instruction words is set aside to service each fault and interrupt condition. These instructions are in protected memory. The instruction words associated with faults are called the fault vector; there is a fault vector table for each processor port. The instruction words associated with interrupts are called the interrupt vector; similarly, there is an interrupt vector table for each processor port.

FAULTS

The 645 processor recognizes 32 types of faults. Many of the fault conditions are deliberately or inadvertently caused by the software and do not necessarily involve error conditions.

When a fault is detected, the processor forces the execution of the instructions in the fault vector. If it is desired to save machine conditions at the time of the fault, the first instruction of the fault vector should be SCU; otherwise, the machine conditions will not be saved. The second instruction of a fault vector should be a transfer to a routine to handle the fault. If the second instruction of the fault vector is not a transfer, execution will resume with the instruction immediately following that instruction which caused the fault.

The 32 faults are classified into six priority groups so that priorities can be established when two or more faults exist concurrently. The overlap of instruction execution and address preparation allows the simultaneous occurrence of faults. Group 1 has the highest priority and group 6 has the lowest. Only one fault within a group is allowed to be active at any one time. The fault which occurs first through the normal program sequence within a group is the one which is serviced. Only within group 6 are the other (nonserviced concurrent) faults saved by the hardware for eventual recognition. The faults not serviced and not saved can be handled only when and if they recur. There is one exception to the handling of the priority groups. When a group 3 fault (overflow or divide check) is caused by an operand having a parity error, the fault will be handled as a group 4 parity fault.

The table on this page and the description on the succeeding pages give details on the 32 faults.

Fault Code	Fault Name	Priority Group	Fault Category
11110	Startup	1	Manually generated
11011	Execute	1	Manually generated
11111	Trouble	2	Hardware generated
11101	Op Not Completed	2	Hardware or program generated
11010	Divide Check	3	Software generated
11001	Overflow	3	Software generated
01100	Parity	4	Hardware generated
01101	Illegal Memory Command	4	Hardware or program generated
11100	Lockup	4	Hardware or program generated
01011	Illegal Descriptor	5	Software generated
01010	Illegal Procedure	5	Software generated
01001	635 Compatibility	5	Software generated
11000	635/645 Compatibility	5	Software generated
00001	Master Mode Entry 1	5	Software generated
00100	Master Mode Entry 2	5	Software generated
00101	Master Mode Entry 3	5	Software generated
00111	Master Mode Entry 4	5	Software generated
00010	Derail	5	Software generated
01000	Fault Tag 1	5	Software generated
01110	Fault Tag 2	5	Software generated
01111	Fault Tag 3	5	Software generated
10000	Directed Fault 0	5	Software generated
10001	Directed Fault 1	5	Software generated
10010	Directed Fault 2	5	Software generated
10011	Directed Fault 3	5	Software generated
10100	Directed Fault 4	5	Software generated
10101	Directed Fault 5	5	Software generated
10110	Directed Fault 6	5	Software generated
10111	Directed Fault 7	5	Software generated
00110	Connect	6	Software generated
00011	Timer Runout	6	Software generated
00000	Shutdown	6	Manually generated

Directed Fault Descriptions

Priority 1 Faults (Startup and Execute)

Priority 1 faults cause the processor to initialize and enter the fault routine unconditionally. The faults cause processor operations to abort when the fault occurs, and the snapshot of the processor status is of no value. If both faults occur simultaneously, the startup fault has priority.

Startup

The fault is recognized when the POWER ON button is pressed. Power is turned on and the processor is initialized.

Execute

The fault is recognized when the EXECUTE pushbutton is depressed or an external frequency is substituted for the EXECUTE pushbutton (e.g., 'scope gate).

Priority 2 Faults (Trouble and Operation not Completed)

Priority 2 faults cause processor operations to abort when the fault is recognized. When the trouble fault occurs, the snapshot of the processor is the status of the original instruction which caused the fault. When the operation not completed fault occurs, the snapshot is of little value except for PBR and IC. If both faults occur simultaneously, the trouble fault has priority.

Trouble

The fault occurs during the execution of an execute double instruction (XED) forced by a fault or external interrupt. The fault also occurs during the execution of a store control unit instruction (SCU) which is the first instruction of an XED. The fault can be generated by the hardware; for example, it can be generated as a result of an operation not complete or by a parity error. The fault can also be generated by the operating system; for example, it can ge generated when a page containing the address generated by the SCU instruction is missing.

Operation Not Completed

This fault is caused by faulty communication between the processor and the system controller as described in the paragraphs that follow.

1. The processor addresses a system controller to which it is not attached.
2. A system controller fails to acknowledge the processor that addressed it.
3. The processor fails to generate a store access request or direct operand request within 1 to 2 milliseconds and is not in a DIS state (delay until interrupt signal).
4. The system controller closed out a double precision store or a read-alter-rewrite cycle (RAR) without receiving data from the processor.
5. The system controller fails to acknowledge the data from the processor; this can occur during a double-precision store or a read-alter-rewrite cycle (RAR).
6. Certain programming rules for the 645 were violated causing the processor to hang up. This can occur as the result of user misprogramming; for example, if DU or DL modifiers are used with store or RAR instructions.
7. Attempts to do a double precision indirect operation from an odd location or an attempt to execute a direct operation with a double precision op-code.

Priority 3 Faults (Divide Check and Overflow)

Priority 3 faults are detected during execution of the instruction causing the fault. The processor halts immediately when the fault is detected. The snapshot of the processor contains the contents of the processor base register (PBR) and the instruction counter (ICTC). These registers respectively contain the segment number and offset of the faulting instruction. Faults in priority 3 do not occur simultaneously. A parity error fault (priority 4) will have priority over a priority 3 fault occurring simultaneously if the parity error occurs in an operand.

Divide Check

The fault occurs when division cannot be carried out. The individual divide instructions list the cases in which division cannot be carried out.

Overflow

The fault occurs when there is arithmetic overflow, exponent overflow, or exponent underflow. If the overflow mask is set on, occurrence of overflow will cause the setting, testing, and storing of indicators but will not generate the fault while the mask is on. The overflow fault is not saved. If the mask is removed, the overflow fault will not be trapped by any instruction which set the overflow indicators while the mask was in effect. Pinpointing of the type of overflow condition that caused the fault is done by the routine servicing the fault.

Priority 4 Faults (Parity, Lockup, and Illegal Memory Command)

A priority 4 fault is detected after the address preparation cycle that generated it is completed. When a lockup or illegal memory command fault occurs, the snapshot of the processor contains the contents of the processor base register (PBR) and the instruction counter from which the segment number and offset of the faulting instruction can be determined. When a lockup fault occurs, the control unit constructs a complete snapshot of processor conditions at the time of the fault. Priority 4 faults cause the processor operation to halt after completion of operations already in execution; that is, the faults must wait for the system controller to acknowledge the last access request before the processor switches to the fault pair. Faults in priority 4 do not occur simultaneously.

Parity

The fault is generated when a parity error exists in a word which is read from a core location:

1. Single precision read: C(Y) is requested. C(Y,Y+1) is retrieved if Y is even; C(Y-1,Y) is retrieved if Y is odd. System controller will not report a parity error if it occurs in C(Y+1) or C(Y-1). The pair is restored with parity bits unchanged.

2. Double precision read: C(Y,Y+1) is requested and retrieved from core. System controller will report a parity error in either Y or Y+1 but does not indicate which word contains the error. The pair is restored with parity bits unchanged.

3. RAR: If a parity error exists in an indirect then tally (IT modification) word that is to be altered and rewritten or in an operand of a "to storage" instruction (e.g., ASA, ANSA, etc.), the alteration is carried out and the word is rewritten in store with a correctly computed parity bit.

If a parity error occurs on an instruction or an indirect word, the word is not used any further and the fault routine is entered. If the parity error occurs on an operand, the processor operation is completed with the faulty operand before the fault routine is entered.

If the parity mask is set on, occurrence of parity will cause the setting, testing, and storing of indicators but will not generate the fault while the mask is on or after the mask has been removed (i.e., the fault is not saved). If a parity error occurs on a segment descriptor word (SDW) or on a page table word (PTW), however, the fault routine is entered regardless of the state of the parity mask indicator.

Illegal Memory Command

The fault occurs when the processor issues a Connect (CIOC) to a channel that is masked off by a program or switch. For details of masking off channels see the description of the SMCM, SMIC and CIOC instructions in Chapter 2. The system controller does not execute the illegal memory command, but sends the applicable illegal action code to the processor. The two illegal action codes are: 100 if the processor is not the control processor and 011 for a store parity error. Another instance in which this fault occurs is when an RCCL instruction is issued to a memory that does not have a clock. Other situations that might cause the illegal memory command fault cause a 635 or 635/645 compatibility fault instead.

Lockup

The fault occurs when the processor is in a program lockup which inhibits recognition of an external interrupt or interrupt type fault for 1 or 2 milliseconds. An example is the continuous use of the inhibit bit in master mode. This is a recoverable fault, since the snapshot is valid. The one to two millisecond time refers to absolute elapsed time rather than processor execution time or memory cycle time. It includes any time during which the processor might have to wait for memory access (due to simultaneous request from the GIOC or associated store).

The lockup fault is deactivated during the time that a DIS (delay until interrupt signal) instruction is in effect and is deactivated also during the DIS state (no operation taking place) after the POWER ON pushbutton is pressed or after the STOP switch is activated during maintenance.

Priority 5 Faults

A priority 5 fault is detected during the address preparation cycle of an instruction. The faults must wait for the system controller to acknowledge the last access request before initiating the abort routine. Priority 5 faults do not occur simultaneously. Most of the faults are classified as priority 5.

Illegal Descriptor

This fault occurs when the class bits (33-35) of the page table word (PTW) or the segment descriptor word (SDW) contain an undefined configuration. There are five legal configurations; four configurations indicate modes of instruction execution and modes of address and the fifth configuration indicates a directed fault. (See the description of the SDW in Appendix E).

Illegal Procedure

The fault usually occurs when a programming violation is detected where one user may adversely affect another user or the operating system. The fault occurs under the following conditions:

1. Attempt to execute certain privileged instructions in Slave mode-- LDBR, SDBR, SAM, ZAM, LAM, CAM, SC, RCU and LACL.
2. Attempt to execute certain instructions in Slave mode when the address base register bit 22 is 0. These instructions are-- EAPn, EABn, TSBn, LDCF, ADBn, and LBRn.
3. Operation codes are detected that are not defined by the 645 instruction repertoire, including the all-zero operation code.
4. An effective address is outside of the segment boundary or a pointer is outside of the descriptor segment boundary.

5. The effective address is not to be accessed because of the access bit, write permit bit, or class conventions.

635 Compatibility

The fault occurs when an instruction is detected that is in the 635 repertoire but not in the 645 repertoire. These instructions are LBAR and SBAR. The operating system determines the course of action to be taken.

635/645 Compatibility

The fault occurs when an attempt is made to execute instructions that are privileged in both 635 and 645 repertoires. These instructions are SMIC, RMCM, SMCM, CIOC, LDT, DIS, and TSS. The last two are not privileged instructions in the 635 but are pointless in Slave mode. The operating system determines whether the fault was caused by a 635 or a 645 program and determines the course of action.

Master Mode Entry 1-4

These faults occur when one of the following instructions is encountered: MME, MME2, MME3, MME4. See the individual descriptions in Chapter 2 of instructions for more information.

Derail

The fault occurs when the DRL instruction is encountered. See the instruction descriptions for further information.

Fault Tags 1-3

These faults occur when a tally designator of FT1, FT2, or FT3 is encountered in an IT address modifier field. The indirect word is not obtained and the operation is not completed. Fault Tags 1, 2, and 3 are also recognized in an IT word brought in by IR modification cycle. Fault Tag 1 is reserved for 635 programs. Fault Tag 2 is the standard linkage fault when it occurs in a linkage section. Fault Tag 3 is available for 645 programmers.

Directed Faults 0-7

These faults occur when one of eight faults are encountered in bit positions 30-35 of the modifier field of the segment descriptor word or page table word. The fault number is indicated in bit positions 30-32 and zeros are in bit positions 33-35. The operating system determines what each of the eight faults shall be. It inserts the fault code into the segment descriptor word (SDW) and page table word (PTW) to prevent access to a segment or page at the current time. After the hardware builds the address of the related fault vector, the operating system services the fault by using the two words it has placed in the fault vector. In most cases these words will be an SCU instruction and a transfer instruction. Directed faults 0-7 are detected in SDW's and PTW's and can be used as the writer of the supervisor chooses.

