PCP-125A

GE-PAC* 30
CONTROL COMPUTER

REFERENCE MANUAL

GENERAL ELECTRIC

GE-PAC 30
CONTROL COMPUTER

REFERENCE MANUAL

GENERAL ELECTRIC

GE 29-004R01

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

ILLUSTRATIONS

CHAPTER 1
SYSTEM ARCHITECTURE

1.1 INTRODUCTION

GE-PAC*30 Digital Systems are modularly structured to provide a high degree of flexibility in configuring application oriented systems. The building blocks used in the organization of a system are the Processor, Memory Modules, interface to peripheral devices, and system modules. See Figure 1-1.

GE-PAC 30 Digital Systems are designed for the user who has small-scale yet sophisticated requirements, and provide maximum system flexibility to solve a wide range of industrial control and scientific computational problems.

These third generation units use dual in-line integrated circuits to provide excellent reliability. The systems are modular,

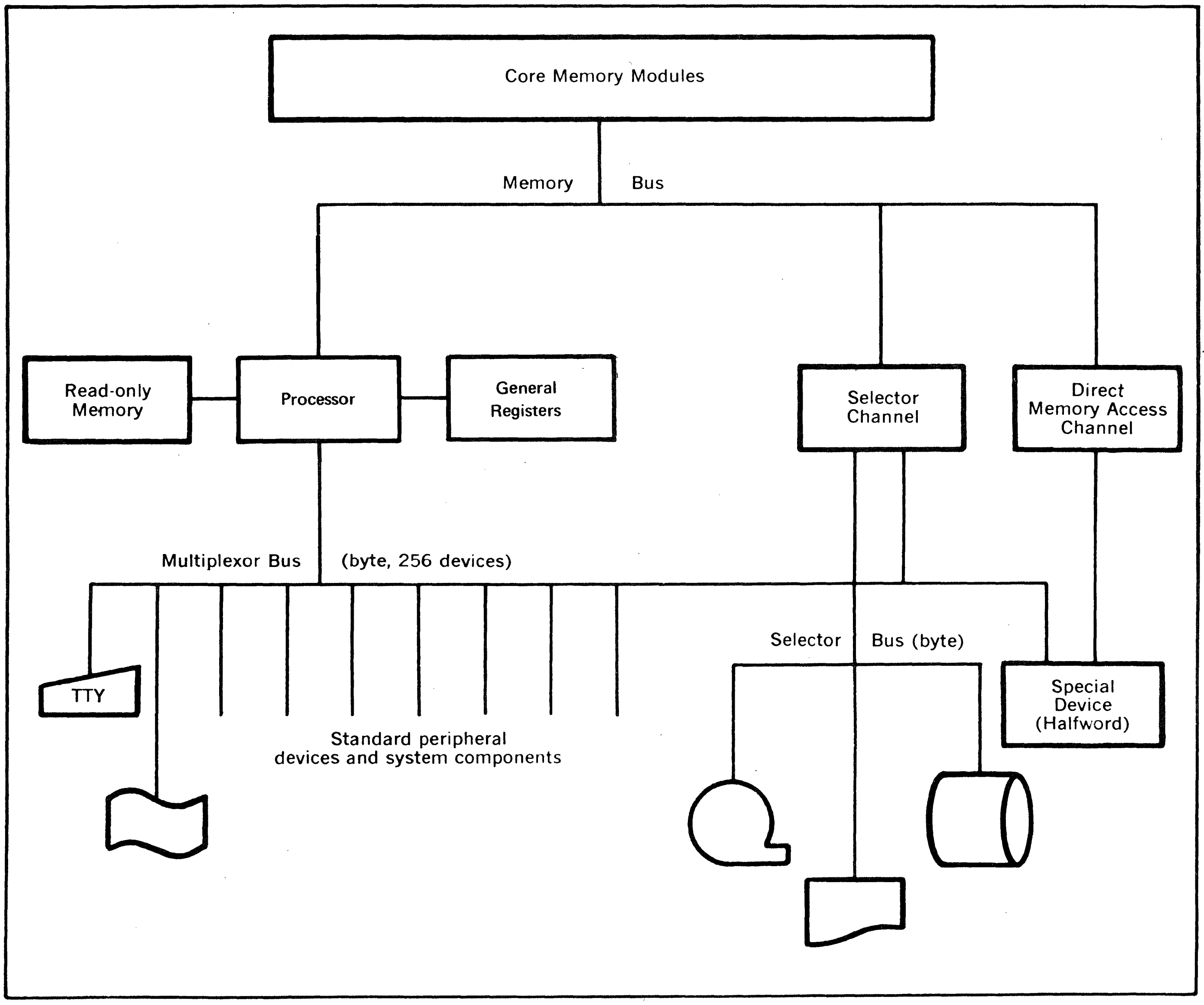

Core Memory Modules
Memory Bus
Read-only Memory
Processor
General Registers
Selector Channel
Direct Memory Access Channel
Multiplexor Bus (byte, 256 devices)
Selector Bus (byte)
TTY
Special Device (Halfword)
Standard peripheral devices and system components

Figure 1-1. GE-PAC 30 Digital Systems, Typical Block Diagram

* Registered Trademark of General Electric Company

furnishing the user with an expandable building block structure that can be adapted to a variety of system requirements. Standard units can easily be configured into operational systems for specialized requirements. This modularity and field expandibility, especially in the I/O area, provides a system which may be easily and economically adapted to changing system requirements.

Features of these systems include a memory that is addressable and alterable to the 8-bit byte level. Memory is field expandable from 1024 bytes to 65,536 bytes.

All memory is directly addressable with the primary instruction word; no paging or indirect addressing is required.

Sixteen 16-bit general purpose registers can be used as accumulators or index registers.

Register-to-register instructions permit operations between any two of the 16 General Registers, eliminating redundant loads and stores.

A comprehensive instruction set includes efficient byte processing instructions, single instructions for loop control which increment, test and branch on indexing values, as well as instructions that test the condition code and branch directly to any location in memory.

Logical and arithmetic shift instructions can shift up to 15 bit positions with a single instruction.

A flexible Systems Interface includes an integrated priority interrupt facility and provides for connecting up to 256 devices.

GE-PAC 30 Digital Systems have third generation data compatibility including ASCII and EBCDIC information codes.

1.2 SCOPE OF MANUAL

This manual is intended as a general reference to all GE-PAC 30 Digital Systems. Because of this general nature, all information provided does not apply equally to all GE-PAC 30 Models. On the contrary, some features described are optional, and/or only available on the more sophisticated systems.

1.3 PROCESSOR ORGANIZATION

The various elements of the system are organized around the primary controlling unit - The Processor. The Processor contains facilities for:

1. Arithmetic and logical processing of data
2. Sequencing instructions in the required order
3. Fetching and storing information
4. Addressing memory
5. Initiating or controlling communications with external devices
6. Changing states in response to interrupts

The Processor consists of a group of sixteen 16-bit General Registers, an Arithmetic/Logical Unit (ALU), and a Read-Only-Memory (ROM) control unit. Figure 1-2 is a block diagram of a GE-PAC 30 Digital System.

1.3.1 General Registers

The General Registers can be used as accumulators in fixed-point arithmetic and logical operations, or as index registers in address arithmetic and indexing operations. Each register has a capacity of sixteen binary digits, which is one halfword. For some operations, such as multiplication and division, two adjacent registers are coupled to form a 32-bit fullword. In 8-bit byte operations the rightmost 8 bits of a General Register are used.

1.3.2 Arithmetic/Logical Unit

The Arithmetic/Logical Unit (ALU) processes binary integers, floating-point fractions, and logical information. The operands are located in the General Registers and/or core

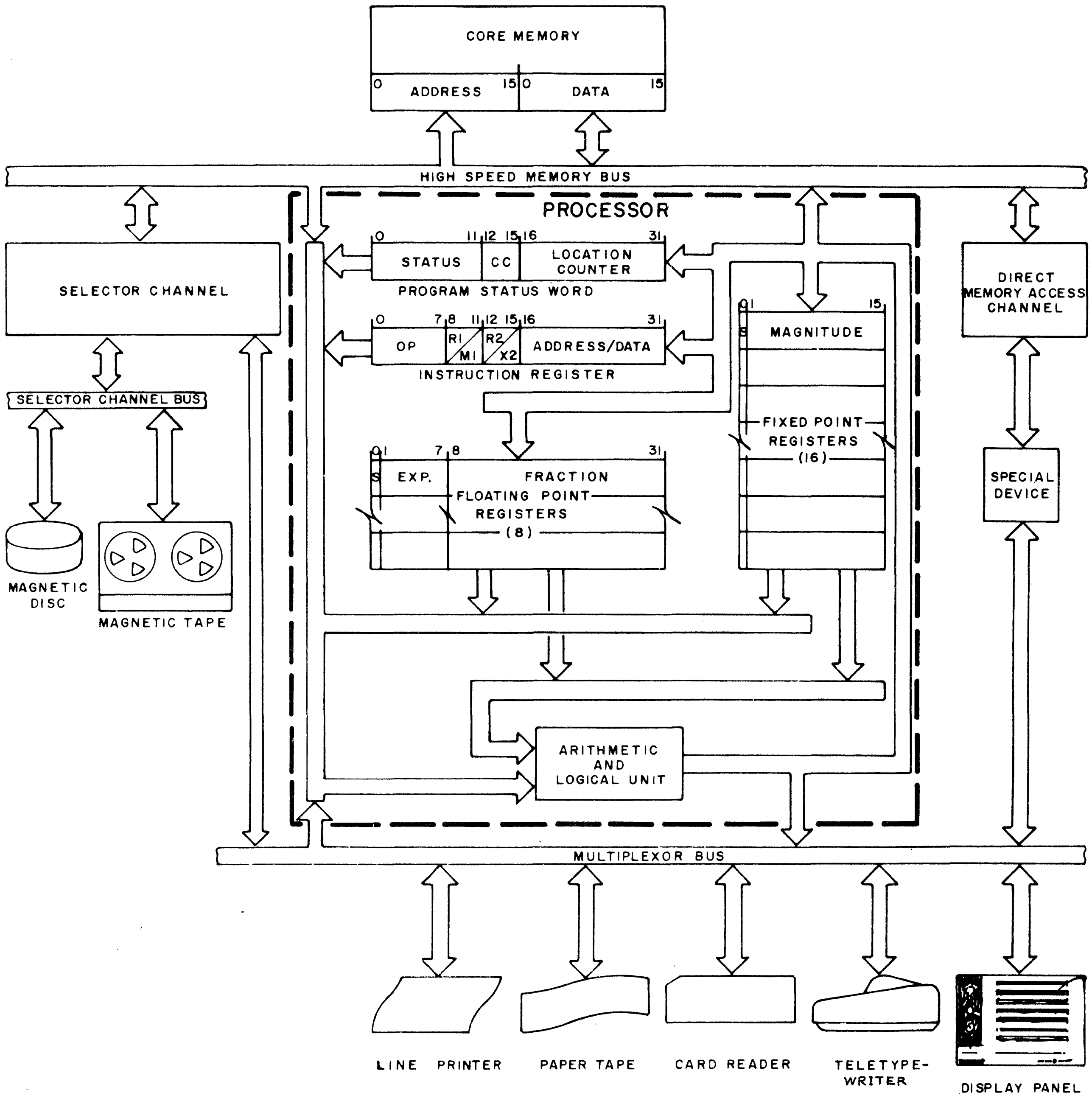


Figure 1-2. System Block Diagram

memory. Fixed-point data is treated as signed, 15-bit integers in the halfword format, or as signed, 31-bit integers in the fullword format. Positive numbers are expressed in true binary form with a sign bit of zero. Negative numbers are represented in two's complement form with a sign bit of one. The numeric value of zero is always represented as positive. Table 1-1 shows several examples of the fixed-point number representation used in GE-PAC 30 Systems.

TABLE 1-1. EXAMPLES OF FIXED-POINT REPRESENTATION

Number	Decimal	Binary
$2^{15}-1$	32767	0111 1111 1111 1111
2^0	1	0000 0000 0000 0001
0	0	0000 0000 0000 0000
$-(2^0)$	-1	1111 1111 1111 1111
$-(2^{15})$	-32768	1000 0000 0000 0000

All fixed-point operations are performed upon one operand in a General Register with the other operand in either a General Register or a core memory location.

Multiple-precision arithmetic operations are made convenient by the two's complement representation, and by recognition of the carry/borrow from one operation to another.

Some GE-PAC 30 Digital Systems provide the capability for floating-point arithmetic operations. The GE-PAC 30 format for single-precision, floating-point data is identical to that used in the IBM System/360. This format represents numbers in the range from 5.4 X 10^{-79} to 7.2 X 10^{75}, with six digits of precision.

A floating-point number consists of a signed exponent and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess 64 binary notation; the fraction is expressed as a hexadecimal number having a radix point to the left of the high order digit. Table 1-2 provides several examples of the floating point number representation.

TABLE 1-2. EXAMPLES OF FLOATING-POINT NUMBER REPRESENTATION

Value	Binary
1.0	0100 0001 0001 0000 0000 0000 0000 0000
-1.0	1100 0001 0001 0000 0000 0000 0000 0000
9.5	0100 0001 1001 1000 0000 0000 0000 0000
-0.5	1100 0000 1000 0000 0000 0000 0000 0000
$-(1-16^{-6})\cdot 16^{63}$	1111 1111 1111 1111 1111 1111 1111 1111
-16^{-65}	1000 0000 0001 0000 0000 0000 0000 0000
$0.1 + 16^{-6}$	0100 0000 0001 1001 1001 1001 1001 1010

1.3.3 Control Unit

The Processor operates under the direction of a control unit which has a pre-wired micro-program contained in the Read-Only-Memory (ROM). The micro program is a sequence of micro operations which fetches the Processor instructions, decodes them, and processes the operands located in the General Registers and core memory locations.

For example, to fetch an instruction, the micro-program loads the memory address register with the instruction address, commands a memory read operation, and when the memory data is ready, transfers the content of the memory data register to the working register.

1.3.4 Memory

GE-PAC 30 Systems provide for connection of multiple memory blocks on a Memory Bus to the Processor. Each memory block consists of a magnetic core memory plane with independent Read/Write Control.

The 16-bit halfword data register permits all 16-bit instructions and arithmetic or logical data to be handled in a single memory cycle. Multiple halfword data requires an additional memory cycle for each 16-bit halfword. Byte operations are performed by selectively manipulating the right or left 8 bits of the 16-bit halfword.

Memory elements can be expanded to a maximum dynamic addressing range of 65,536 8-bit bytes or 32,768 16-bit halfwords.

The optional Memory Parity feature provides for checking of all data transfers in and out of memory.

1.4 STORAGE WORD FORMATS

The GE-PAC 30 Instruction Set manipulates data of three different word lengths: 8 bit bytes, 16 bit halfwords or 32 bit fullwords. In each format the bits are numbered from left to right, starting with the number zero. The format for each word length is shown on Figure 1-3.

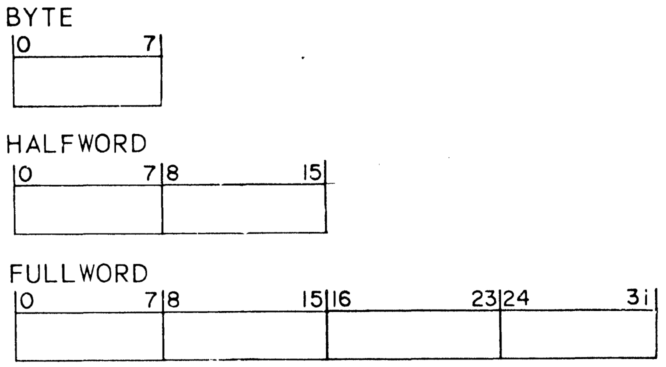


Figure 1-3. Storage Word Formats

1.4.1 Hexadecimal Notation

Binary information is expressed in hexadecimal notation (base 16) in the GE-PAC 30 Systems. Four Binary bits of information can be expressed by a single hexadecimal digit. Thus, byte information can be expressed by a string of two hexadecimal digits, halfword information by four hex digits, and fullword information by 8 hex digits. Table 1-3 lists hexadecimal, binary, and decimal equivalents.

TABLE 1-3. HEXADECIMAL NOTATION

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

1.4.2 Arithmetic Data

The basic fixed-point arithmetic operand is the 16-bit halfword. In multiply and divide operations, 32-bit fullwords are manipulated. See Figure 1-4.

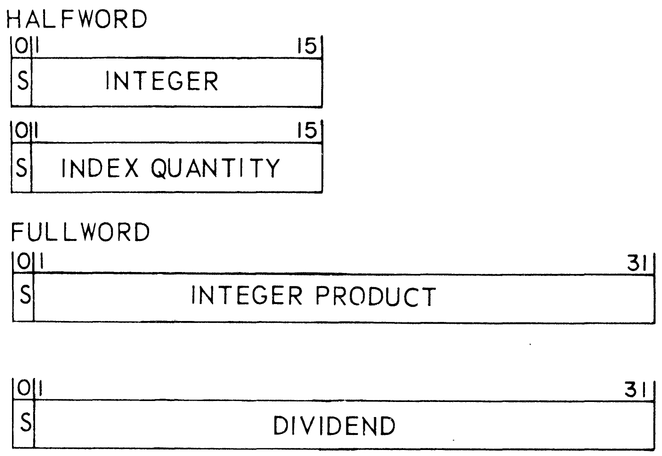
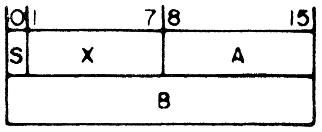


Figure 1-4. Fixed-Point Word Formats

The halfword arithmetic operand matches the address field of an instruction, permitting fixed-point arithmetic instructions to be used for address arithmetic. Arithmetic, logical, and shift instructions can also be used for address manipulation or computation.

Each floating-point value requires two halfwords. The floating-point format is shown in Figure 1-5.



S = sign of the fraction
X = exponent, in excess 64 code
AB = fraction

Figure 1-5. Floating-Point Word Format

Sign and magnitude representation is used, in which the sign bit S is zero for positive values, and one for negative values. The fraction AB contains six hexadecimal digits as shown in Figure 1-6. The value of a floating-point fraction can be expressed as:

$$F_1.16^{-1}+F_2.16^{-2}+F_3.16^{-3}+\dots+F_6.16^{-6}$$

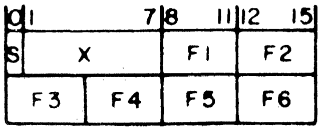


Figure 1-6. Floating-Point Word Layout

A normalized floating-point number has a non-zero, high-order hexadecimal fraction digit (F_1). If one or more high-order fraction digits ($F_1F_2\ldots$) are zero, the number is said to be unnormalized. The range of the magnitude (M) of a normalized floating-point number is:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \cdot 16^{63}$$

or approximately

$$5.4 \cdot 10^{-79} \leq M \leq 7.2 \cdot 10^{75}$$

All floating point numbers are assumed to be normalized prior to their use as operands. No pre-normalization is performed, all results are post-normalized. The floating-point load instruction will normalize unnormalized floating-point numbers.

Exponent overflow is defined as a resultant exponent greater than +63. Exponent underflow is defined as a resultant exponent less than -64. The Overflow flag is set whenever exponent overflow or underflow is detected. If overflow, the exponent and fraction of the result are set to all ones. The sign of the result is not affected by the overflow. If underflow, the sign, exponent and fraction of the sum are set to zero.

The floating-point value in which all data bits are zero is called true zero. A true zero may arise as the result of an arithmetic operation due to exponent underflow, or when a result fraction is zero due to loss of significance. In general, zero values participate as normal numbers in all arithmetic operations. If the resultant exponent of an addition, subtraction, multiplication, or division overflows, all bits of the exponent and fraction are set, and the correct sign is generated.

The floating-point registers have even numbers. The register address specified by the R_1 and R_2 fields should be even numbers (0, 2,4,6 etc.) otherwise the next lower even register will be used. There are eight 32-bit floating-point registers available. The floating-point registers are separate from the general registers and are addressable only by floating-point instructions.

1.4.3 Logical Data

Logical information is handled as 16-bit halfwords or as 8-bit bytes. Halfword operations are performed on all 16 bits of an operand located in memory or a General Register. Logical data is subject to such operations as AND, OR, EXCLUSIVE OR, and COMPARE LOGICAL.