Priority 6 Faults (Connect, Timer Runout, and Shutdown)

Faults in piority group 6 are recognized under conditions similar to those of program (external) interrupts. The processor checks for the three faults in this group at the same time it makes its periodic checks for an interrupt present signal for a program interrupt. The check is made at the beginning of address preparation cycles for an instruction word, indirect word, terminating indirect cycle, and indirect then tally word. The recognition of any one of these faults may be inhibited in Master mode by the interrupt inhibit bit (28) of the instruction word. The presence of group 6 faults is checked from the beginning of every address preparation cycle until the generation of the corresponding store location access request. Faults in this group have priority over program interrupts and cause the operations in the processor to abort conditionally upon completion of all pending operations. Faults must wait for the system controller to acknowledge the last access request before initiating the abort routine. If simultaneous faults occur in this group, priorities from highest to lowest are: shutdown, timer runout, and connect.

Connect

The fault occurs when the processor receives a connect signal from another active device through the system controller. This event should not be confused with the connect (CIOC) instruction encountered in a program sequence. Note that the connect fault is the only fault generated by a condition external to the processor, i.e., from another processor.

Timer Runout

The fault occurs when contents of the timer register reach zero. If the processor is in Master mode, recognition of the fault is delayed until the processor returns to Slave mode. This delay does not limit the counting of the timer register which continues to decrement after it rolls over to the maximum count of $2^{24}-1$.

Shutdown

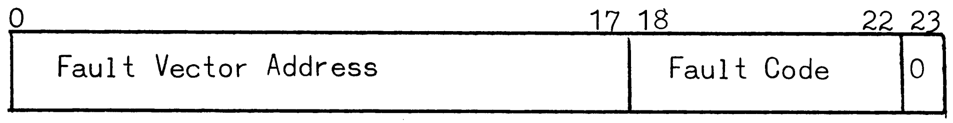
This fault can be originated in either of two ways. Power is turned off approximately one millisecond after the occurrence of either:

a. Depressing of the POWER OFF button on the main panel.

b. Receipt of a remote signal which indicates a commercial power failure.

Fault Vector Address

Each fault has a corresponding pair of instructions in its fault vector and is trapped to these by the hardware. The 32 vector pairs constitute the fault vector table which is a block of 64 words. The base address of the block, a 0 modulo(64) number, is set on the thumb wheels located on the back of the processor maintenence panel. The absolute 24-bit memory address of the fault vector pair is generated by extending the fault code by a trailing zero and appending this value to the value of the fault vector address. The generated address is that of the first word of the desired fault vector pair, as shown:



Address of Fault Vector Pair

Sequence of Fault Procedure

When a fault occurs, the control unit of the processor determines the time to initiate the fault procedure. This depends upon the type of fault. The fault procedure consists of the following steps:

1. Take a snapshot of processor status in anticipation of an SCU instruction and abort the processor.

2. At the end of the abort cycle, prepare the address of the fault vector pair.

3. Force the operation code of an XED instruction and the address of the fault vector pair into the instruction registers Y and C and set the interrupt inhibit bit (28) to 1.

   If the instruction causing the fault was not the result of a previous forced XED, the even or odd flag is set by the hardware to tell whether the faulting instruction was in an even or odd location. The flag is saved by the SCU instruction.

   If an attempt to execute the XED results in a parity error or a priority group fault, the processor switches to a trouble fault. The snapshot remains the same, but a new XED instruction is forced with an address corresponding to the trouble fault into the instruction registers.

4. Set a temporary absolute flag (not affecting the absolute indicator).

5. Cause Master mode of execution of the XED and vector pair instructions.

6. Issue a memory access request for the fault vector pair. The two instructions of the fault vector must adhere to the programming rules of a normal XED instruction.

7. Begin address preparation for the even instruction which is most likely to be an SCU instruction. If it is an SCU, bit 28 should be set to 1 by the operating system to avoid destroying the original snapshot.

   If a directed fault, illegal descriptor, illegal procedure (out-of-bounds or access violation fault) is encountered during any address preparation cycle forthe SCU operand or indirect word or if a parity error follows the fetch of an indirect word, the trouble fault routine is entered.

8. If the execution of the first instruction of the fault vector pair did not result in a transfer, the processor begins address preparation for the second instruction. The processor can now recognize external interrupts or faults and recovery. Therefore, use of bit 28 is optional. If either of the two instructions results in a transfer other than TSS, or a transfer with bit 29 = 1, or ITB or ITS indirection, the absolute indicator will be on. The mode of execution remains Master unconditionally for as long as the absolute indicator is on. If the **absolute** indicator is off, the mode of execution is set according to the current procedure.

9. Execute the second instruction.

10. If neither instruction caused a transfer, execute the instruction next in the normal sequence after the instruction which caused the fault directly or indirectly. This is the instruction in ICTC+1. The snapshot data stored by SCU is ignored.

    If the executed instruction caused a transfer, for example to a routine to service the fault, there must be an RCU instruction to restore the snapshot data stored by the SCU instruction. In most cases the original status is restored by an RCU instruction at the end of the service routine. After the status is restored, the original sequence is completed as though the interrupt never occurred. In some cases, as with the programmed Fault Tag 2, the restoration is achieved by an RCU pointing to modified SCU data.

Segment Address and Segment Number Generation

Faults in priorities 5 and 6 can have various effects on address preparation. Faults in these groups are detected during address preparation and the actual fetch from memory is inhibited. However, a segment address and a segment number are generated to define the word whose fetch is being attempted. The computed segment address and number become part of the safe-stored data.

In all other cases, the computed address field of the snapshot contains the address of the word causing the fault during address modification. Certain faults may be encountered during any phase of address preparation. These are:

directed faults

faults which occur when access rights are violated.

Some faults are trapped during the initial phase of address preparation. They are:

635 compatibility faults

635/645 compatibility faults

privileged instructions in Slave mode

Some faults are trapped during segment address preparation. These are:

Master mode entry faults

derail fault

illegal procedure fault (when an attempt is made to load an ABR with the lock bit= 1).

EXTERNAL (PROGRAM) INTERRUPTS

When a device, such as an I/O device, needs a service routine, a request is made to interrupt a current program and use a processor to service the device. The interrupt is signalled automatically, relieving the system of need to continuously test for events requiring attention.

Execute Interrupt Register

The system controller has a 32-bit execute interrupt register which receives interrupt requests from active devices. Each device is allocated specific cells (bit positions) in the register, according to its functional requirements. Cell 0 has the highest priority and cell 31 has the lowest. A device can set (to 1) any of the cells allocated to it by issuing an SMIC command to the system controller. A processor program can also set any of the cells in the execute interrupt register by issuing an SMIC instruction. However, the program cannot read or reset the cells to 0.

Interrupt Mask Register

The system controller has a 32-bit interrupt mask register which operates in conjunction with the execute interrupt register. The interrupt mask register can be set or reset by the "control" processor issuing an SMCM instruction in Master mode. (Designation of the "control" processor is made by an 8-position selector switch on the system controller panel.) Each 1-bit in the interrupt portion of the mask allows recognition of the information in the corresponding cell of the interrupt register. This 1-bit is said to unmask the interrupt cell. When one or more of the unmasked execute interrupt cells is set to 1, the system controller notifies the "control" processor of the interrupt. The "control" processor can read the mask register by executing an RMCM instruction.

Interrupt Vector

Each execute interrupt cell has an instruction pair associated with it in memory. The absolute address of the pair is a function of the cell number and the processor port number. These core locations constitute the interrupt vector and they are used in the same way as the fault vector locations already described.

The first of these locations normally contains an instruction which stores the processor status so that restoration can be made to the point of interrupt. The second location contains a transfer instruction to the service routine corresponding to the particular interrupt cell. The operating system places the instructions in the interrupt vector locations. The processor checks periodically for an interrupt present signal. The checking takes place in the beginning of every address preparation cycle other than for an instruction fetch. This occurs about once every 0.750 to 2.250 microseconds. The checking is inhibited in Master mode when the inhibit bit (bit 28 of the instruction) is on and also after the first step of the execution of a multiple-step instruction such as LREG and STB.

When one or more interrupt present signals are encountered and when operations already in progress are completed (and no fault is encountered), the processor issues an execute interrupt command. The command is issued in Master mode to the highest priority cell which is on. Because the processor can be designated "control" by more than one system controller, the highest priority is assigned to the controller connected to processor port A and the lowest to port H.

The system controller verifies that the execute interrupt command was issued by the "control" processor and was in Master mode. (If it was not, the controller responds with an illegal action code and the processor traps the illegal memory command fault. Since these are hardware rather than programmed functions, a hardware malfunction should be suspected. The processor would have a 635/645 compatibility fault; see pages 7-6 and 7-8. If the command is validated, the system controller responds by sending the number (5 bits) of the highest unmasked interrupt cell to the processor and resets the cell. The processor uses the cell number as part of the address of the interrupt vector. The address is used with a hardware-forced XED instruction to retrieve the instruction pair in memory. The routine which services the interrupt is now executed. Other interrupt cells remain set until priority permits their servicing.

Interrupt Priority

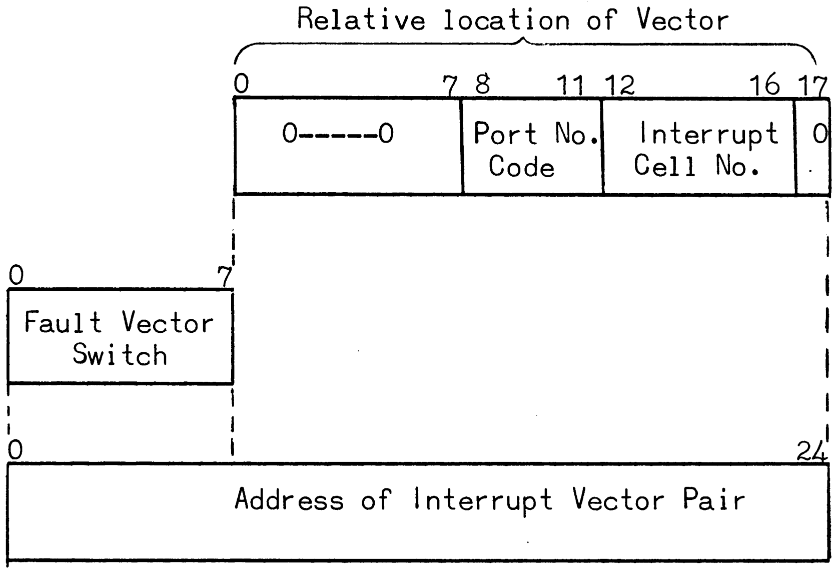
Interrupts have a priority equal to fault group 6. Within group 6, interrupts have the lowest priority. The checking for interrupts resumes when the first instruction having a 0 in bit position 28 is encountered. The programmer must insert 1's in bit 28 of all instructions which he desires to be interrupt-inhibited. Bit 28 has no effect in Slave mode. When the XED instruction is forced by the hardware to fetch the interrupt or fault pair, the effect of bit 28 being 1 does not extend to the XED's instructions. It does, however, extend to the subsequent indirect words and operand and address preparation (even if DU or DL).

Interrupt Vector Address

Selection of the interrupt vector address is similar to the selection of the fault vector address. There can be as many as eight interrupt vector tables for each processor--one for each processor port. Each table consists of the 32 vector pairs and constitutes a 64-word block. The interrupt tables for a processor immediately follow the fault vector tables for the same processor. The tables for each port are in the order of the port designation--from A to H.

All of the tables for a processor are relocatable (as a group) and can be placed anywhere in store by use of the fault vector thumb wheels. The switches provide the base address, 0 modulo(1024), of the fault table. The 18-bit address formed for a forced XED instruction points to the relative location of the vector pair. This relative address is formed by adding a 4-bit port number code (from

0001 to 1000) and a 5-bit interrupt cell number to the fault vector table.



Formation of Interrupt Vector Pair Address

Use of Interrupt Inhibit

Bit position 28 of the instruction word can be used in Master mode to inhibit the recognition of the three faults of priority group 6. When bit 28 is set to 1, interrupts are inhibited until either Slave mode is entered or until a subsequent instruction having 0 in bit position 28 is encountered.