Load Byte and Store Byte instructions are provided to facilitate byte manipulation. These instructions, when combined with indexed addressing, enable the processing of input/output character strings.

1.4.4 Information Positioning

Core memory locations are numbered consecutively, beginning at location 0000, for each eight bit byte. Since the address field of an instruction word is 16-bits in length, each of the 65,536 bytes in memory is directly addressable with the primary instruction word.

The GE-PAC 30 System transmits binary information between memory and the Processor as 16-bit halfwords. The instruction being performed determines if the address specified is that of a byte, a halfword or a fullword. If a byte of information is desired, either the left or right byte of the halfword read from memory is manipulated as determined by the specific address. If a halfword of information is desired, the entire 16 bits read from memory are used. If a fullword is desired, a second 16 bits is read from memory and combined with the original halfword.

Bytes of information are addressed by their specific hexadecimal address. A group of bytes combined to form a halfword or a full word are addressed by the leftmost byte in the group. Halfwords are positioned so that the address is a multiple of 2. Fullwords are positioned so that the address is a multiple of 4. Table 1-4 illustrates the addressing scheme. Table 1-5 lists the valid last hexadecimal digits for each type of addressing.

TABLE 1-4. MEMORY ADDRESS DATA

	Hexadecimal Address							
	0050	0051	0052	0053	0054	0055	0056	0057
Hexadecimal Contents	01	23	45	67	89	AB	CD	EF
Word Length Positions	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
	Halfword		Halfword		Halfword		Halfword	
	Fullword				Fullword			

TABLE 1-5. PERMISSIBLE ADDRESSES

Word Length Desired	Last Hex Digit of Address
Byte	any
Halfword	0,2,4,6,8,A,C,E
Fullword	0,4,8,C

Refer to Table 1-4. If the address specified were 0050:

1. A byte oriented instruction would extract the data constant 01_{16} as its operand.
2. A halfword oriented instruction would extract the data constant
3. 0123_{16} as its operand.
3. A fullword oriented instruction would extract the data constant 01234567_{16} as its operand.

1.5 INSTRUCTION WORD FORMATS

Instructions in GE-PAC 30 Systems have three formats:

1. Register to Register [RR]
2. Register to Indexed Memory [RX]
3. Register to Storage [RS]

In general, each format specifies three things: The operation to be performed, the address of the first operand, and the address of the second operand. The first operand is normally a General Register which contains the result of a previous operation. The second operand is normally the contents of a General Register, the contents of a core memory location, or a data constant used as the other participating operand.

A 16-bit halfword format is used for register to register operations. A 32-bit fullword format is used for the register to indexed memory, and the register to storage formats. The specific formats are shown on Figure 1-7.

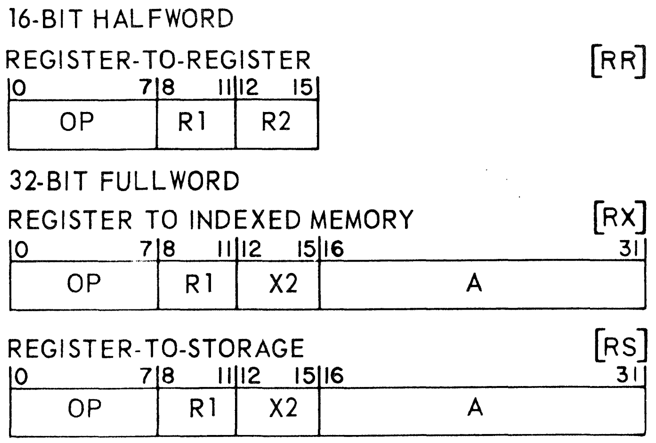


Figure 1-7. Instruction Word Formats

The 8-bit OP field in all three formats specifies the machine operation to be performed. The operation code can be written as two hexadecimal characters.

The 4-bit R1 field in the three instruction formats specifies the address of the first operand. The R1 field is normally the address of a General Register and is written as one hexadecimal character.

The 4-bit R2 field in the RR instruction format specifies the address of the second operand. The R2 field is always a register address and is written as one hexadecimal character.

The 4-bit X2 field in the RX and RS formats specifies a General Register whose content is used as an index value. The X2 field is always the address of a General Register and is written as a single hex character.

The 16-bit A field specifies a memory address in the RX format, or contains an integer value to be used as an immediate operand in the RS format. It is written as a string of four hex characters.

The RR instructions are used for operations between two registers. The first operand is the contents of the register specified by the R1 field of the instruction word. The second operand is the contents of the register specified by the R2 field.

The RX instructions are used for operations between a register and memory with the option of indexing. The first operand is the register specified by the R1 field of the instruction word. The second operand is the contents of the memory location specified by the A field of the instruction word, or by the sum of the A field and the contents of the General Register specified by the X2 field if indexing is specified.

In the RS instructions, the first operand is the contents of the General Register specified by the R1 field of the instruction word. The second operand is the number contained in the A field, or the number generated by adding the A field to the contents of the General Register specified by the X2 field if indexing is specified. The second operand of an RS instruction specifies the number of bit positions in shift instructions, or forms the second operand in immediate instructions. An immediate operand is two bytes of data used as an operand and carried in the halfword address field itself. The value in the address field is treated as a signed integer instead of a memory location address.

For the Branch on Condition instructions the first operand is the M1 field. This field is a 4-bit mask which is to be tested against the condition code contained in the Program Status Word.

Table 1-6 summarizes the first and second operand designations for each instruction format.

TABLE 1-6. DESIGNATIONS FOR FIRST AND SECOND OPERANDS

First Operand:	The contents of the register specified by the R1 Field (R1).	RR, RX and RS
	The M1 Field	RR and RX, Branch on Condition.
Second Operand:	The contents of the register specified by the R2 Field (R2).	RR
	The contents of the address derived by adding the A field and the contents of the General Register specified by the X2 field. [A + (X2)]	RX
	The A field plus the contents of the General Register specified by the X2 field. A + (X2)	RS

All instructions are aligned on halfword boundaries. The RR instruction format is a 16-bit halfword; the RX and RS formats are 32-bit fullwords which are treated as two halfwords for alignment purposes. This permits mixing of halfword and fullword instructions without the requirement of halfword No Operation instructions to force fullword instruction alignment.

1.6 GENERAL REGISTERS AND STORAGE ADDRESSING

1.6.1 General Registers

The sixteen General Registers function as accumulators or index registers in all arithmetic and logical operations. Each General Register is a 16-bit halfword consisting of two 8-bit bytes. For arithmetic operations, bit zero (leftmost position) is considered the sign bit. Bit one is the most significant bit.

The General Registers are numbered from zero to fifteen (decimal) which is written in hexadecimal notation as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. General Register addresses are only permitted in the R1, R2 and X2 fields of an instruction word.

The General Registers have not been given specific functional assignments. However, the following operational restrictions should be noted:

1. It is not possible to use General Register zero as an index register. In the RX and RS instruction formats, a zero entry in the X2 field indicates that no indexing is to take place.
2. The first operand (R1) must specify an even numbered General Register for multiplication and division operations.
3. The first operand (R1) for the Branch on Index instructions specifies the first of three general registers. General Register D is the maximum value for R1 in this case.

1.6.2 Storage Addressing

Locations in core memory are addressed by the RX instruction. The address portion, A, of the instruction is a 16-bit halfword, making it possible for the address field to specify all 65,536 bytes, the maximum available memory.

If an address specified is greater than the highest memory location available, no memory access takes place, and a word consisting of all zeros is used in place of the word normally read from memory.

Programs cannot be looped from the highest memory location back to location 0000.

1.6.3 Address Modification by Indexing

The General Registers in GE-PAC 30 systems facilitate address modification. Fifteen different General Registers may be used as in index registers for this purpose.

If the contents of the A field of an instruction word are to be modified, the address of the General Register, whose content is to be used as the modifier, is placed in the X2 field of the instruction word. During decoding of the instruction word, the contents of the specified index register is added to the A field to obtain the effective address of the second operand. The index value in a General Register may be signed to permit indexing in either direction.

All of the General Registers except General Register Zero may by used as index registers. If the X2 field of the instruction word is zero, no indexing is specified, and the A portion of the instruction word is not modified. Thus, General Register Zero cannot be used as an index register.

1.7 PROGRAM STATUS WORD

The 32-bit Program Status Word (PSW) contains the information required for program execution. The PSW has a 12-bit Status field, a 4-bit Condition Code field, and a 16-bit Instruction Address field. See Figure 1-8.

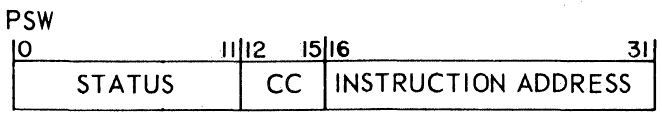


Figure 1-8. Program Status Word Format

In general, the Program Status Word is used to control instruction sequencing and to store indications of the status of the system in relation to the program currently being executed. The active or controlling PSW is referred to as the current PSW. When a program interrupt occurs, the current PSW is automatically preserved for subsequent reinstatement or inspection. By loading a new PSW, the status of the Processor can be changed.

1.7.1 Status

The status of the current user program is defined by bits 0 through 11 of the Program Status Word. When bit 0 is set the Processor is halted in a high speed, interruptable wait loop during which interrupts will be recognized immediately. When bit 0 is reset, the Processor is active and interrupts which are enabled will be recognized after execution of the current instruction. Bits 1 through 11 are mask bits for interrupts.

Assignment of the Status bits is listed on Table 1-7.

1.7.2 Condition Code

The 4-bit Condition Code (CC) of the Program Status Word is set after execution of arithmetic, logical, shift, and input/output instructions. In general, the condition code bits 12 through 15 indicate Carry, Overflow, Greater, and Less, in that order. The condition code setting has a different interpretation when set by an input/output instruction and is described in that section.

Following an arithmetic operation the condition code indicates whether the result was greater or less than zero, whether a carry or borrow took place, and whether an overflow has occurred.

TABLE 1-7. PSW STATUS BIT ASSIGNMENTS

PSW bit	Assignment
0	Wait state
1	External Interrupt Enable
2	Machine Malfunction Interrupt Enable
3	Fixed-point Divide Fault Interrupt Enable
4	Reserved
5	Floating-point Divide Fault Interrupt Enable
6 thru 11	Not Assigned

Assignment of Condition Code bits is listed on Table 1-8.