APPENDIX A - INSTRUCTIONS LISTED BY OCTAL CODE

001	MME
002	DRL
004	MME2
005	MME3
007	MME4
011	NOP
015	CIOC
020	ADLX0
021	ADLX1
022	ADLX2
023	ADLX3
024	ADLX4
025	ADLX5
026	ADLX6
027	ADLX7
033	ADL
035	ADLA
036	ADLQ
037	ADLAQ
040	ASX0
041	ASX1
042	ASX2
043	ASX3
044	ASX4
045	ASX5
046	ASX6
047	ASX7
050	ADB0
051	ADB1
052	ADB2
053	ADB3
054	AOS
055	ASA
056	ASQ
060	ADX0
061	ADX1
062	ADX2
063	ADX3
064	ADX4
065	ADX5
066	ADX6
067	ADX7
071	AWCA
072	AWCQ
073	LREG
075	ADA
076	ADQ
077	ADAQ
100	CMPX0
101	CMPX1
102	CMPX2
103	CMPX3
104	CMPX4
105	CMPX5
106	CMPX6
107	CMPX7
111	CWL
115	CMPA
116	CMPQ
117	CMPAQ
120	SBLX0
121	SBLX1
122	SBLX2
123	SBLX3
124	SBLX4
125	SBLX5
126	SBLX6
127	SBLX7
135	SBLA
136	SBLQ
137	SBLAQ
140	SSX0
141	SSX1
142	SSX2
143	SSX3
144	SSX4
145	SSX5
146	SSX6
147	SSX7
150	ADB4
151	ADB5
152	ADB6
153	ADB7
154	SDBR
155	SSA
156	SSQ
157	ZAM
160	SBX0
161	SBX1
162	SBX2
163	SBX3
164	SBX4
165	SBX5
166	SBX6
167	SBX7
171	SWCA
172	SWCQ
173	LDB
175	SBA
176	SBQ
177	SBAQ
200	CNAX0
201	CNAX1
202	CNAX2
203	CNAX3
204	CNAX4
205	CNAX5
206	CNAX6
207	CNAX7
211	CMK
215	CNAA
216	CNAQ
217	CNAAQ
220	LDX0
221	LDX1
222	LDX2
223	LDX3
224	LDX4
225	LDX5
226	LDX6
227	LDX7
231	RSW
232	LDBR
233	RMCM
234	SZN
235	LDA
236	LDQ
237	LDAQ
240	ORSX0
241	ORSX1
242	ORSX2
243	ORSX3
244	ORSX4
245	ORSX5
246	ORSX6
247	ORSX7
250	STP0
251	STP1
252	STP2
253	STP3
254	STB
255	ORSA
256	ORSQ
257	LAM
260	ORX0
261	ORX1
262	ORX2
263	ORX3
264	ORX4
265	ORX5
266	ORX6
267	ORX7
270	TSB0
271	TSB1
272	TSB2
273	TSB3
275	ORA
276	ORQ
277	ORAQ
300	CANX0
301	CANX1
302	CANX2
303	CANX3
304	CANX4
305	CANX5
306	CANX6
307	CANX7
310	EAB0
311	EAB1
312	EAB2
313	EAB3
315	CANA
316	CANQ
317	CANAQ
320	LCX0
321	LCX1
322	LCX2
323	LCX3
324	LCX4
325	LCX5
326	LCX6
327	LCX7
330	EAB4
331	EAB5
332	EAB6
333	EAB7
335	LCA
336	LCQ
337	LCAQ
340	ANSX0
341	ANSX1
342	ANSX2
343	ANSX3
344	ANSX4
345	ANSX5
346	ANSX6
347	ANSX7
350	EAP0
351	EAP1
352	EAP2
353	EAP3
354	STAC
355	ANSA
356	ANSQ
357	STCD
360	ANX0
361	ANX1
362	ANX2
363	ANX3
364	ANX4
365	ANX5
366	ANX6
367	ANX7
370	EAP4
371	EAP5
372	EAP6
373	EAP7
375	ANA
376	ANQ
377	ANAQ
401	MPF
402	MPY
405	CMG
411	LDE
415	ADE
421	UFM
423	DUFM
425	FCMG
427	DFCMG
430	FSZN
431	FLD
433	DFLD
435	UFA
437	DUFA
440	SXL0
441	SXL1
442	SXL2
443	SXL3
444	SXL4
445	SXL5
446	SXL6
447	SXL7
450	STZ
451	SMIC
453	LACL
454	STT
455	FST
456	STE
457	DFST
461	FMP
463	DFMP
470	FSTR
475	FAD
477	DFAD
500	RPL
505	BCD
506	DIV
507	DVF
512	LDCF
513	FNEG
515	FCMP
517	DFCMP
520	RPT
525	FDI
527	DFDI

531	NEG
532	CAM
533	NEGL
535	UFS
537	DUFS
540	SBR0
541	SBR1
542	SBR2
543	SBR3
544	SBR4
545	SBR5
546	SBR6
547	SBR7
551	STBA
552	STBQ
553	SMCM
554	STC1
557	SAM
560	RPD
565	FDV
567	DFDV
573	FNO
575	FSB
577	DFSB
600	TZE
601	TNZ
602	TNC
603	TRC
604	TMI
605	TPL
607	TTF
610	RTCD
613	RCU
614	TEO
615	TEU
616	DIS
617	TOV
620	EAX0
621	EAX1
622	EAX2
623	EAX3
624	EAX4
625	EAX5
626	EAX6
627	EAX7
630	RET
633	RCCL
634	LDI
635	EAA
636	EAQ
637	LDT
640	ERSX0
641	ERSX1
642	ERSX2
643	ERSX3
644	ERSX4
645	ERSX5
646	ERSX6
647	ERSX7
650	STP4
651	STP5
652	STP6
653	STP7
655	ERSA
656	ERSQ
657	SCU
660	ERX0
661	ERX1
662	ERX2
663	ERX3
664	ERX4
665	ERX5
666	ERX6
667	ERX7
670	TSB4
671	TSB5
672	TSB6
673	TSB7
675	ERA
676	ERQ
677	ERAQ
700	TSX0
701	TSX1
702	TSX2
703	TSX3
704	TSX4
705	TSX5
706	TSX6
707	TSX7
710	TRA
715	TSS
716	XEC
717	XED
720	LXL0
721	LXL1
722	LXL2
723	LXL3
724	LXL4
725	LXL5
726	LXL6
727	LXL7
731	ARS
732	QRS
733	LRS
735	ALS
736	QLS
737	LLS
740	STX0
741	STX1
742	STX2
743	STX3
744	STX4
745	STX5
746	STX6
747	STX7
750	STC2
751	STCA
752	STCQ
753	SREG
754	STI
755	STA
756	STQ
757	STAQ
760	LBR0
761	LBR1
762	LBR2
763	LBR3
764	LBR4
765	LBR5
766	LBR6
767	LBR7
771	ARL
772	QRL
773	LRL
774	GTB
775	ALR
776	QLR
777	LLR

APPENDIX B INSTRUCTION TIMING

This appendix describes the instruction execution times for the 645 processor. The actual time for each instruction is also included in a table at the end.

BASIS FOR CALCULATION OF THE LISTED EXECUTION TIMES

The listed execution times are the average times for a pair of instructions and are determined from the conditions listed below which are considered the general case.

The pair is preceded and followed by instructions of the same type. The addresses are such that the memory cycles for the preceding instruction fetch and the memory cycles for the operands of the pair are overlapped. The address modification is register modification (for example R = Xn, AU, etc., but not DU, DL).

All necessary descriptor words for the appending operation are available in the associative memory. In the case of store instructions, the "written bit" of the page table word does not have to be set.

The average execution time for an instruction pair is defined as the time interval between the start of address preparation for the even instruction of the pair, and the start of address preparation for the even instruction of the next pair:

Average Execution Time = (Execution Time of Pair) divided by (2)

There are five exceptions to the definition of average. The listed execution times reflect these five exceptions. The exceptions are:

1. Short load type instructions including LDA, LDAQ, ADA, ADAQ. Any instruction which requires an operand to be loaded from core, and for which the required operations unit execution time is less than 1.35 microseconds. When the memory cycle for the preceding instruction fetch, and the memory cycles for the operands of the pair are not overlapped, the execution time for a short load type is 1.5 microseconds.

2. Store types such as STA, STAQ, FST, DFST, AOS, ASA etc. Any instruction which alters the contents of a core location is a store type instruction and the listed execution time specified reflects the following conditions:

   a. The store type instruction is preceded by a short load type instruction (for example, LDA - STA; LDAQ - STAQ; LDA - ASA).

   b. Items 2, 3, 4 and 5 of the <u>general case</u> apply.

   The listed execution time is calculated as follows:

   Store Type Execution Time = (Execution Time of Pair) - (Execution Time of Load Type)

3. Long load types (for example, FAD, FMP, MPS) - Any instruction for which the required operations unit execution time is greater than 1.35 microseconds is a long load type instruction. For a string of these instructions, the address preparation time and instruction fetch time are less than the operations unit time. The listed execution time is the time interval between the start of the operations unit operation for one long load type and the next long load type instruction.

4. Control types (for example, TRA, TNC, XEC) - Any instruction that fetches another instruction to be executed is a control type instruction. The listed execution time is the time interval between the start of address preparation for the control type instruction and the start of address preparation for the next instruction to be executed.

5. Base types (for example, LBR_n, SBR_n, EAP_n, etc.) - Any instruction which changes or stores a base register (ABR_{0-7}), the descriptor segment base register or the associative registers is a base type instruction. The listed execution time is the time interval between the start of address preparation of the next base type plus one-half of the instruction address preparation time.

ADDRESS MODIFICATION AND APPENDING OPERATION TIMES

The instruction execution time should be increased by the following factors for address modification and appending operations:

1.	MOD R (R = Xn, AU, etc.):	add nothing
2.	MOD R (R = DU, DL) :	add nothing
3.	MOD IR, RI :	add 2 microseconds for each indirection; add 2.3 microseconds for each ITS, ITB indirection.
4.	MOD IT :	add 2 microseconds if the contents of the indirect word are not changed, add 2.7 microseconds if the contents of the indirect word are changed.
5.	Appending Word Fetch :	add 1.6 microseconds for a DSPTW, SDW, or PTW fetch.

INSTRUCTION SEQUENCE TIMES

When an instruction sequence contains one or more instructions whose timing factor is an exception to the average execution time, these additional times must be added to the calculated average execution time of the sequence. Cases which need additional time factors are:

1. A transfer to or from a long load type instruction.

2. An instruction at location n (n even) modifies an instruction at n + 1, n + 2, n + 3, or an instruction at n (n odd) modifies an instruction at location n + 1, n + 2.

3. An instruction at location n (even or odd) modifies a register needed for the address modification of an instruction at location n + 1, n + 2.