TABLE 1-8. PSW CONDITION CODE BIT ASSIGNMENTS

PSW Bit	Assignment	Symbol
12	Carry/Borrow	(C)
13	Overflow	(V)
14	Greater than zero	(G)
15	Less than zero	(L)

1.7.3 Instruction Address

The 16-bit Instruction Address field of the Program Status Word specifies the location of the next instruction to be fetched and processed. The sixteen bit address field has the capability of addressing the maximum core memory of 32,768 halfwords.

After instruction execution, the instruction Address Field is incremented by 2 if the executed instruction was in the halfword RR format (2 bytes). The Address Field is incremented by 4 if the executed instruction was in the fullword RX or RS format (4 bytes).

1.7.4 Instruction Execution

During normal processing of a program, instructions are fetched from the location specified by the Instruction Address, the instruction is executed, the Instruction Address is incremented, and another fetch and execute cycle begins.

This sequence can be changed when a two-way conditional choice is required, for entrance and return to and from a subroutine, or for iterative groups of instructions, called loops.

Subroutine linkage provides for the introduction of a new Instruction Address and preservation of the incremented current Instruction Address as the location for return to the main program. The instruction that provides this facility is the Branch and Link instruction.

Decision making is implemented by the Branch on Condition instructions which inspect the setting of the 4-bit Condition Code (PSW 12:15)

Loop control can be performed by the conditional branch when it tests the outcome of arithmetic and counting operations. For frequent combinations of such tests, the Branch on Index instructions provide a convenient means of performing these tasks.

1.8 INTERRUPT SYSTEM

System interrupts are provided to detect the presence of illegal instructions, machine malfunctions, divide faults, and requests for service from external devices. The control of interrupts centers around the Status field of the Program Status Word (PSW (0:11)). A zero in this field disables an interrupt; a one in this field enables an interrupt.

The PSW which defines the operating status of the Processor is called the current PSW. There are five additional Program Status Words, each associated with a specific class of interrupt. The new PSW defines the action to be taken for each type of interrupt; the old PSW is a reserved storage area in which the current PSW is placed when an interrupt is recognized.

Each new PSW re-defines the status of the machine, usually inhibiting interrupts of its own class, or possibly all interrupts. The instruction address field of each new PSW specifies the starting location of the sub-program to service the interrupt condition. Exit from an interrupt service sub-program is accomplished by the Load Program Status Word instruction specifying the stored old PSW. This restores the machine status and the instruction address which was current at the time the interrupt occurred.

The Dedicated core locations of the re-definitive Program Status Word Pairs varies from model to model and is given in separate documents pertinent to a particular system.

1.8.1 Interrupt Procedure

After execution of each instruction, the Processor interrogates for interrupts. If an interrupt is found pending and the appropriate bit in the Status Field of the PSW is a one (enabled) the interrupt will take place. The current PSW is automatically stored as the old PSW for the class of interrupt which is to be serviced and the new PSW for the class of interrupt being serviced becomes the current PSW. After the sequence of instructions servicing the interrupt has been completed, the old PSW for the class of interrupts being serviced is normally loaded and becomes the current PSW.

Note that the new PSW location is not altered by this interrupt procedure, so that subsequent interrupts of the same class will be serviced in the same manner. The old PSW location serves as a temporary storage register for exit from the interrupt service sub-program and may vary each time an interrupt request is processed.

If an interrupt request occurs and the appropriate bit in the Status Field of the PSW is a zero (disabled) an interrupt will not occur and the request is ignored.

External interrupt requests from peripheral devices remain pending, that is the interrupt request will be repeated after execution of each instruction, until enabled by the PSW and serviced by the program. Program restart use of the Initialize switch clears pending interrupts from external devices.

1.8.2 Acknowledgement of External Interrupts

The Acknowledge Interrupt instruction clears the interrupt request and returns the device address and status byte from the peripheral causing the interrupt. The rightmost 4 bits of the status byte are copied into the condition code (PSW 12:15) while the leftmost 4 bits of the status byte have meanings unique to each peripheral device. See Figure 1-9. The device number and device status byte provide sufficient information to determine the cause and action required by any external interrupt.

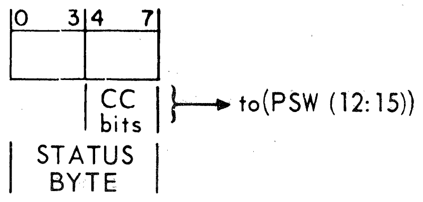


Figure 1-9. Status Byte Format

1.8.3 Internal Interrupts

Interrupts which originate in the Processor are the Illegal Instruction, Machine Malfunction, and Divide Fault Interrupts.

The Illegal Instruction interrupt is not represented by an enabling bit in the PSW, and is therefore always operative. An illegal instruction is defined as an operation code which cannot be decoded into a legal representation for processing. No attempt is made to execute the illegal instruction, nor is the instruction address field of the PSW incremented. Therefore, the old PSW stored as a result of the illegal instruction interrupt points to the address of the illegal instruction.

The Machine Malfunction Interrupt, enabled by bit 2 of the Program Status Word, is indicative of a Processor failure from which no programmed recovery can be made. The Machine Malfunction Interrupt is generated by Memory parity error. When the memory parity option is present in the Processor, a parity bit is appended to each byte of memory. The parity bit is set to maintain odd parity. That is, if a memory byte contains an odd number of ones the parity bit is zero; if the memory byte contains an even number of ones, the parity bit is one.

Parity is recomputed for each byte transfer, and the parity bits of the transferred byte and the original byte are compared. If the parity bits are different, and bit 2 of the Program Status Word is set to enable the interrupt, a Machine Malfunction Interrupt is generated.

The fixed-point Divide Fault interrupt, enabled by bit 3 of the Program Status Word, is indicative of quotient overflow. The interrupt takes place prior to alteration of the operand registers, permitting the interrupt service subroutine to examine these values.

The floating-point Divide Fault interrupt, if enabled by bit 5 of the current Program Status word, results from a floating-point division by zero.

1.8.4 Power Failure

When power failure is detected, the instruction being executed is completed and the Processor and memory are put in a locked state. Power up will initialize the Processor to the status at the time of power failure. The Processor will be placed in the Halt mode, from which normal execution may proceed.

1.8.5 High Speed Interrupt Option (GE-PAC 30-2 only)

Core location X'0020 (the High Speed Interrupt Pointer) contains the starting address of an eight byte block defined as follows:

PSW	(save)
LOC	(save)

The first fullword is a save area for the current PSW when a High Speed Interrupt is taken. The next fullword is the next instruction to be per formed. This instruction should be a branch to a service subroutine as an automatic "push-down" takes place when High Speed Interrupts occur. Location X'0020' is incremented by eight every time the interrupt takes place, defining another eight byte block. For example, if location X'0020' contains the address X'1300', when the High Speed Interrupt is taken, the Current Program Status Word is saved in location X'1300'; the Pointer (location X'0020') is incremented by eight; the hardware PSW is set to zero (disabling all other categories of interrupts) and the location counter is set to X'1304'. Fullword X'1304' should contain a branch to a service subroutine. If, during this service subroutine, another High Speed Interrupt occurs, the same actions take place and another service subroutine is entered. As many interrupt service subroutines as the programmer anticipates may be so nested. (See CAUTION)

When a service subroutine is completed, those unfinished subroutines entered earlier must be completed before the main program is re-entered. This reversal of the "push-down" process is done through the Un-Chain (UNCH) instruction.

The Un-Chain instruction has the following format:

UNCH R1, R2 (RR)

0 7	8 11	12 15
90	R1	R2

The Un-Chain instruction decrements the High Speed Interrupt Pointer (location X'0020') by eight and loads the Program Status Word from the fullword core location specified by the new value of location X'0020'.

The first and second operands (R1 and R2) may indicate any register. Neither operand will be changed.

The following is a sample of "push-down" stack programming.

```
HSINTA     DS     2
           DS     2
           B      SRVCA
HSINTB     DS     2
           DS     2
           B      SRVCB
HSINTC     DS     2
           DS     2
           B      SRVCC
```

If the UNCH instruction is terminating service subroutine SRVCC, the PSW is loaded from location HSINTC which returns program control to the remainder of SRVCB.

Figure 1-10 depicts a three level chain and un-chain process which eventually returns control to the main program with the High Speed Interrupt Pointer back to its original value.

The High Speed I/O Interrupt, available as an option on GE-PAC 30-2, operates on a priority above the normal I/O Interrupt. If I/O Attention and Fast I/O Attention occur simultaneously, the Fast I/O Interrupt is serviced first. There is no enabling bit in the PSW for Fast I/O Interrupt.