4. Entry into a fault routine.

Alphabetical List of Instructions with Timing and Page Numbers

MNEMONIC	PAGE	TIMING IN usec
ADA	2-29	1.67
ADAQ	2-29	1.87
ADB$_{0-7}$	2-24	3.31
ADE	2-51	1.67
ADL	2-34	1.67
ADLA	2-31	1.67
ADLAQ	2-32	1.87
ADLQ	2-32	1.67
ADLX$_{0-7}$	2-33	1.67
ADQ	2-29	1.67
ADX$_{0-7}$	2-30	1.67
ALR	2-28	1.47
ALS	2-26	1.47
ANA	2-64	1.67
ANAQ	2-64	1.87
ANQ	2-64	1.67
ANSA	2-65	3.73
ANSQ	2-65	3.73
ANSX$_{0-7}$	2-66	3.73
AOS	2-35	3.73
ARL	2-27	1.47
ARS	2-25	1.47
ASA	2-30	3.74
ASQ	2-31	3.74
ASX$_{0-7}$	2-31	3.73
AWCA	2-33	1.67
AWCQ	2-33	1.67
BCD	2-89	3.71
CAM	2-89	1.99
CANA	2-78	1.67
CANAQ	2-78	1.87
CANQ	2-78	1.67
CANX$_{0-7}$	2-79	1.67
CIOC	2-90	1.35
CMG	2-76	1.67
CMK	2-77	1.68
CMPA	2-72	1.67
CMPAQ	2-74	1.87
CMPQ	2-73	1.67
CMPX$_{0-7}$	2-75	1.67
CNAA	2-80	1.67
CNAAQ	2-80	1.87
CNAQ	2-80	1.67
CNAX$_{0-7}$	2-80	1.67
CWL	2-75	1.68
DFAF	2-51	2.10
DFCMG	2-62	1.87
DFCMP	2-62	1.87
DFDI	2-59	23.15
DFDV	2-58	23.56
DFLD	2-47	1.87
DFMP	2-55	11.85
DFSB	2-52	2.54
DFST	2-48	2.61
DIS	2-90	varying
DIV	2-44	14.12
DRL	2-91	2.0
DUFA	2-51	2.10
DUFM	2-55	11.61
DUFS	2-53	2.54
DVF	2-45	14.12
EAA	2-5	1.51
EAB$_{0-7}$	2-19	1.99
EAP$_{0-7}$	2-19	2.41
EAQ	2-5	1.51
EAX$_{0-7}$	2-6	1.51
ERA	2-69	1.67
ERAQ	2-69	1.87
ERQ	2-69	1.67
ERSA	2-70	3.73
ERSQ	2-71	3.73
ERSX$_{0-7}$	2-71	3.73
ERX$_{0-7}$	2-70	1.67
FAD	2-50	1.99
FCMG	2-61	1.79
FCMP	2-61	1.79
FDI	2-57	14.12
FDV	2-56	14.12
FLD	2-47	1.67
FMP	2-54	6.04
FNEG	2-60	1.47
FNO	2-60	1.47
FSB	2-52	2.54
FST	2-48	2.08
FSTR	2-48	2.84
FSZN	2-63	1.67
GTB	2-89	9.87
LACL	2-110	2.1
LAM	2-111	29.87 abs. 33.33 app.
LBR$_{0-7}$	2-20	2.92
LCA	2-10	1.67
LCAQ	2-11	1.87
LCQ	2-11	1.67
LCX$_{0-7}$	2-11	1.67
LDA	2-7	1.67
LDAQ	2-7	1.87
LDB	2-20	6.88
LDBR	2-21	2.92
LDCF	2-21	3.73
LDE	2-47	1.67
LDI	2-9	1.67
LDQ	2-7	1.67
LDT	2-8	1.67
LDX$_{0-7}$	2-8	1.67
LLR	2-28	1.47
LLS	2-26	1.47
LREG	2-10	6.29
LRL	2-27	1.47
LRS	2-25	1.47
LXL$_{0-7}$	2-8	1.67
MME$_{1-4}$	2-92	2.0
MPF	2-43	7.09
MPY	2-43	7.09
NEG	2-46	1.51
NEGL	2-46	1.51
NOP	2-109	1.47
ORA	2-67	1.67
ORAQ	2-67	1.87
ORQ	2-67	1.67
ORSA	2-68	3.73
ORSQ	2-68	3.73
ORSX$_{0-7}$	2-68	3.73
ORX$_{0-7}$	2-68	1.67
QLR	2-28	1.47
QLS	2-26	1.47
QRL	2-27	1.47
QRS	2-25	1.47
RCCL	2-110	1.7
RCU	2-118	4.6
RET	2-84	4.0
RMCM	2-97	1.35
RPD	2-103	1.35
RPL	2-106	1.35

(Revised September 14, 1970)

MNEMONIC	PAGE	TIMING IN usec
RPT	2-100	1.35
RSW	2-109	1.35
RTCD	2-84	3.7
SAM	2-112	29.87 abs. 33.33 app.
SBA	2-36	1.67
SBAQ	2-37	1.87
SBLA	2-39	1.67
SBLAQ	2-40	1.87
SBLQ	2-39	1.67
SBLX$_{0-7}$	2-40	1.87
SBQ	2-36	1.67
SBR$_{0-7}$	2-22	2.19
SBX$_{0-7}$	2-37	1.67
SCU	2-113	6.44
SDBR	2-22	2.19
SMCM	2-98	3.97
SMIC	2-99	2.0
SREG	2-14	8.59
SSA	2-38	3.73
SSQ	2-38	3.74
SSX$_{0-7}$	2-38	3.74
STA	2-13	2.08
STAC	2-13	3.74
STAQ	2-13	2.61
STB	2-22	8.25
STBA	2-16	2.08
STBQ	2-16	2.08
STC1	2-18	2.47
STC2	2-18	2.47
STCA	2-15	2.08
STCD	2-12	2.84
STCQ	2-15	2.08
STE	2-49	2.08
STI	2-17	2.47
STP$_{0-7}$	2-23	2.84
STQ	2-13	2.08
STT	2-17	2.41
STX$_{0-7}$	2-14	2.08
STZ	2-18	2.39
SWCA	2-41	1.67
SWCQ	2-42	1.67
SXL$_{0-7}$	2-14	2.08
SZN	2-76	1.67
TEO	2-87	2.0
TEU	2-87	2.0
TMI	2-85	2.0

MNEMONIC	PAGE	TIMING IN usec
TNC	2-86	2.0
TNZ	2-85	2.0
TOV	2-87	2.0
TPL	2-86	2.0
TRA	2-81	2.0
TRC	2-86	2.0
TSB$_{0-7}$	2-82	2.0
TSS	2-83	2.0
TSX$_{0-7}$	2-83	2.0
TTF	2-88	2.0
TZE	2-85	2.0
UFA	2-50	1.99
UFM	2-54	5.80
UFS	2-52	2.54
XEC	2-96	2.0
XED	2-96	2.0
ZAM	2-112	2.84 abs. 6.30 app.

(Revised September 14, 1971)

APPENDIX C -- ASCII CHARACTER SET

	First Two Digits of Octal Representation of the Character	Last Digit of Octal Representation of Character							
		0	1	2	3	4	5	6	7
Control Characters	00	(NUL)							BEL
	01	BS	HT	NL	VT	NP		RRS	BRS
	02			HLF		HLR			
	03								
Language Characters	04	space	!	"	#	$	%	&	'
	05	(	)	*	+	,	-	.	/
	06	0	1	2	3	4	5	6	7
	07	8	9	:	;	<	=	>	?
	10	@	A	B	C	D	E	F	G
	11	H	I	J	K	L	M	N	O
	12	P	Q	R	S	T	U	V	W
	13	X	Y	Z	[	\	]	^	_
	14	`	a	b	c	d	e	f	g
	15	h	i	j	k	l	m	n	o
	16	p	q	r	s	t	u	v	w
	17	x	y	z	{	¦	}	~	PAD

Multics Implementation of ASCII Character Set

Control Character Definitions

BEL	-	Sounds an audible alarm.
BRS	-	Black ribbon shift. Character code 017 (ASCII- SI) is used for this function.
BS	-	Move carriage back one column. (Implies overstriking, not erasing).
HLF	-	Half-line forward feed. Character code 022 (ASCII - DC2) is used for this function.
HLR	-	Half-line reverse feed. Character code 024 (ASCII - DC4) is used for this function.
HT	-	Horizontal tabulate. Move carriage to next horizontal tab stop. (On variable tab machines, the default tab settings are at columns 11, 21, 31, etc.)
NL	-	New line. Move carriage to left edge of next line. Character code 012 (ASCII - LF) is used for this function.

NP - New page. Character code 014 (ASCII- FF) is used for this function.

NUL - This character is treated exactly as PAD. It is used in an "edited" output mode by the operating system. In normal output mode it is considered a "not-used" character, and printed with an octal escape sequence. This character cannot appear in a canonical character string.

PAD - Padding character. Character code 177 (ASCII- DEL) is used for this function. This character cannot appear in a canonical character string.

RRS - Red ribbon shift. Character code 016 (ASCII-SO) is used for this function.

VT - Vertical tabulate. (On variable tab machines, the default tab settings are at lines 11, 21, 31, etc.) This character cannot appear in a canonical character string.

The following ASCII control characters are not used:

SOH	001	STX	002	ETX	003	EOT	004	ENQ	005	ACK	006
CR	015	DLE	020	DC1	021	DC3	023	NAK	025	SYN	026
ETB	027	CAN	030	EM	031	SUB	032	ESC	033	FS	034
GS	035	RS	036	US	037						

APPENDIX D - SCU/RCU SUMMARY

The SCU instruction is used to store data needed for restoring a processor to the precise point of interruption after a fault or interrupt has been serviced. The RCU instruction is used to restore the processor status. SCU and RCU allow an instruction to be interrupted in mid-execution.

A total of three double-words is used by the SCU/RCU instructions. See next page. The SCU instruction actually stores 203 bits of information. The remaining 13 bits are not used; they are indicated by X on the next page. The RCU instruction restores only 168 bits. (The 35 bits used by the SCU but ignored by the RCU are indicated by * on the next page.)

To facilitate definitions and descriptions, the SCU/RCU data is grouped into the following functional classes:

1. Control Unit Status
   a. Cycle flags: PL, PA, PZ, PT, PN
   b. Repeat modes flags: RF, FT, FD, FL
   c. Execute Double mode flags: XDO, XDE
   d. Segmentation flags: $ESTR_{0-3}$, $OSTR_{0-3}$, ITS, ITB
   e. Mode of Execution flags: M/S, MASF, ABS (included in IR)
   f. Control tag register: CT_{0-5}
   g. Auxiliary flags: IC, TRZ

2. Computed address.

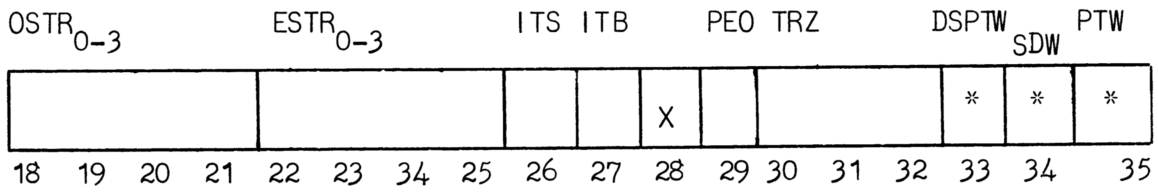
3. Registers - TBR, PBR, IR, ICTC, IE, IO

4. Software - assisting status
   a. Appending subcycles: DSPTW, SDW, PTW
   b. Fault code
   c. Processor number
   d. Illegal procedure fault type: a-e
   e. Auxiliary indicators: EA, PEO

WORD Y

$C(Y)_{0-17} \Longleftrightarrow C(TBR)_{0-17}$

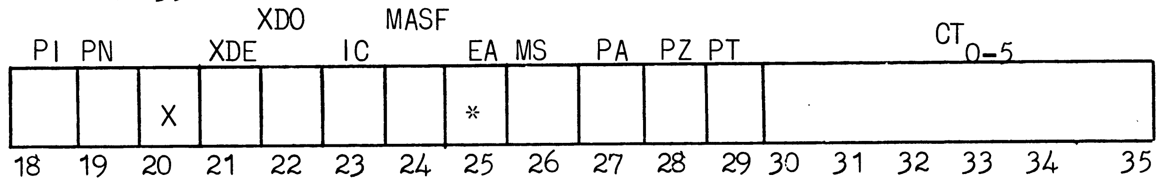
$C(Y)_{18-35} \Longleftrightarrow$ Appending Unit Status



WORD Y 1

$C(Y\ 1)_{0-17} \Longleftrightarrow$ Computed Address (*)

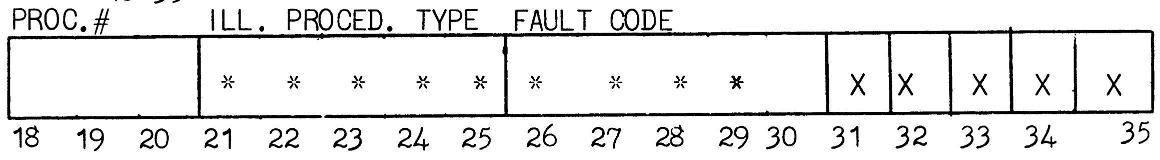
$C(Y\ 1)_{18-35} \Longleftrightarrow$ Control Unit Status



WORD Y 2

$C(Y\ 2)_{0-17} \Longleftrightarrow C(PBR)_{0-17}$

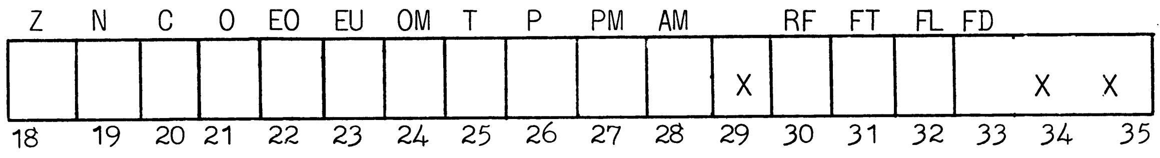
$C(Y\ 2)_{18-35} \Longleftrightarrow$ Fault Data (*)



WORD Y 3

$C(Y\ 3)_{0-17} \Longleftrightarrow C(ICTC)_{0-17}$

$C(Y\ 3)_{18-35} \Longleftrightarrow C(IR)_{0-10}$ and $C(Y\ 3)_{30-35} \Longleftrightarrow$ Control Unit Status



WORDS Y 4 Y 5

$C(Y\ 4)_{0-35} \Longleftrightarrow$ Even Instruction (IE)

$C(Y\ 5)_{0-35} \Longleftrightarrow$ Odd Instruction (IO)

Format of the SCU Data

CONTROL UNIT AND APPENDING UNIT STATUS

Cycle Flags

PI - This flag defines the cycle during which an address derived from the instruction counter (ICTC) is prepared for instruction fetch. This does not include instruction fetch due to transfer, return, execute or execute double.

PA - This flag defines the cycle during which an address (or a direct operand) is prepared as specified by the original address and tag fields of an instruction. If it is a direct instruction (e.g., NEG, CAM, RPT shifts) or a direct operand (DU or DL) then no memory access request will be made. If it is not direct but the modifier is R, a request will be made for an indirect word and address modifications will proceed when the latter is received.

PZ - This flag defines the cycle during which an address is prepared as specified by the address and tag fields of an indirect word and the immediately preceding tag was IR or RI. If the new tag is R or IT (other than fault tag), the preparation is for an operand. If the new tag is RI or IR, a request will be made for a further indirect word.

PT - This flag defines the cycle during which an address is prepared as specified by the address and tag (only for DIC or IDC) fields of an indirect and tally word (and the tag of the immediately preceding IT tag). The alteration of the tally word (for tags other than I and CI) took place previously. In cases other than DIC and IDC the preparation is for an operand. In the case of DIC and IDC, further indirection is possible depending on the new tag.

PN - This flag defines the cycle during which address modification tags are ignored and the indirect address becomes the effective address. This cycle is employed in two cases. One is in the Return mode (RET and RTCD) after the new ICTC has been received and the address for the instruction fetch is to be prepared. The other case is the operand from repeated instruction (under RPT and RPD) with RI modifier. In this case only R=N is allowed in the indirect word.

The cycle flags field may be viewed as a one out of five code. One and only one bit must be set to 1. The one exception is interrupts and restorations to the DIS state which may be made with all flags reset, or with PA alone set.

Repeat Modes Flags

RF - This flag defines the period extending to the end of the first address preparation cycle of the repeated instruction in the Repeat or Repeat Link mode, or to the end of the first address preparation cycle of the odd repeated instruction in the Repeat Double mode. Address preparation during the first execution of a repeated instruction follows a different formula than during subsequent executions.

FT - This flag indicates that the processor is in the Repeat mode. It is set following a RPT instruction and reset when a prespecified terminate condition is met, or the abort sequence is entered due to a fault or an external interrupt.

FD - This flag indicates that the processor is in the Repeat Double mode. It is set following a RPD instruction and reset when a prespecified terminate condition is met, or the abort sequence is entered due to a fault or an external interrupt.

FL - This flag indicates that the processor is in the Repeat Link mode. It is set following a RPL instruction and reset when a prespecified terminate condition is met, or the abort sequence is entered due to a fault or an external interrupt.

The RF flag may be set to 1 only if one of the Repeat modes is in effect. No more than one Repeat mode may be set to 1 at any given time.

Execute Double Mode Flags

XDE - This flag is set following a XED instruction encountered in an even location. It remains set until one of the XED'ed instructions causes a transfer of control, or all XED'ed instructions have been executed and the odd instruction, which follows the XED in program sequence, is about to be executed.

XDO - This flag is set following a XED instruction encountered in an odd location. It remains set until one of the XED'ed instructions causes a transfer of control, or all XED'ed instructions have been executed and the even instruction, which follows the XED in program sequence, is about to be executed.

XDE and XDO are interlocked so that no more than one may be on at a time, even though one XED may bring in another XED as the odd member of the pair. XDE or XDO will indicate the location of the original XED.

Segmentation Flags

During PI cycles, segmentation is carried out according to PBR if the absolute indicator is off. No segmentation takes place during PI if the absolute indicator is on. However, during PA or PZ or PT or PN cycles, the following flags define how segmentation is carried out:

ITS - This flag is set to 1 when an ITS modifier is encountered and remains set until an ITB modifier is encountered subsequently, or address modification is completed. When ITS is set, segmentation is carried out according to TBR. The following flags are significant only if ITS is reset.

ITB - This flag is set to 1 when an ITB modifier is encountered and remains set for the duration of the PZ cycle corresponding to the indirect word adjacent to the ITB word. This flag allows an internal base, if one is specified, to be added to the indirect address.

$ESTR_{0-2}$ The even segment tag register contains a pointer to one of eight ABR's (address base registers). The pointer is loaded from the three most significant bits of the original address of an even instruction, or from an ITB word encountered during address modification for an even instruction.

$ESTR_3$ This bit is set to 1 if bit 29 of the original even instruction was 1, or if an ITB modifier has been encountered during address modification for an even instruction.

$OSTR_{0-2}$ Similiar to $ESTR_{0-2}$ for an odd instruction.

$OSTR_3$ Similiar to $ESTR_3$ for an odd instruction.

The ITS and ITB flags shoud not be set to 1 concurrently.

The Mode of Execution Flags

M/S - The Master/Slave flag represents the class of the most recent active instruction. When in Absolute or temporary Absolute, it is arbitrarily set to Master (1). (Absolute mode addressing is Master mode execution by definition).

MASF - This flag defines the temporary Absolute mode, which begins when the processor forces the XED for the fault vector or issues the execute interrupt command to the memory controller, and ends when the Execute Double mode terminates.

Control Tag Register

CT_{0-5} - The control tag register is used to store IT or IR modifiers during address modifications. Once an IT or IR modifier has been encountered, a definite pattern is established for further indirection. Since the tag of the incoming indirect word replaces the previous tag in the tag field of the instruction register, it is necessary to hold IT or IR modifiers in a special register. If CT_0=0, then an IT or IR modifier has been encountered and the modifier in the tag field of the instruction register (R or RI) controls address preparation.

Auxiliary Flags

IC - This flag denotes whether current address modifications are for the even member (0) or the odd member (1) of the instruction pair. Also, the least significant bit of the instruction counter (i.e., $ICTC_{17}$) indicates whether the active instruction is odd (1) or even (0). Normally $ICTC_{17}$ and IC represent the same state, however, in the Execute Double mode $ICTC_{17}$ corresponds to the location of the original XED instruction whereas IC will correspond to the XED'ed instructions.

In the case of faults other than groups 5 and 6, the IC flag has no meaning; however, ICTC will point to the active instruction for group 3 and 4 faults. If this happens in the Execute Double mode, IC will point to the faulting XED'ed instruction.

TRZ - When certain IT words are altered, the tally may reach zero. This should be reflected by the Tally Runout indicator when the execution of the instruction is completed. During the remainder of the address modification process, a flag is set to 1 to indicate that the new tally was zero (RS_{0-11}=0).

COMPUTED ADDRESS

This is the address calculated according to the rules imposed by the particular type of cycle and/or the history of address modifiers before any relocation takes place.

During normal PI cycles ($PI \cdot \overline{XDE}$) the address is computed according to the following formula:

$ICT + 1$ if $ICTC_{17} = 1$ or

$ICT + 2$ if $ICTC_{17} = 0$

During PI cycles intended to refetch an odd instruction due to XED in the even location (PI • XDE) the formula is:

$ICTC + 1$

The following terms will be defined in order to present the formulae for address computation for operand or indirect word fetch:

Y	The most recent address field of the instruction or indirect word held by IE_{0-17} or IO_{0-17}
Δ_1	The delta specified by certain IT words.
Δ_2	The delta specified by RPT or RPD.
A	Bit 8 of RPD (a binary coefficient).
B	Bit 9 or RPD (a binary coefficient).
X	$X_1 - X_7$
D	A binary coefficient, which is equal to 0 if the register designator is N, DU, DL. The binary coefficient is equal to 1 for other designators.
TAG	The most recent tag of the instruction or indirect word held by IE_{30-35} or IO_{30-35}.

The address remains unchanged (i.e., equal to Y) in the following cases:

$PT \mid (PA+PZ)\&(TAG=IR) \mid PN \mid PN\&(FL\&\overline{RF})$

In the case of PT certain computations may have taken place in the preceding PR cycle. In particular,

$Y - \Delta_1$	if	$CT_{2-5} = SD$
		or
$Y - 1$	if	$CT_{2-5} = DI \mid DIC$
		or
		$CT_{2-5} = SCR$ and old $C_f = 0$

The address is computed according to

$y \mid D\&C(R)$ when $(PA \mid PZ)\&(TAG{=}R \mid RI) \mid PZ\&(CT_{0-1} = IR)\ (TAG{=}R \mid IT)$

The address is computed according to

$y \mid C(X)$ when $PA\ (FT \mid FD \mid FL)\&RF$

The address is computed according to

$C(X) \mid \Delta_2$ when $PA\&FT\&\overline{RF}$

The address is computed according to

$C(X) \mid A\&\Delta_2$ when $PA\&\overline{IC}\&FD\&\overline{RF}$

The address is computed according to

$C(X) \mid B\&\Delta_2$ when $PA\&IC\&FD\&\overline{RF}$

Also, during any PA if bit 29 of the instruction word is 1, or during PZ if ITB = 1, if the selected ABR is designated as internal its address field will be added to the relative address.

The computed address stored by a SCU for a group 5 or 6 fault will be in accordance with one of the above formulae. In the case of a Fault Tag 1-3 encountered in RI or IR indirect word, however, it will be the address computed in the previous cycle (i.e., the relative address of the word containing the Fault Tag). Also, in the case of a parity error encountered in an

RI, IR, or IT indirect word, the address is that of the word containing the parity error.

REGISTERS

Temporary Base Register (TBR)

During any address preparation cycle the effective pointer is placed in TBR. If ITS = 0, then TBR contains either PBR or one of the eight ABR's. If ITS = 1, then TBR contains the pointer which was brought in by the ITS pair. The TBR captured by a SCU for a group 1-4 fault is not necessarily the one involved in the fault.

During the cycle restored by the RCU, the effective pointer is generated again and placed in TBR. Therefore, if ITS = 0, then TBR is superfluous in the RCU.

The Procedure Base Register (PBR)

The PBR captured by the SCU defines the active procedure which generated the fault, except for groups 1 and 2.

In general, the RCU must contain a valid PBR.

The Indicator Register (IR)

$C(Y+3)_{25}$ will contain the state of the Tally Runout indicator prior to address modification of the SCU instruction (for Tally operations). Following the RCU the Tally Runout indicator will reflect $C(Y+3)_{25}$ regardless of what Tally is generated by address modifications of the RCU instruction.

The indicators stored by a SCU following a group 5 or 6 fault represent the proper state. However, certain indicators may have changed due to an operation executed with an operand having a parity error.

Only the Absolute indicator is initialized (to the off state). Other indicators may assume either state in a SCU following a group 1 fault.

The Instruction Counter (ICTC)

The ICTC stored by a SCU following a fault in PI represents the last successfully executed instruction. In general, for a group 3 to 6 fault, the ICTC points to the active instruction. In the Execute Double mode, the original XED is defined as the active instruction. In the Repeat modes, the current repeated instruction is defined as the active instruction.

The ICTC is not initialized; consequently, a group 1 fault may cause any random number to appear in ICTC. The ICTC stored due to a Lockup and Op Not Complete may be 0, 1, or 2 greater than that of the faulting instruction.

Even and Odd Instructions

The instruction pair stored by the SCU is irrelevant for faults which occur during the PI address preparation cycle. For other faults in group 4 to 6 the active instruction is one of the pair as determined by $ICTC_{17}$, or by IC in the Execute Double mode. The address field (0-17) contains the original address of the most recent indirect address. The Op Code field and control bits (18-29) contain the original values. The tag field (30-35) contains the original tag or the most recent indirect tag except during the repeat modes, where the original tag is maintained.

For group 3 faults, the instruction pair may be irrelevant in the normal mode. However, in Execute Double mode the active pair is captured by the SCU. In normal mode PBR and ICT may be used to refetch the faulting instruction.

CODES

Illegal Procedure Type - A one out of five code identifies the type of violation causing the illegal procedure fault. The violations are: a) privileged instruction in slave (PIF); b) locked base in slave (LBF); c) illegal op codes (IOC); d) out of bounds (OOB); e) access violations (AVF). The last two may occur concurrently.

DSPTW, SDW, PTW - These flags give the location of a directed fault or an illegal descriptor as the descriptor segment page table word, or the page table word. Also, DSPTW or SDW indicates an out of bounds pointer, and PTW or none of the three indicate an out of bounds relative address. These bits are also valid during parity faults.

Fault Code - The fault code is included for use by the Trouble Fault recovery routine. It identifies the fault vector, which the processor was attempting to execute when trouble occurred.