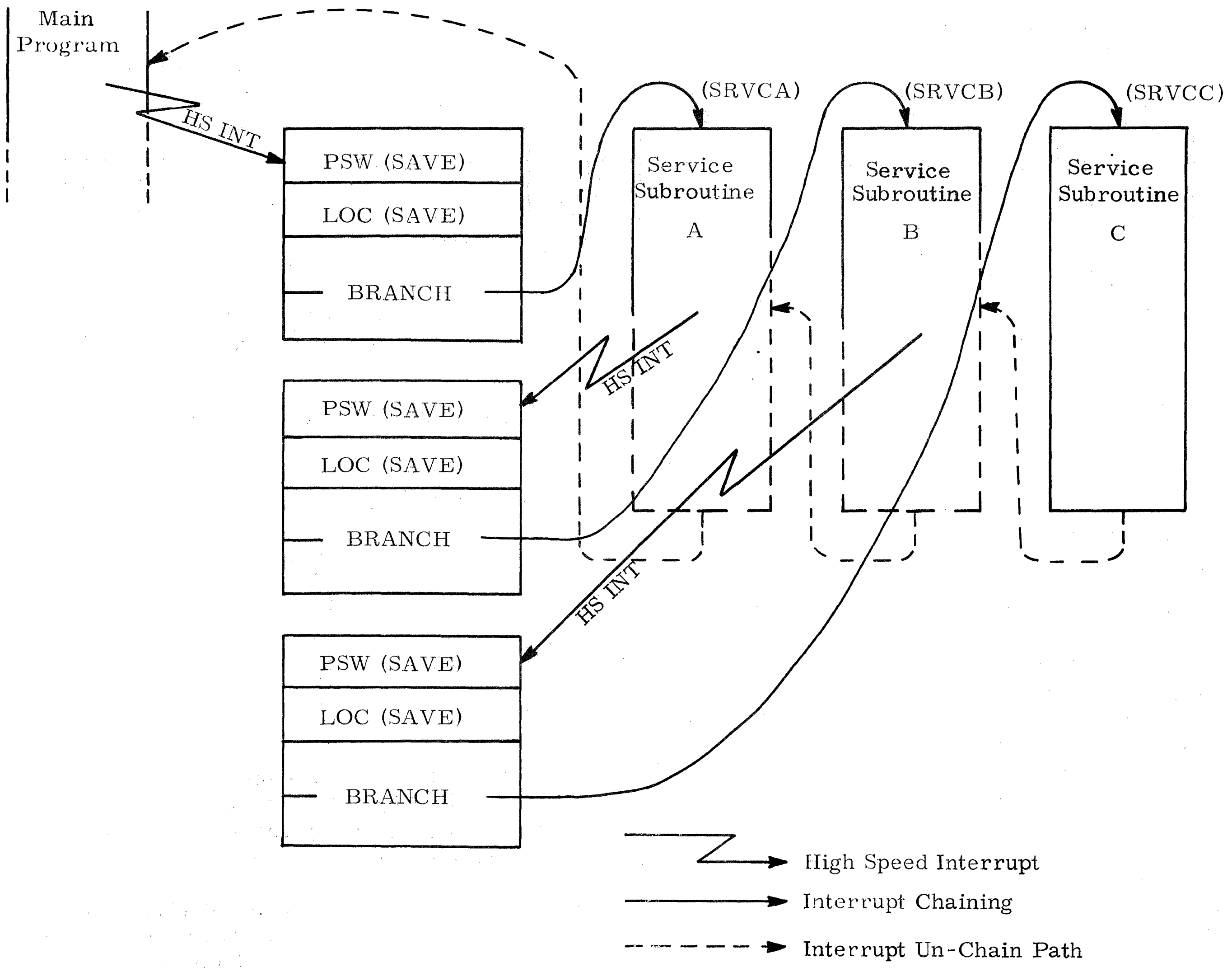


Figure 1-10 Interrupt Chaining

The fact that the High Speed Interrupt cannot be disabled in the Processor presents some interresting consequences. Too many interrupts or a steady interrupt (always active) will rapidly fill memory with "push-down" stacks unless a procedure similar to that shown below is implemented.

```
.
.
.
DS      X'2'
DS      X'2'
B       SRVCZ          Last Service Subroutine
DS      X'2'
DS      X'2'
UNCH      0,0
```

The hardware guarantees that the instruction following the PSW save area will be performed before High Speed Interrupts are again accepted.

The UNCH instruction, therefore, will cause an immediate return to the point of interruption in the last service subroutine, effectively ignoring all excess interrupts.

1.9 INPUT/OUTPUT SYSTEM

GE-PAC 30 Systems can transfer information between the Processor and peripheral devices in several modes:

1. A single 8-bit byte at a time through the General Registers.

2. A single 8-bit byte at a time through core memory.
3. A block of information at a time (string of bytes) under Processor control.
4. A block of information directly from, or to memory and the peripheral device under control of an optional Selector Channel.

1.9.1 Basic Input/Output Programming

In general, any data transfer requires a series of operations concerned with the device or system with which information is being transferred. Before data can be transferred, the device or system must be able to accept a command. The Output Command instructs the device to perform such functions as: switch to send mode, switch to receive mode, go forward, etc. Once the device is in the correct mode of operation, the data transfer can take place.

There are two methods of input/output programming. The first method, called program controlled, interrogates the device to determine if it is ready to transfer data, and waits if necessary until transfer can take place. The second method, called interrupt controlled, permits the device to demand service when the device itself is ready for data transfer.

Either method of input/output, program controlled or interrupt controlled, can be used with the Read Data and Write Data instructions to transfer information to or from the General Registers or core memory.

1.9.2 Program Controlled Input/Output

Program controlled data transfer can be accomplished in many ways. The exact sequence of instructions depends on the particular device with which data transfer is to take place. The following steps describe the general approach to program controlled data transfer.

1. An Output Command which specifies the function to be performed is sent to the device.
2. A Sense Status instruction sets the condition code, indicating the state of the device, i.e., busy, device unavailable, etc.
3. A Branch on True Condition instruction waits for the not true condition. In this case the branch is taken back to the Sense Status instruction. The effect of this is to produce a wait loop until the device is able to transfer data.
4. When the Branch on True Condition fails, the device is ready to transfer data. The next instruction, Read Data or Write Data, causes the data transfer to take place.
5. If more than a single byte of information is to be transferred, additional steps are required for indexing. A typical procedure would be:
    1. Initialize general registers with an index value and increment
    2. Output Command
    3. Sense Status
    4. Branch on True Condition to sense status if not ready
    5. Read Data, indexed
    6. Branch on Index to cause increment and test for number of characters input.

1.9.3 Interrupt Controlled Input/Output

Interrupt controlled data transfer involves the same basic principles used for program controlled data transfer. The important difference is that the device is permitted to interrupt when ready to transfer data. The wait loop is eliminated and the time saved can be used for internal processing. The following steps describe the general approach to interrupt controlled data transfer.

1. Device signals Processor with an interrupt request.
2. An Acknowledge Interrupt instruction returns the device address and status byte to the Processor.
3. A Read/Write Data instruction causes data transfer to take place.

1.9.4 Block Input/Output Programming

The Optional Read Block and Write Block instructions greatly simplify programming of strings of data. The single instruction causes information to be transferred between a device and sequential locations in core memory. Transfer is terminated when a pre-determined location is reached, or when an unusual device status is encountered.

Prior to block transfer, an Output Command and Sense Status instruction are used to specify the function and test the status of the device. The block transfer instruction can then perform all remaining steps of input/output. Note that the complete attention of the processor is given to the data block transfer and that normal processing will not resume until completion of this instruction.

1.9.5 Condition Code for Input/Output

The 4-bit Condition Code (CC) of the Program Status Word is set after execution of input/output instructions and the device interrupt and control instructions. The interpretation of the condition code after an input/output instruction differs from the setting caused by arithmetic and logical operations.

Following an input/output or device control instruction, the condition code indicates the device response such as available, busy, or unavailable. It is important to note that data transfer cannot take place until all bits of the condition code are zero.

Assignment of Condition Code bits for input/output is shown on Table 1-9.

TABLE 1-9. PSW CONDITION CODE BIT ASSIGNMENTS I/O INSTRUCTIONS

PSW Bit	Assignment	Mnemonic
12	Device busy	(BSY)
13	Examine status	(EX)
14	End of medium	(EOM)
15	Device unavailable	(DU)

The Device Busy condition indicates that the device is not available or ready for transfer of data.

An Examine Status condition indicates that the leftmost 4-bits of the device status byte must be tested to fully determine the device condition.

If, after a Sense Status or Acknowledge Interrupt instruction, the examine bit of the condition code is set, and the leftmost 4 bits of the status byte are zero, an improper device response has occurred or a power down is in process. The data transfer is aborted and the device is released. If the examine bit is set after a Read or Write, or Output Command Instruction, an improper device response has occurred or a power down is in process. A Sense Status instruction should be executed and the leftmost 4 bits of the status byte tested to determine the nature of the failure.

The End Of Medium condition is caused by the presence of a code or indicator at the end of a punched card, or paper or magnetic tape.

The Device Unavailable condition indicates that the device is mechanically unable to transfer data.

1.9.6 Direct Memory Access Channel

The optional Direct Memory Access Channel provides high speed data transfer between core memory and a single external device. Data is transferred 16 bits in parallel at up to the cycle rate of the memory.

The DMAC operates on a cycle stealing basis; that is, when the channel is ready to transfer data, a memory service request is generated causing the memory to service the DMAC at the conclusion of its present cycle. The transfer takes place autonomously, the Processor having no awareness of the transfer, and with no apparent interruption to normal processing.

1.9.7 Selector Channel

The optional Selector Channel provides GE-PAC 30 Digital Systems with the capability for block data transfer between an I/O device and memory. Once initiated, the transfer is performed automatically by the Selector Channel. No further control by the Processor is required. The Processor initiates the transfer by specifying the device address, whether to read or write, the starting address in memory, and the final address in memory. The Processor is then free to continue with its program while the Selector Channel completes the transfer. When the transfer is completed successfully, or terminated due to a fault, the Processor is notified via an interrupt.

1.10 REGISTER SAVE POINTER (GE-PAC 30-2 only)

The halfword in core location X'0022' contains the starting address of the save area in core for storing the fixed-point General Registers. When the Processor is initialized - either manually or due to a power failure - the General Registers are automatically stored in this save area. For example, if the Pointer contains the address X'1FE0', R0 is stored in location X'1FE0', R1 is stored in location X'1FE2',...R14 is stored in location X'1FFC', and R15 is stored in location X'1FFE'. When power is restored or the initialize sequence terminates, the General Registers are fetched from the save area and loaded into the hardware registers. The address placed in the Register Save Pointer should be selected so as not to overwrite current resident programs, or the dedicated core area. On machines without the floating-point option, it is convenient to store the fixed-point registers in the area normally used by the floating-point registers. This is done by setting the halfword at location X'22' to zero. Halfword core location X'0024' is used to save the current PSW Status and Condition Code during initialize sequences. Location X'0026' is used to save the Current PSW Location Counter during initialize sequences.

1.11 CORE MEMORY ALLOCATION

The micro-program uses the first 80 bytes of core memory. TABLE 1-10 shows the allocation of core memory for the GE-PAC 30-01. TABLE 1-11 shows the allocation for the GE-PAC 30-02.

TABLE 1-10
GE-PAC 30-1
CORE MEMORY ALLOCATION FOR
GENERAL REGISTERS AND PROGRAM STATUS WORDS

Hexadecimal Memory Address	Register Assignment
General Registers	
00 - 01	R0
02 - 03	R1
04 - 05	R2
06 - 07	R3
08 - 09	R4
0A - 0B	R5
0C - 0D	R6
0E - 0F	R7
10 - 11	R8
12 - 13	R9
14 - 15	R10
16 - 17	R11
18 - 19	R12
1A - 1B	R13
1C - 1D	R14
1E - 1F	R15
Micro-Processor Registers	
20 - 21	Instruction Register
22 - 23	Instruction Address Register
24 - 25	Current PSW: Status and Condition Code
26 - 27	Current PSW: Instruction Address Counter
28 - 2F	Reserved for Micro-Processor
Program Status Words	
30 - 33	Old PSW: Illegal Instruction Interrupt
34 - 37	New PSW: Illegal Instruction Interrupt
38 - 3B	Old PSW: Machine Malfunction Interrupt
3C - 3F	New PSW: Machine Malfunction Interrupt
40 - 43	Old PSW: External Device Interrupt
44 - 47	New PSW: External Device Interrupt
48 - 4B	Old PSW: Divide Fault Interrupt
4C - 4F	New PSW: Divide Fault Interrupt
50	First user available memory location

TABLE 1-11
GE-PAC 30-2
CORE MEMORY ALLOCATION FOR REGISTERS
AND PROGRAM STATUS WORDS

Hexadecimal Memory Address	Register Assignment
Floating-Point Registers	
00 - 03	R0
04 - 07	R2
08 - 0B	R4
0C - 0F	R6
10 - 13	R8
14 - 17	R10
18 - 1B	R12
1C - 1F	R14
Micro-Processor Registers	
20 - 21	High Speed Interrupt Pointer
22 - 23	Register Save Pointer
24 - 25	Current PSW: Saved Status and CC
26 - 27	Current PSW: Saved Location Counter
Program Status Words	
28 - 2B	Old PSW Flp Divide Fault Interrupt
2C - 2F	New PSW Flp Divide Fault Interrupt
30 - 33	Old PSW Illegal Instruction Interrupt
34 - 37	New PSW Illegal Instruction Interrupt
38 - 3B	Old PSW Machine Malfunction Interrupt
3C - 3F	New PSW Machine Malfunction Interrupt
40 - 43	Old PSW External Device Interrupt
44 - 47	New PSW External Device Interrupt
48 - 4B	Old PSW Fix Divide Fault Interrupt
4C - 4F	New PSW Fix Divide Fault Interrupt
50	First User Available Memory Location

PSW STATUS FIELD ASSIGNMENTS

Bit Set	Meaning
0	Wait State
1	External Device Interrupt
2	Machine Malfunction Interrupt
3	Fixed-Point Divide Fault Interrupt
4	Reserved
5	Floating-Point Divide Fault Interrupt
6 through 11	Unassigned

CHAPTER 2

INSTRUCTION REPERTOIRE

2.1 INTRODUCTION

The instruction repertoire has been grouped by function in this Chapter. The use and operation of each instruction is presented in the following format:

1. The name of the instruction.
2. Instruction word chart for every format the instruction uses, including: mnemonic operation code, and first and second operand designations in the correct assembler format. The format type is designated by [RR], [RX], or [RS]. An instruction diagram with hexadecimal operation code and the locations of all fields is also provided.
3. A description of instruction operation.
4. A diagrammatic representation of instruction operation.
5. A chart illustrating the possible variations of the condition code in the Current Program Status Word as a result of performing the instruction. A 1 indicates set, a zero indicates reset. It is important to note that any instruction which changes the condition code can change all four bits. The conditions listed on the chart are only those conditions which are meaningful after a particular instruction. Other bits may be changed, but their condition is not meaningful.
6. A programming note to provide additional pertinent or clarifying information.

1. **ADD HALFWORD**

2.

AHR R1, R2 [RR]

0 7	8 11	12 15
ØA	R1	R2

AH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
4A	R1	X2	A

AHI R1, A(X2) [RS]

0 7	8 11	12 15	16 31
CA	R1	X2	A

3. The 16-bit second operand is algebraically added to the General Register specified by R1. The resulting sum is contained in R1, the second operand is unchanged.

4.

(R1) ⟵ (R1) + (R2) [RR]

(R1) ⟵ (R1) +[A + (X2)] [RX]

(R1) ⟵ (R1) + A + (X2) [RS]

RESULTING CONDITION CODE:

5.

12	13	14	15	
C	V	G	L	
		0	0	Sum is zero.
		0	1	Sum is less than zero.
		1	0	Sum is greater than zero.
	1			Arithmetic overflow.
1				Carry

PROGRAMMING NOTE

6. The ADD HALFWORD IMMEDIATE (AHI) instruction produces a value which is the algebraic sum of the address field itself plus the content of a General Register index (X2), plus the first operand General Register (R1).

The symbols and abbreviations used in the instruction diagrams are defined as follows:

( ) [ ]	Parentheses or Brackets. Read as "the content of ...".
⟵	Arrow. Read as "is replaced by ..." or "replaces ...".
A	The 16-bit halfword address which is a part of the RX and RS instructions.
R1	The register address designated as the first operand.
R2	The register address designated as the second operand of an RR instruction.
X2	The address of a General Register the content of which is used as an index value.
M1	Mask of 4 bits specifying Branch on Condition testing.
(0:7) (8:15) (16:31)	A bit grouping within a byte, a halfword, or a fullword. Read as "0 thru 7 inclusive", "bits 8 thru 15 inclusive", etc.
PSW	Program Status Word of 32 bits containing the Status, Condition Code, and current instruction address.
CC	Condition Code of 4 bits contained in the PSW.
C	Carry Bit contained in the condition code (bit 12 of PSW).
V	Overflow Bit contained in the condition code (bit 13 of PSW).
G	Greater Than bit contained in the condition code (bit 14 of PSW).
L	Less Than bit contained in the condition code (bit 15 of PSW).
+ - * /	Arithmetic operations - Add, Subtract, Multiply, and Divide respectively.
:	Logical comparison

2.2 LOAD AND STORE INSTRUCTIONS

The load and store instructions transfer information between core memory and the General Registers or the Program Status Word. Load and store operations are performed on 8-bit bytes, 16-bit halfwords, or 32-bit fullwords.

2.2.1 Load Halfword

LHR R1, R2 [RR]

0 7	8 11	12 15
Ø8	R1	R2

LH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
48	R1	X2	A

LHI R1, A(X2) [RS]

0 7	8 11	12 15	16 31
C8	R1	X2	A

The 16-bit second operand is loaded into the General Register specified by R1. The second operand is unchanged.

$(R1) \longleftarrow (R2)$ [RR]

$(R1) \longleftarrow [A + (X2)]$ [RX]

$(R1) \longleftarrow A + (X2)$ [RS]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Operand is zero.
		0	1	Operand is less than zero.
		1	0	Operand is greater than zero.

Programming Note:

The LOAD HALFWORD IMMEDIATE (LHI) instruction produces a value which is the algebraic sum of the value of the address field itself and the content of a General Register index (X2).

2.2.2 Store Halfword

STH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
4Ø	R1	X2	A

The 16-bit first operand is stored in the core memory location specified by the second operand. The first operand is unchanged.

$(R1) \longrightarrow [A + (X2)]$ [RX]

Resulting Condition Code:

Unchanged.

2.2.3 Load Byte

LBR R1, R2 [RR]

0 7	8 11	12 15
93	R1	R2

LB R1, A(X2) [RX]

0 7	8 11	12 15	16 31
D3	R1	X2	A

The 8-bit second operand is loaded into the rightmost (least significant) 8 bits of the General Register specified by R1. The left-most (most significant) 8 bits of R1 are set to zero. The second operand is unchanged.

[R1 (8:15)] ← [R2 (8:15)] [RR]

[R1 (0:7)] ← Zero

[R1 (8:15)] ← [A + (X2)] [RX]

[R1 (0:7)] ← Zero

Resulting Condition Code:

Unchanged.

2.2.4 Store Byte

STBR R1, R2 [RR]

0 7	8 11	12 15
92	R1	R2

STB R1, A(X2) [RX]

0 7	8 11	12 15	16 31
D2	R1	X2	A

The rightmost (least significant) 8-bit byte of the first operand is stored in the General Register or core memory location specified by the second operand. The first operand is unchanged.

[R1 (8:15)] → [R2 (8:15)] [RR]

[R1 (8:15)] → [A + (X2)] [RX]

Resulting Condition Code:

Unchanged.

Programming Note:

In the register-to-register (RR) form of this instruction the leftmost byte, R2(0:7), is unchanged.

2.2.5 Load Multiple

LM R1, A(X2) [RX]

0 7	8 11	12 15	16 31
D1	R1	X2	A

Sequential halfwords from memory are loaded into successive General Registers, beginning with the General Register specified by the R1 field. The first halfword is defined by[A+(X2)]. The operation is terminated when R15 is loaded from memory.

Note that any number of sequential General Registers can be loaded in this manner.

```
1.  $(R1) \longleftarrow [A + (X2)]$
2.  R1: X'F'
      if R1 = X'F', the instruction is
        finished
      if R1 $\neq$ X'F', then:
3.  $R1 + 1 \longrightarrow R1$
4.  $A + 2 \longleftarrow A$, return to equation 1
```

Resulting Condition Code:

Unchanged.

2.2.6 Store Multiple

STM R1, A(X2) [RX]

0 7	8 11	12 15	16 31
DO	R1	X2	A

Successive General Registers are stored sequentially into memory, beginning with the General Register specified by the R1 field. The first storage address is determined by[A + (X2)] . The operation is terminated when R15 is stored in memory. Note that any number of sequential General Registers can be transferred in this manner.

```
1.  $(R1) \longrightarrow [A + (X2)]$
2.  R1: X'F'
      if R1 = X'F', then instruction is
        finished
      if R1 $\neq$ X'F', then:
3.  $R1 + 1 \longrightarrow R1$
4.  $A + 2 \longrightarrow A$, return to equation 1
```

Resulting Condition Code:

Unchanged.

2.2.7 Floating-Point Load

LER R1, R2 [RR]

0 7	8 11	12 15
28	R1	R2

LE R1, A(X2) [RX]

0 7	8 11	12 15	16 31
68	R1	X2	A

The floating-point second operand is normalized and placed in the floating-point register specified as the first operand. During normalization, the fraction is shifted left hexadecimally (4 bits at a time) until the most significant hexadecimal digit is not zero. The exponent is decremented by one for each hexadecimal shift required. Zeros are shifted into the least significant hexadecimal digit of the fraction. The second operand is unchanged.