Processor Number - The processor number identifies each processor in a multi-processor system for purposes of Test and Diagnostic routines and machine checks due to hardware-generated faults.

Auxiliary Indicators - The EA flag indicates whether a group 5 or 6 fault was encountered during address preparation for an operand (1) or an indirect word (0). The PEO flag indicates a parity error in the operand.

Hardware Flags - Allowable Codes

The following table is a summary of the codes allowed to represent the various groups of hardware status.

Flags	Allowable Codes
PI-PA-PZ-PT-PN	1 out of 5
XDE-XDO	00, 01, 10
IC	0, 1
FT-FD-FL	000, 1 out of 3
RF	0, 1 with FT|FD|FL=1
ITS-ITB	00, 01, 10
M/S	0, 1
MASF	0, 1 with forced XED
$OSTR_{0-3}$	any
$ESTR_{0-3}$	any
ITR	0, 1
CT_{0-5}	any except 100010 (undefined IT)

VALIDITY OF SCU DATA

The following is a summary of the validity of SCU data for various fault situations.

The effective pointer ("TBR") and the relative address ("Computed Address") are valid for faults in groups 5 and 6. The "Software-assisting Status" is valid for specific group 5 faults, except the fault code and processor number, which are always valid. The validity of other data is tabulated below.

	Hardware Status	PBR	ICT	IR	IE/IO
Group 1	a	0	0	a	DIS/DIS
Group 2	0	1	b	0	0
Group 3	c	1	1	1	c
Group 4	c	1	1	d	c
Group 5	1	1	1	1	1
Group 6 or Interrupts	1	1	1	1	1

1 - Valid

0 - Invalid or questionable

a - Some initialized, some don't care

b - ICTC + 0, 1, 2

c - If XDE|XDO|FL|FT|FD = 1, then IC, RF and IE, IO are valid. Otherwise, invalid.

d - May be invalid due to operand parity error

SCU MODIFICATIONS

Fault Tag 2

The following discussion illustrates the modifications to the SCU data, required prior to the RCU, in the case of Fault tag 2 employed in the linking process.

Fault Tag 2 may be encountered in any one of four types of words. The exact type may be determined from the SCU data, stored in Y through Y+5, as follows:

1. The instruction word. In this case $PA=C(Y+1)_{27} = 1$.

2. An indirect word under control of IT with DIC|IDC modifier. In this case $PT=C(Y+1)_{29} = 1$.

3. An odd indirect word under control if IR|RI modifier. In this case $PZ=C(Y+1)_{28} = 1$ and $C(Y+1)_{17} = 1$.

4. An even indirect word under control of IR|RI modifier. In this case, $PZ=C(Y+1)_{28} = 1$ and $C(Y+1)_{17} = 0$.

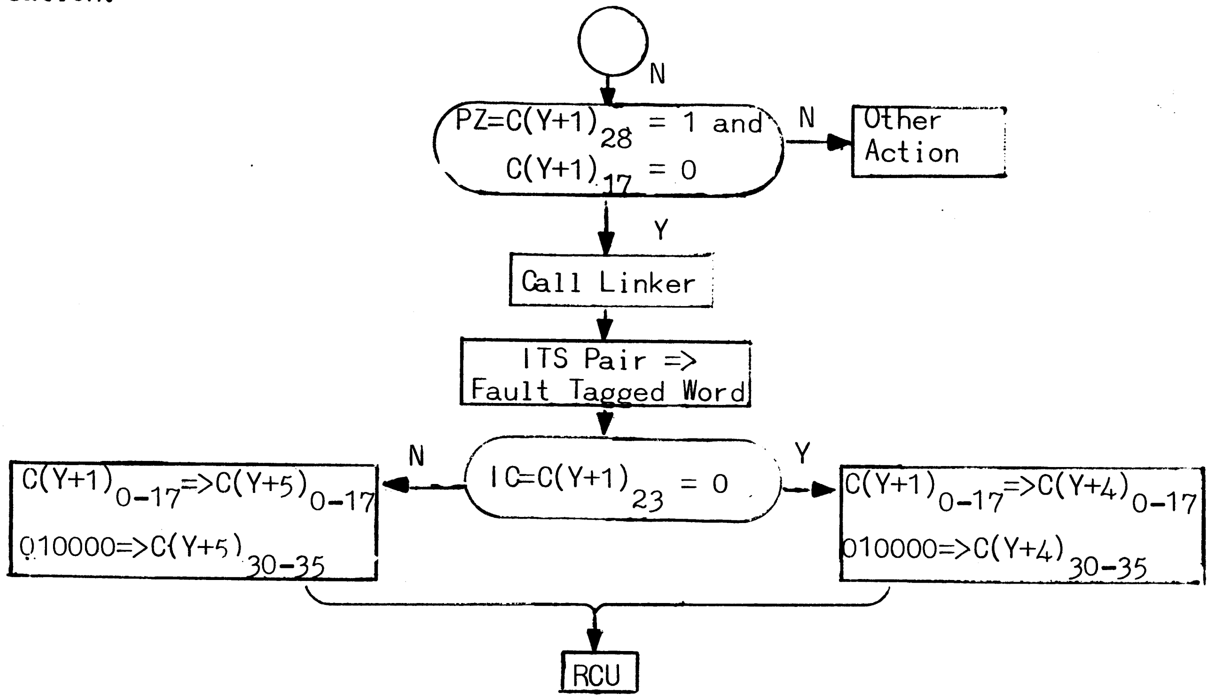
The actions taken for cases 1 to 3 will not be discussed here. It is assumed that linking is desired for case 4; that implementation is described below.

The linker can determine the segment and address of the word containing the Fault tag from $TBR=C(Y)_{0-17}$, and the computed address = $C(Y+1)_{0-17}$, respectively. The address field of this word contains a pointer to the call name. The linker will find the segment number for the call name and use it to form an ITS pair. The ITS pair will be stored in the location of the Fault Tagged word and the adjacent word.

Alternately, the pointer to the call name may be obtained from the address field of the active instruction in the SCU data. If $IC=C(Y+1)_{23}=0$, the active instruction is the even one, and the pointer is $C(Y+4)_{0-17}$. If $IC=C(Y+1)_{23}=1$, the active instruction is the odd one, and the pointer is $C(Y+6)_{0-17}$.

Next, the SCU data must be modified so when used by the RCU, the processor will refetch the word pair now containing the ITS. Only the address and tag field of the active instruction need be altered. The address of the Fault Tagged word, i.e., the computed address = $C(Y+1)_{0-17}$, must be stored in the address field of the active instruction, which is in $(Y+4)_{0-17}$ if $IC=C(Y+1)_{23}=0$, or in $(Y+5)_{0-17}$ if $IC=C(Y+1)_{23}=1$. The tag of the active instruction, which is stored in $(Y+4)_{30-35}$ if IC=0, or $(Y+5)_{30-35}$ if IC=1, must be RI, N for which the code is 010000.

Since the Fault Tagged indirect word was replaced by an ITS pair, no trap will occur on successive executions. The following illustrates an encounter with Fault tag 2 in an even indirect word under control of IR or RI modification.



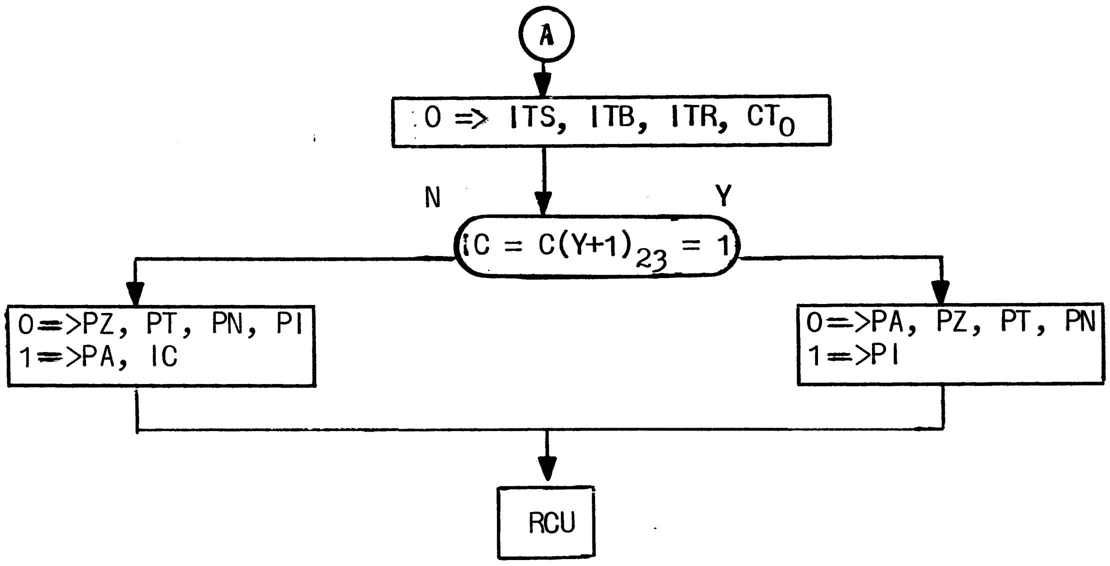
Skip to the Next Instruction - Group 5 Faults

In certain cases of group 5 faults, it is necessary to modify the SCU data so that the RCU, which follows the fault-initiated routine, resumes execution following the faulting instruction. This action is likely to take place in the case of MME's and DRL, and perhaps in the case of certain Fault Tags.

It is assumed that if the fault occurred in the normal mode due to an instruction specified by ICTC, the return will be to the instruction specified by ICTC+1. If the fault was encountered in an even XED'ed instruction, the return will be made to the odd XED'ed instruction. If the fault was encountered in the odd XED'ed instruction return will be to the instruction which follows the XED. Due to programming restrictions these faults should not occur in the Repeat modes. This can be tested by ascertaining that FD|FT|FL = 0 in the SCU data.

The technique for modifying the SCU data turns out to be rather simple. If the faulting instruction is even, i.e., IC = $C(Y+1)_{23}$ = 0, then zeros must be inserted in the positions corresponding to PZ, PT, PN, PI (bits 28, 29, 19, 18 of Y+1); and ones must be inserted in the bit positions corresponding to IC, PA (bits 23, 27). If the faulting instruction is odd, i.e., IC=1, then zeros must be placed in the bit positions corresponding to PA, PZ, PT, PN; and a one must be placed in the bit positions corresponding to PI. In either case, zeros must be placed in the bit position corresponding to ITS, ITB (bits 26, 27 of Y) and CT_0 (bit 30 of Y+1).

If a tally operation occurred during address modifications for the faulting instructions, and the instruction is considered complete when the fault is recognized (e.g., MME's, DRL), the Tally Runout indicator will reflect the new state. However, if the faulting instruction is not considered complete when the fault is recognized (e.g., Fault Tag), the state of the tally is saved by the ITR bit, $C(Y)_{30}$ and is not reflected by the indicator. If the instruction is resumed, the hardware will transfer the state of ITR to the indicator. However, if the process is returned to the instruction following the faulting instruction, the ITR bit should be reset to zero by the software.



Illegal Memory Command

In general, this fault is interpreted as an illegal request by the processor for action of the system controller. If the validity of the request can be determined within the processor, then the trap is handled via other faults (i.e., 635 and 635/645 compatibility). However, one violation is detected by the system controller, namely, when a processor (in the Master mode) has issued a connect to a channel that is masked off (by program or switch). In this case, the connect is not sent to the designated channel, and the processor is notified of the illegal action. The clock request to memory that doesn't have a clock results in an illegal memory command. These are the violations which will cause the illegal memory command fault in the 645.

The location of an invalid CIOC instruction is given by ICTC and PBR of the SCU data.

It is desirable for the fault handling routine to have the location of the CIOC operand. This word (considered as the connect word by the GIOC, or the PCW by the Drum) was used by the system controller to determine the designated channel (the decode of bits 33-35). At the present time, the hardware does not save the effective address and the effective pointer, corresponding to this word. Therefore, the burden of tracing through the CIOC instruction is on the fault handling routine.

Memory Parity Error

Additional information is provided with the SCU data of parity faults to help identify the type of word containing the parity error and its location.

If the parity error occurred in a DSPTW, SDW, or PTW a one would be placed in $(Y)_{33-35}$ of the SCU data as follows:

DSPTW --- $(Y)_{33}$
SDW --- $(Y)_{34}$
PTW --- $(Y)_{35}$

If the parity error occurred in the operand, then a one is placed in $(Y)_{29}$ (PEO). It may be quite difficult to determine the location of the operand since the SCU does not contain the effective address and pointer in this case. Furthermore, the faulty operand had already been used in the operation, which might have altered a register used in its address preparation. No simple solution is offered for this problem. The problem is also complicated in the cases of XEC and XED, since these operations involve more than one operand each.