If the normalization causes exponent underflow, the entire floating-point result is set to zero and the overflow flag is set.

$(R1) \longleftarrow (R2)$ (RR)

$(R1) \longleftarrow [A + (X2)]$ (RX)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Zero
		0	1	Less than zero.
		1	0	Greater than zero.
	1			Exponent underflow.

2.2.8 Floating Point Store

STE R1, A(X2) [RX]

0 7	8 11	12 15	16 31
60	R1	X2	A

The floating-point first operand is placed in the core memory location specified by A + (X2). The first operand is unchanged.

$(R1) \longrightarrow [A + (X2)]$ (RX)

Resulting Condition Code:

Unchanged.

2.2.9 Load Program Status Word

LPSW A(X2) [RX]

0 7	8 11	12 15	16 31
C2	▨	X2	A

A 32-bit operand is loaded into the Current Program Status Word. The operand is unchanged.

[PSW (0:31)] ⟵ [A + (X2)] [RX]

Resulting Condition Code:

Determined by PSW loaded by the instruction.

2.2.10 Autoload

AL A(X2) [RX]

0 7	8 11	12 15	16 31
D5	R1	X2	A

The AUTOLOAD instruction loads memory with a block of data from a byte oriented input device (e.g. teletype, photo-electric paper tape reader, magnetic tape, etc.). The data is read a byte at a time and stored in successive memory locations starting with location X'80'. The last byte is loaded into the memory location specified by the address of the second operand, A + (X2). Any blank or zero bytes that are input prior to the first non zero byte are considered to be leader and are therefore ignored; all other zero bytes are stored as data. The input device is specified by memory location X'78'. The device command code is specified by memory location X'79', this is the normal binary input device specification.

1. (X'80') ⟵ byte #1
2. (X'7F' + n) ⟵ byte #n
3. if A + (X2) < X'80' + n, instruction is finished, otherwise return to equation 2.

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	0	0	0	Data transfer completed correctly.
1				Device Busy (BSY)
	1			Examine Status (EX)
		1		End of Medium (EOM)
			1	Device Unavailable (DU)

2.3 FIXED POINT ARITHMETIC INSTRUCTIONS

The Fixed Point Arithmetic instructions provide for addition, subtraction, multiplication and division of halfword operands. Multiple precision arithmetic operations are performed by the add/subtract with carry halfword instructions.

2.3.1 Add Halfword

AHR R1, R2 [RR]

0 7	8 11	12 15
ØA	R1	R2

AH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
4A	R1	X2	A

AHI R1, A(X2) [RS]

0 7	8 11	12 15	16 31
CA	R1	X2	A

The 16-bit secona operand is algebraically added to the General Register specified by R1. The resulting sum is contained in R1, the second operand is unchanged.

$(R1) \longleftarrow (R1) + (R2)$ [RR]

$(R1) \longleftarrow (R1) + [A + (X2)]$ [RX]

$(R1) \longleftarrow (R1) + A + (X2)$ [RS]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Sum is zero.
		0	1	Sum is less than zero.
		1	0	Sum is greater than zero.
	1			Arithmetic overflow.
1				Carry

Programming Note:

The ADD HALFWORD IMMEDIATE (AHI) instruction produces a value which is the algebraic sum of the address field itself plus the content of a General Register index (X2), plus the first operand General Register (R1).

2.3.2 Add With Carry Halfword

ACHR R1, R2 [RR]

0 7	8 11	12 15
ØE	R1	R2

ACH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
4E	R1	X2	A

The 16-bit second operand and the carry bit of the condition code are algebraically added to the General Register specified by R1. The resulting sum is contained in R1, the second operand is unchanged.

$(R1) \longleftarrow (R1) + (R2) + C$ [RR]

$(R1) \longleftarrow (R1) + [A + (X2)] + C$ [RX]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Sum is zero.
		0	1	Sum is less than zero.
		1	0	Sum is greater than zero.
	1			Arithmetic overflow.
1				Carry

Programming Note:

Multiple precision addition operations require a carry forward from the least significant operands to the most significant. To accomplish this, the locations containing the least significant portions of the two operands are summed using the Add Halfword instruction. A carry forward, if it occurs, is retained in the carry bit position of the condition code (PSW 12).

The locations containing the next least significant portions of the two operands are then summed using the Add With Carry Halfword instruction. The carry bit contained in the condition code (set from the previous addition) participates in this sum; the carry bit position is then set to reflect the new result.

The Add With Carry Halfword instruction is used on succeeding pairs of operands until the most significant operands of the multiple precision words have been summed. The resulting condition code is valid for testing the multiple precision word.

2.3.3 Subtract Halfword

SHR R1, R2 [RR]

0 7	8 11	12 15
ØB	R1	R2

SH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
4B	R1	X2	A

SHI R1, A(X2) [RS]

0 7	8 11	12 15	16 31
CB	R1	X2	A

The 16-bit second operand is subtracted from the General Register specified by R1. The difference is contained in R1, the second operand is unchanged.

$(R1) \longleftarrow (R1) - (R2)$ [RR]

$(R1) \longleftarrow (R1) - [A + (X2)]$ [RX]

$(R1) \longleftarrow (R1) - A + (X2)$ [RS]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Difference is zero.
		0	1	Difference is less than zero.
		1	0	Difference is greater than zero.
	1			Arithmetic overflow
1				Borrow

Programming Note:

The SUBTRACT HALFWORD IMMEDIATE (SHI) instruction produces a value which is the difference between the first operand General Register (R1) less the sum of the address field itself and the content of a General Register index (X2).

2.3.4 Subtract With Carry Halfword

SCHR R1, R2 [RR]

0 7	8 11	12 15
ØF	R1	R2

SCH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
4F	R1	X2	A

The 16-bit second operand with the carry (borrow) bit is subtracted from the General Register specified by R1. The difference is contained in R1, the second operand is unchanged.

$(R1) \longleftarrow (R1) - (R2) - C$ [RR]

$(R1) \longleftarrow (R1) - [A + (X2)] - C$ [RX]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Difference is zero.
		0	1	Difference is less than zero.
		1	0	Difference is greater than zero.
	1			Arithmetic overflow.
1				Borrow

Programming Note:

See Add with Carry Halfword.

2.3.5 Multiply Halfword

MHR R1, R2 [RR]

0 7	8 11	12 15
ØC	R1	R2

MH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
4C	R1	X2	A

The 16-bit second operand is multiplied by the contents of the General Register specified by R1 + 1. The first operand, the contents of the General Register specified by R1, must specify an even numbered register. The resulting 32-bit product is contained in R1 and R1 + 1, an even-odd pair; the second operand is unchanged. The sign of the product is determined by the rules of algebra.

(R1, R1 + 1) $\longleftarrow$ (R1 + 1)*(R2) [RR]

(R1, R1 + 1) $\longleftarrow$ (R1 + 1)*[A + (X2)] [RX]

Resulting Condition Code:

Unchanged.

Programming Note:

After multiplication, the most significant 15 bits with sign are contained in R1. The least significant 16 bits are contained in R1 + 1.

2.3.6 Divide Halfword

DHR R1, R2 [RR]

0 7	8 11	12 15
ØD	R1	R2

DH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
4D	R1	X2	A

The 16-bit second operand is divided into the 32-bit dividend contained in the General Register specified by R1 and R1 + 1. The first operand, R1, must specify an even numbered register. The resulting 15-bit quotient with sign is contained in R1 + 1; a 15-bit remainder with sign is contained in R1, the second operand is unchanged. The sign of the result is determined by the rules of algebra; the sign of the remainder is the same as the sign of the dividend.

(R1 + 1) $\longleftarrow$ (R1, R1 + 1)/(R2) [RR]

(R1) $\longleftarrow$ Remainder

(R1 + 1) $\longleftarrow$ (R1, R1 + 1)/[A + (X2)] [RX]

(R1) $\longleftarrow$ Remainder

Resulting Condition Code:

Unchanged.

Programming Note:

A quotient which cannot be expressed in 16 bits will cause a Divide Fault interrupt if enabled by bit 3 of the Program Status Word. The operands will remain unchanged.

2.4 FLOATING-POINT ARITHMETIC INSTRUCTIONS

The Floating-Point Arithmetic instructions provide for addition, subtraction, multiplication, and division of floating-point operands. These instructions are normally used to perform calculations on operands with a wide range of magnitude, and yield results which are scaled to preserve precision.

NOTE

Floating-Point Registers

The eight Floating-Point Registers are used as accumulators in floating-point arithmetic operations. Each register is thirty-two bits, or one fullword, long. Bit 0 is the sign bit of the fraction, bits 1 through 7 are the exponent of the fraction, and bits 8 through 31 contain a fraction expressed and manipulated in hexadecimal.

The Floating-Point Registers are assigned consecutive byte addresses beginning at address X'0000'. The address in memory is equal to twice the register number (only even number addresses are permitted). For example, Floating-Point Register 6 is maintained in the fullword core location at address X'000C'.

2.4.1 Floating-Point Add

AER R1, R2 [RR]

0 7	8 11	12 15
2A	R1	R2

AE R1, A(X2) [RX]

0 7	8 11	12 15	16 31
6A	R1	X2	A

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is right shifted hexadecimally (4 bits at a time) and its exponent is incremented by one for each hexadecimal shift until the two exponents agree. The fractions are then algebraically added and if a carry results, the exponent of the sum is incremented by one and the fraction (result) is shifted right one hexadecimal position (4 bits). The carry is shifted into the most significant hexadecimal digit of the fraction. If an exponent overflow results, the exponent and fraction of the result are set to all ones and the Overflow flag is set. The sign of the result is not affected by the overflow.

If no carry results from the addition of fractions, the sum is normalized. During normalization, the fraction is shifted left hexadecimally (4 bits at a time) until the most significant hexadecimal digit is <u>not</u> zero. The exponent is decremented by one for each hexadecimal shift required. Zeros are shifted into the least significant hexadecimal digit of the fraction.

If the normalization causes exponent underflow, the sign, exponent and fraction of the sum are set to zero and the Overflow flag is set. If a zero sum is generated from adding two equal magnitudes with unlike signs, the entire floating-point result is zeroed.