If the appending words and the operand were eliminated as parity candidates, i.e., $(Y)_{29} = (Y)_{33-35} = 0$ in the SCU, then the error is in an indirect word, or in the instruction word, whose location is defined by ICTC and PBR. If ICT is even, the parity error could be in either the even or the odd instruction word, or both. If PT = 1 in the SCU, the parity error occurred in an IT type indirect word whose location is given by the computed address and the TBR of the SCU (if the word were of the RAR type, the parity error may no longer exist).

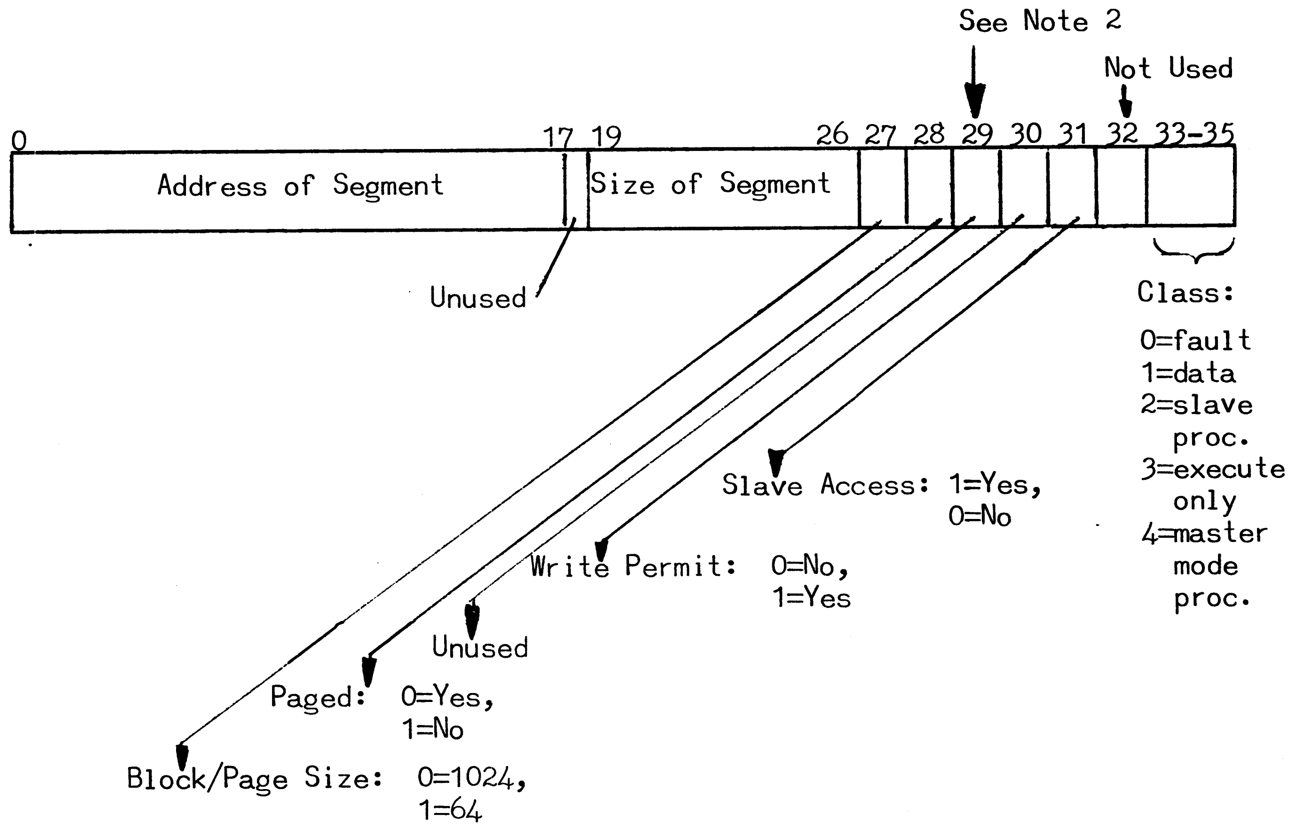
If PZ = 1 in the SCU data, the parity error occurred in an IR | RI type indirect word. The address of the indirect word containing the parity error is given by the computed address of the SCU data. If this address is odd, then the pointer (i.e., segment number) is given by TBR of the SCU data. However, if the address is even, then the pointer is not available in the SCU data if the parity error occurred in an ITS or an ITB pair. Consequently, if ITS | ITB = 1, in the SCU data, the fault handling routine must trace through the instruction to derive the effective pointer. If PN = 1 and FT | FD = 1 in the SCU data, the parity error occurred in an RI type indirect word in a repeated instruction. If PN = 1 and FT FD = 0, then the parity error is in the word containing ICTC, etc., during the execution of RET or RTCD.

APPENDIX E FORMAT OF WORDS USED IN ADDRESS APPENDING

This appendix contains descriptions of the segment descriptor word (SDW), the page table word (PTW), and the descriptor segment page table word (DSPTW).

FORMAT OF SEGMENT DESCRIPTOR WORD (SDW)

Each SDW has an address field pointing to the origin of a segment or its page table. The SDW also has size and control information about the segment which it describes. The SDW has the format shown below.



Format of Segment Descriptor Word

Notes: 1. If bit 28 is 1 in the SDW, the address points to the segment: otherwise, the address points to the page table.

2. Bits 30-32 are treated as a special field. If there is a directed fault, they contain the fault code. If there is no fault, they contain information as shown in the illustration.

<u>Bits 0-17, Address</u> - The address field contains the high-order 18 bits of a 24-bit address. It is the absolute address of either the origin of a segment being accessed or the segment page table if the segment being accessed is paged. This address is 0 modulo(64) for 64-word blocks of unpaged segments, 0 modulo(1024) for 1024-word blocks of unpaged segments, and 0 modulo(64) for page tables.

<u>Bit 18</u> - Not used.

<u>Bits 19-26, Size</u> - These bits contain the number of pages in the segment if paged; or the number of blocks if not paged.

<u>Bit 27, Descriptor</u> - Indicates block or page size:

0 = 1024

1 = 64

<u>Bit 28</u> - Indicates paging:

0 = paged

1 = non-paged

<u>Bit 29</u> - Not used.

<u>Bit 30</u> - Indicates permission to write when in Slave mode:

0 = may <u>not</u> be written into in Slave mode

1 = may be written into in Slave mode

(see bits 33-35)

<u>Bit 31</u> - Indicates permission to access the segment of page table:

0 = access may be made <u>only</u> in Master mode

1 = access may be made in either Slave or Master (see bits 33-35)

<u>Bit 32</u> - Used only when bit positions 33-35 indicate a directed fault.

<u>Bits 33-35, Class bits</u> - These are called class bits because they classify th the type of segment.

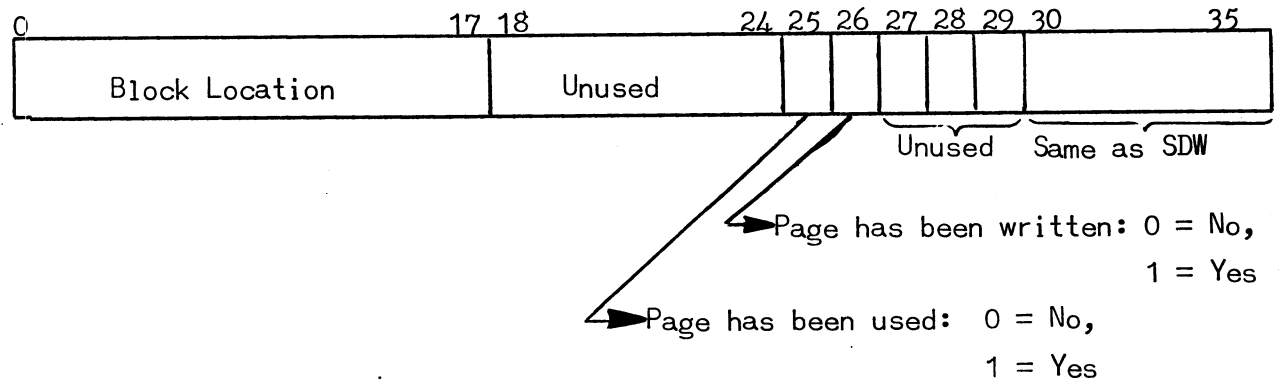
Bits 33-35 of the SDW are interpreted as follows:

30	31	32	33	34	35	Meaning
X	X	X	0	0	0	Directed Fault xxx; the code which is in bit positions 30-32
			0	0	1	Data Segment Only
			0	1	0	Slave procedure
			0	1	1	Execute-Only
			1	0	0	Master Procedure
			1	0	1	These codes constitute an illegal descriptor
			1	1	0	
			1	1	1	

If bit positions 33-35 contain a fault code, the number of the fault code is contained in positions 30-32.

FORMAT OF PAGE TABLE WORD.

Each page table word (PTW) contains an address field which points to the origin of the block in core memory to which the corresponding page of the segment is assigned. Each word also contains a control field. The word has the following format:

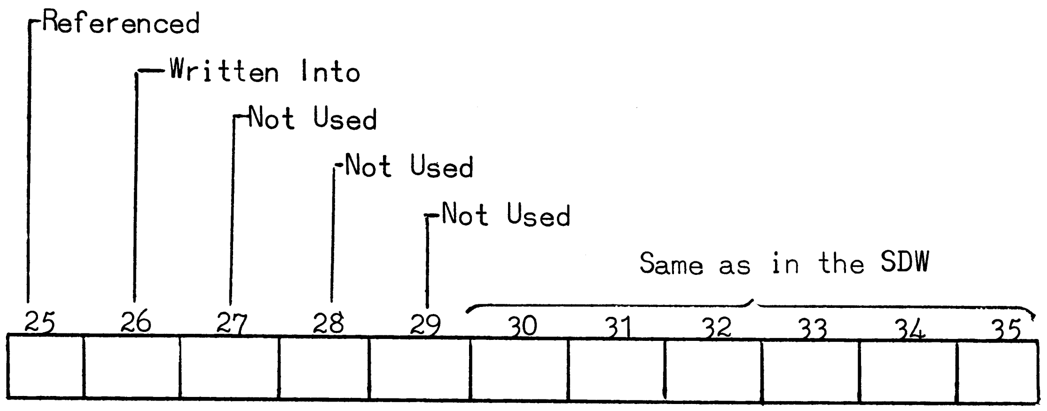

0
17 18
24 25 26 27 28 29 30
35
Block Location
Unused
Unused
Same as SDW
Page has been written: 0 = No, 1 = Yes
Page has been used: 0 = No, 1 = Yes

Page Table Word

Bits 0-17, Address - The address field contains the high-order 18 bits of a 24-bit address. This address is 0 modulo(64) for 64-word pages or 0 modulo(1024) for 1024-word pages.

Bits 18-24 - Not used.

Bits 25-35, Control Bits - In the control field, bits 30-35 have the same meanings as they do in the SDW, except that they apply to the page. Bit positions 25 and 26 are used to indicate page use. The control bit positions for the PTW are summarized in the following illustration.



Bit 25 - Indicates whether the page has been referenced (used):

0 = not yet referenced

1 = has been referenced

Bit 26 - Indicates whether the page has been written into (modified):

0 = not yet written into

1 = has been written into

Bits 27-29 - Not used

Bits 30-35 - These bit positions have the same meaning as those in the segment descriptor word already described.

FORMAT OF DESCRIPTOR SEGMENT PAGE TABLE WORD (DSPTW)

The format of a DSPTW is the same as that for a PTW. The processor ignores the contents of bits 25 and 26 of a DSPTW.

APPENDIX F SEGMENT AND PAGE ACCESS RIGHTS

The 645 processor possesses a set of access rules which are applied to each word referenced in a segment. The settings of control bits in the SDW and PTW are used to determine the access rights. For each reference, access is determined by the value of the control bits for both the referenced segment and the segment being executed. The following table lists the possible control bit settings and summarizes the corresponding access rights.

Current Procedure Mode	Control Bits in SDW or PTW of Segment Being Accessed			Access Permitted			
				Non-Transfers		Transfers	
						Same Segment	Different Segment
	Class (Bits 33-35)	Access (Bit 31)	Write (Bit 30)	Read	Write	TBR=PBR	TBR≠PBR
Master	Data Segment (001)	0/1	0/1	Yes	Yes	No	No
Master	Procedure Segment Slave(010) Execute(011) Master(100)	0/1	0/1	Yes	Yes	Yes	Yes
Slave	Data Segment (001)	0	0/1	No	No	No	No
		1	0	Yes	No	No	No
		1	1	Yes	Yes	No	No
	Procedure Slave(010)	0	0/1	No	No	No	No
		1	0	Yes	No	Yes	Yes
		1	1	Yes	Yes	Yes	Yes
	Procedure Execute Only (011)	0	0/1	No	No	No	No
		1	0	①	No	Yes	②
		1	1	①	①	Yes	②
	Procedure Master (100)	0	0/1	No	No	No	No
		1	0/1	No	No	No	②

NOTES: ① If TBR = PBR (i.e., if the segment being referenced is the segment being executed, then yes; otherwise, no.

② If effective address = 0, then yes; otherwise, no.

When referencing an unpaged segment, the processor generates a fault if the reference is incompatible with the access indicated by the class bits of the SDW for that segment. If a segment is paged, the processor logically combines the class bits of the SDW and the class bits of the PTW. Then, if the references is incompatible with the most restrictive access produced by combining the two sets of access rights, the processor generates a fault and the segment is not accessed.

INDEX (EXCLUDING INSTRUCTIONS)

INDEX (EXCLUDING INSTRUCTIONS), Cont'd.

INDEX (EXCLUDING INSTRUCTIONS), Cont'd.

INDEX (EXCLUDING INSTRUCTIONS), Cont'd.

INDEX TO INSTRUCTIONS

ADA	ADD TO A	2-30
ADAQ	ADD TO AQ	2-31
ADBn	ADD TO ABRn	2-21
ADE	ADD TO EXPONENT REGISTER	2-56
ADL	ADD LOW TO AQ	2-36
ADLA	ADD LOGIC TO A	2-33
ADLAQ	ADD LOGIC TO AQ	2-34
ADLQ	ADD LOGIC TO Q	2-34
ADLXn	ADD LOGIC TO Xn	2-35
ADQ	ADD TO Q	2-30
ADXn	ADD TO Xn	2-31
ALR	A LEFT ROTATE	2-26
ALS	A LEFT SHIFT	2-29
ANA	AND TO A	2-72
ANAQ	AND TO AQ	2-72
ANQ	AND TO Q	2-72
ANSA	AND TO STORAGE A	2-73
ANSQ	AND TO STORAGE Q	2-73
ANSXn	AND TO STORAGE Xn	2-74
ANXn	AND TO Xn	2-73
AOS	ADD ONE TO STORAGE	2-37
ARL	A RIGHT LOGIC	2-25
ARS	A RIGHT SHIFT	2-27
ASA	ADD STORED TO A	2-32
ASQ	ADD STORED TO Q	2-32
ASXn	ADD STORED TO Xn	2-33
AWCA	ADD WITH CARRY TO A	2-35
AWCQ	ADD WITH CARRY TO Q	2-36
BCD	BINARY TO BCD	2-103
CAM	CLEAR ASSOCIATIVE MEMORY	2-105
CANA	COMPARATIVE AND WITH A	2-88
CANAQ	COMPARATIVE AND WITH AQ	2-89
CANQ	COMPARATIVE AND WITH Q	2-88
CANXn	COMPARATIVE AND WITH Xn	2-89
CIOC	CONNECT I/O CHANNEL	2-106
CMG	COMPARE MAGNITUDE	2-86
CMK	COMPARE MASKED	2-87
CMPA	COMPARE WITH A	2-81
CMPAQ	COMPARE WITH AQ	2-83
CMPQ	COMPARE WITH Q	2-82
CMPXn	COMPARE WITH Xn	2-84
CNAA	COMPARATIVE NOT WITH A	2-90
CNAAQ	COMPARATIVE NOT WITH AQ	2-91
CNAQ	COMPARATIVE NOT WITH Q	2-90
CNAXn	COMPARATIVE NOT WITH Xn	2-91
CWL	COMPARE WITH LIMITS	2-85
DFAD	D-P FLOATING ADD	2-55
DFCMG	D-P FLOATING COMPARE MAGNITUDE	2-70
DFCMP	D-P FLOATING COMPARE	2-69
DFDI	D-P FLOATING DIVIDE INVERTED	2-64
DFDV	D-P FLOATING DIVIDE	2-63
DFLD	D-P FLOATING LOAD	2-50
DFMP	D-P FLOATING MULTIPLY	2-60
DFSB	D-P FLOATING SUBTRACT	2-58
DFST	D-P FLOATING STORE	2-51
DIS	DELAY UNTIL INTERRUPT SIGNAL	2-106
DIV	DIVIDE INTEGER	2-47
DRL	DERAIL	2-107
DUFA	D-P UNNORMALIZED FLOATING ADD	2-55
DUFM	D-P UNNORMAL FLOATING MULTIPLY	2-60
DUFS	D-P UNNORMALIZED FLOATING SUBTRACT	2-58
DVF	DIVIDE FRACTION	2-48
EAA	EFFECTIVE ADDRESS TO A	2-5
EABn	EFFECTIVE ADDRESS TO BASE n	2-20
EAPn	EFFECTIVE ADDRESS TO PAIR n	2-20
EAQ	EFFECTIVE ADDRESS TO Q	2-5
EAXn	EFFECTIVE ADDRESS TO Xn	2-11
ERA	EXCLUSIVE OR TO A	2-78
ERAQ	EXCLUSIVE OR TO AQ	2-78
ERQ	EXCLUSIVE OR TO Q	2-78
ERSA	EXCLUSIVE OR TO STORAGE A	2-79
ERSQ	EXCLUSIVE OR TO STORAGE Q	2-80
ERSXn	EXCLUSIVE OR TO STORAGE Xn	2-80
ERXn	EXCLUSIVE OR TO Xn	2-79
FAD	FLOATING ADD	2-54
FCMG	FLOATING COMPARE MAGNITUDE	2-68
FCMP	FLOATING COMPARE	2-67
FDI	FLOATING DIVIDE INVERTED	2-62
FDV	FLOATING DIVIDE	2-61
FLD	FLOATING LOAD	2-50
FMP	FLOATING MULTIPLY	2-59
FNEG	FLOATING NEGATE	2-65
FNO	FLOATING NORMALIZE	2-66
FSB	FLOATING SUBTRACT	2-57
FST	FLOATING STORE	2-51
FSTR	FLOATING STORE ROUNDED	2-52
FSZN	FLOATING SET ZERO AND NEGATIVE INDICATORS FROM MEMORY	2-71
GTB	GRAY TO BINARY	2-105
LACL	LOAD ALARM CLOCK	2-127
LAM	LOAD ASSOCIATIVE MEMORY	2-129
LBRn	LOAD ADDRESS BASE REGISTER n	2-21
LCA	LOAD COMPLEMENT A	2-9
LCAQ	LOAD COMPLEMENT AQ	2-10

LCQ	LOAD COMPLEMENT Q	2-10
LCXn	LOAD COMPLEMENT Xn	2-11
LDA	LOAD A	2-6
LDAQ	LOAD AQ	2-6
LDB	LOAD BASES	2-21
LDBR	LOAD DESCRIPTOR SEGMENT BASE REGISTER	2-22
LDCF	LOAD CONTROL FIELD	2-22
LDE	LOAD EXPONENT REGISTER	2-50
LDI	LOAD INDICATOR REGISTER	2-8
LDQ	LOAD Q	2-6
LDT	LOAD TIMER REGISTER	2-7
LDXn	LOAD Xn	2-7
LLR	LONG LEFT ROTATE	2-26
LLS	LONG LEFT SHIFT	2-28
LREG	LOAD REGISTERS	2-9
LRL	LONG RIGHT LOGIC	2-25
LRS	LONG RIGHT SHIFT	2-29
LXLn	LOAD Xn FROM LOWER	2-7
MME	MASTER MODE ENTRY 1	2-108
MME2	MASTER MODE ENTRY 2	2-109
MME3	MASTER MODE ENTRY 3	2-110
MME4	MASTER MODE ENTRY 4	2-111
MPF	MULTIPLY FRACTION	2-46
MPY	MULTIPLY INTEGER	2-45
NEG	NEGATE A	2-49
NEGL	NEGATE LONG	2-49
NOP	NO OPERATION	2-126
ORA	OR TO A	2-75
ORAQ	OR TO AQ	2-75
ORQ	OR TO Q	2-75
ORSA	OR TO STORAGE A	2-76
ORSQ	OR TO STORAGE Q	2-76
ORSXn	OR TO STORAGE Xn	2-77
ORXn	OR TO Xn	2-76
QLR	Q LEFT ROTATE	2-26
QLS	Q LEFT SHIFT	2-28
QRL	Q RIGHT LOGIC	2-25
QRS	Q RIGHT SHIFT	2-27
RCCL	READ CALENDAR CLOCK	2-128
RCU	RESTORE CONTROL UNIT	2-136
RET	RETURN	2-96
RMCM	READ MEMORY CONTROLLER MASK REGISTER	2-114
RPD	REPEAT DOUBLE	2-120
RPL	REPEAT LINK	2-123
RPT	REPEAT	2-117
RSW	READ SWITCHES	2-126
RTCD	RETURN DOUBLE	2-97
SAM	STORE ASSOCIATIVE MEMORY	2-130
SBA	SUBTRACT FROM A	2-38
SBAQ	SUBTRACT FROM AQ	2-39
SBLA	SUBTRACT LOGIC FROM A	2-41
SBLAQ	SUBTRACT LOGIC FROM AQ	2-42
SBLQ	SUBTRACT LOGIC FROM Q	2-42
SBLXn	SUBTRACT LOGIC FROM Xn	2-44
SBQ	SUBTRACT FROM Q	2-38
SBRn	STORE ADDRESS BASE REGISTER n	2-23
SBXn	SUBTRACT FROM Xn	2-39
SCU	STORE CONTROL UNIT	2-131
SDBR	STORE DESCRIPTOR SEGMENT BASE REGISTER	2-23
SMCM	SET MEMORY CONTROLLER MASK REGISTER	2-115
SMIC	SET MEMORY CONTROLLER INTERRUPT CELLS	2-116
SREG	STORE REGISTERS	2-14
SSA	SUBTRACT STORED FROM A	2-40
SSQ	SUBTRACT STORED FROM Q	2-40
SSXn	SUBTRACT STORED FROM Xn	2-41
STA	STORE A	2-13
STAC	STORE A CONDITIONAL	2-13
STAQ	STORE AQ	2-13
STB	STORE BASES	2-23
STBA	STORE CHARACTER OF A	2-16
STBQ	STORE CHARACTER OF Q	2-16
STC1	STORE INSTRUCTION COUNTER PLUS 1	2-18
STC2	STORE INSTRUCTION COUNTER PLUS 2	2-18
STCA	STORE CHARACTER OF A (SIX BIT)	2-15
STCD	STORE CONTROL DOUBLE	2-12
STCQ	STORE CHARACTER OF Q (SIX BIT)	2-15
STE	STORE EXPONENT REGISTER	2-53
STI	STORE INDICATOR REGISTER	2-17
STPn	STORE PAIR n	2-24
STQ	STORE Q	2-13
STT	STORE TIMER REGISTER	2-17
STXn	STORE Xn INTO UPPER	2-14
STZ	STORE ZERO	2-19
SWCA	SUBTRACT WITH CARRY FROM A	2-43
SWCQ	SUBTRACT WITH CARRY FROM Q	2-44
SXLn	STORE Xn INTO LOWER	2-14
SZN	SET ZERO AND NEGATIVE INDICATORS FROM MEMORY	2-86
TEO	TRANSFER ON EXPONENT OVERFLOW	2-101
TEU	TRANSFER ON EXPONENT UNDERFLOW	2-102

INDEX TO INSTRUCTIONS (Continued)

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form*

TITLE:	MODEL 645 PROCESSOR REFERENCE MANUAL	ORDER No.:	AH82, REV. 0
		DATED:	MAY 1972

ERRORS IN PUBLICATION:

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION:

(Please Print)

FROM: NAME ______________________ DATE: ____________

COMPANY ______________________

TITLE ______________________

____

____

CUT ALONG LINE

*Your comments will be promptly investigated by appropriate technical personnel, action will be taken as required, and you will receive a written reply. If you do not require a written reply, please check here. ☐

CUT ALONG LINE

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WELLESLEY HILLS,
MASS. 02181

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

POSTAGE WILL BE PAID BY:

HONEYWELL INFORMATION SYSTEMS
60 WALNUT STREET
WELLESLEY HILLS, MASS. 02181

ATTN: PUBLICATIONS, MS 050

FOLD ALONG LINE

Honeywell