$(R1) \longleftarrow (R1) + (R2)$ (RR)

$(R1) \longleftarrow (R1) + [A + (X2)]$ (RX)

2.4.1 Floating-Point Add (Continued)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Sum is zero.
		0	1	Sum is less than zero.
		1	0	Sum is greater than zero.
	1			Exponent overflow or underflow.

2.4.2 Floating-Point Subtract

SER R1, R2 [RR]

0 7	8 11	12 15
2B	R1	R2

SE R, A(X2) [RX]

0 7	8 11	12 15	16 31
6B	R1	X2	A

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is right shifted hexadecimally (4 bits at a time) and its exponent is incremented by one for each hexadecimal shift until the two exponents agree. The fractions are then algebraically subtracted. If a carry results, the exponent of the difference is incremented by one and the fraction (result) is shifted right one hexadecimal position (4 bits). The carry is shifted into the most significant hexadecimal digit of the fraction. If an exponent overflow occurs, the exponent and fraction of the result are set to all ones and the Overflow flag is set. The sign of the result is not affected by the overflow.

If no carry results from the subtraction of fractions, the difference is normalized by shifting the fraction left hexadecimally (4 bits at a time) until the most significant hexadecimal digit is not zero. The exponent is decremented by one for each hexadecimal shift required. Zeros are shifted into the least significant hexadecimal digit of the fraction.

If the normalization causes exponent underflow, the entire floating-point result is set to zero and the Overflow flag is set.

$$(R1) \longleftarrow (R1) - (R2) \qquad (RR)$$

$$(R1) \longleftarrow (R1) - [A + (X2)] \qquad (RX)$$

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Difference is zero.
		0	1	Difference is less than zero.
		1	0	Difference is greater than zero.
	1			Exponent overflow or underflow.

2.4.3 Floating-Point Multiply

MER R1, R2 [RR]

0 7	8 11	12 15
2C	R1	R2

ME R1, A(X2) [RX]

0 7	8 11	12 15	16 31
6C	R1	X2	A

The exponents of the two operands are added to produce the exponent of the result. The resultant exponent is readjusted to excess 64 notation. If an exponent overflow occurs, the exponent and fraction of the product are set to ones and the Overflow flag is set. The sign of the product is determined by the rules of algebra. If an exponent underflow occurs, the entire floating-point result is set to zero and the Overflow flag is set.

If an exponent overflow or underflow does not occur, the multiplication takes place. If the product is zero, the entire floating-point result is zero. If the result is not zero, normalization may occur. During normalization, the fraction is shifted left hexadecimally (4 bits at a time) until the most significant hexadecimal digit is not zero. The exponent of the result is decremented by one for each hexadecimal shift required. After normalization, the product is rounded to 24 bits.

If normalization causes the exponent to underflow, the entire floating point result is set to zero and the Overflow flag is set.

$(R1) \longleftarrow (R1)*(R2)$ (RR)

$(R1) \longleftarrow (R1)*[A + (X2)]$ (RX)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Product is zero.
		0	1	Product is less than zero.
		1	0	Product is greater than zero.
	1			Exponent overflow or underflow.

2.4.4 Floating-Point Divide

DER R1, R2 [RR]

0 7	8 11	12 15
2D	R1	R2

DE R1, A(X2) [RX]

0 7	8 11	12 15	16 31
6D	R1	X2	A

The exponents of the two operands are subtracted to produce the exponent of the result. The resultant exponent is readjusted to excess 64 notation. If an exponent overflow occurs, the exponent and fraction of the quotient are set to all ones and the Overflow flag is set. The sign of the quotient is determined by the rules of algebra. If an exponent underflow occurs, the entire floating-point result is set to zero and the Overflow flag is set. If the divisor (the second operand) is zero, a floating-point divide fault interrupt is caused if enabled by bit 5 of the Program Status Word, and the operands are unchanged.

If the exponent overflow or underflow does not occur, and if the divisor is not zero, the second operand is divided into the first operand . Division continues until the quotient is normalized, adjusting the exponent for each additional division required. If an exponent underflow occurs, the entire floating-point result is set to zero and the Overflow flag is set.

No remainder is returned to the user. The quotient is rounded to compensate for the loss of the remainder.

$(R1) \longleftarrow (R1)/(R2)$ (RR)

$(R1) \longleftarrow (R1)/[A + (X2)]$ (RX)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Quotient is zero.
		0	1	Quotient is less than zero
		1	0	Quotient is greater than zero
	1			Exponent overflow or underflow

2.5 LOGICAL INSTRUCTIONS

The Logical instructions operate bit by bit on the first operand and its corresponding bit in the second operand. These operations provide for masking selected portions of a halfword, or comparison for relative magnitude.

2.5.1 AND Halfword

NHR R1, R2 [RR]

0 7	8 11	12 15
Ø4	R1	R2

NH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
44	R1	X2	A

NHI R1, A(X2) [RS]

0 7	8 11	12 15	16 31
C4	R1	X2	A

The logical product of the 16-bit second operand and the content of the General Register specified by R1 replaces the content of R1. The 16-bit product is formed on a bit-by-bit basis.

(R1) $\longleftarrow$ (R1) AND (R2) [RR]

(R1) $\longleftarrow$ (R1) AND [A + (X2)] [RX]

(R1) $\longleftarrow$ (R1) AND A + (X2) [RS]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Logical product is zero.
		0	1	Logical product is not zero.
		1	0	

Programming Note:

The AND HALFWORD IMMEDIATE (NHI) instruction produces a value which is the logical product of the address field itself plus the content of a General Register index (X2) with the first operand General Register (R1).

The truth table for the AND function is:

0 AND 0 = 0

0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

2.5.2 OR Halfword

OHR R1, R2 [RR]

0 7	8 11	12 15
Ø6	R1	R2

OH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
46	R1	X2	A

OHI R1, A(X2) [RS]

0 7	8 11	12 15	16 31
C6	R1	X2	A

The logical sum of the 16-bit second operand and the content of the General Register specified by R1 replaces the content of R1. The 16-bit sum is formed on a bit-by-bit basis.

(R1) ← (R1) OR (R2) [RR]

(R1) ← (R1) OR [A + (X2)] [RX]

(R1) ← (R1) OR A + (X2) [RS]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Logical sum is zero.
		0	1	} Logical sum is not zero.
		1	0	

Programming Note:

The OR HALFWORD IMMEDIATE (OHI) instruction produces a value which is the logical sum of the address field itself plus the content of a General Register index (X2) with the first operand General Register (R1).

The truth table for the OR function is:

0 OR 0 = 0

0 OR 1 = 1

1 OR 0 = 1

1 OR 1 = 1

2.5.3 Exclusive OR Halfword

XHR R1, R2 [RR]

0 7	8 11	12 15
Ø7	R1	R2

XH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
47	R1	X2	A

XHI R1, A(X2) [RS]

0 7	8 11	12 15	16 31
C7	R1	X2	A

The Logical difference of the 16-bit second operand and the General Register specified by R1 replaces the content of R1. The 16-bit difference is formed on a bit-by-bit basis.

(R1) ← (R1 XOR (R2) [RR]

(R1) ← (R1) XOR[A + (X2)] [RX]

(R1) ← (R1) XOR A + (X2) [RS]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Logical difference is zero.
		0	1	} Logical difference is not zero.
		1	0	

Programming Note:

The EXCLUSIVE OR HALFWORD IMMEDIATE (XHI) instruction produces a value which is the logical difference of the address field itself plus the content of the General Register index (X2) with the first operand General Register (R1).

The truth table for the EXCLUSIVE OR function is:

0 XOR 0 = 0

0 XOR 1 = 1

1 XOR 0 = 1

1 XOR 1 = 0

2.5.4 Compare Logical Halfword

CLHR R1, R2 [RR]

0 7	8 11	12 15
Ø5	R1	R2

CLH R1, A(X2) [RX]

0 7	8 11	12 15	16 31
45	R1	X2	A

CLHI R1, A(X2) [RS]

0 7	8 11	12 15	16 31
C5	R1	X2	A

The first operand specified by R1 is compared logically to the 16-bit second operand. The result is indicated by the setting of the condition code (PSW 12:15); both operands remain unchanged.

$(CC) \longleftarrow (R1) : (R2)$ [RR]

$(CC) \longleftarrow (R1) : [A + (X2)]$ [RX]

$(CC) \longleftarrow (R1) : A + (X2)$ [RS]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	First operand equal to second operand.
		0	1	} First operand not equal to second operand.
		1	0	
1				First operand less than second operand.
0				First operand equal to or greater than second operand.

Programming Note:

The logical comparison is performed by subtracting the second operand from the first operand. The result is in the condition code setting, the operands are not modified.

The COMPARE LOGICAL HALFWORD IMMEDIATE (CLHI) instruction produces a value which is the logical comparison of the address field itself plus the content of a General Register index (X2) with the first operand General Register (R1).

2.5.5 Floating-Point Compare

CER R1, R2 [RR]

0 7	8 11	12 15
29	R1	R2

CE R1, A(X2) [RX]

0 7	8 11	12 15	16 31
69	R1	X2	A

The first operand is compared to the second operand. Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. The result is indicated by the setting of the condition code (PSW12:15). Both operands remain unchanged.

$(CC) \longleftarrow (R1):(R2)$ (RR)

$(CC) \longleftarrow (R1):[A + (X2)]$ (RX)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	First operand equals second operand
		0	1	First operand is less than the second operand
		1	0	First operand is greater than the second operand
		0		First operand is less than or equal to the second operand
0				First operand is greater than or equal to the second operand
1				First operand is less than the second operand

2.6 SHIFT INSTRUCTIONS

The Shift instructions provide for arithmetic and logical manipulation of information contained in the General Registers. Bits shifted out of the high or low order end of a General Register are passed through the carry bit position of the condition code (PSW 12). After execution of a shift instruction, the last bit which was shifted out is contained in the carry position.

The number of bit positions shifted is specified by the sum of the value A with the content of the General Register index (X2). Note that the address field of the instruction (A) is not interpreted as a memory location address but as an unsigned integer. The value of A may be form 0 to FFFF.

A shift of zero positions causes the condition code to be set properly with no alteration to the information contained in the General Register.

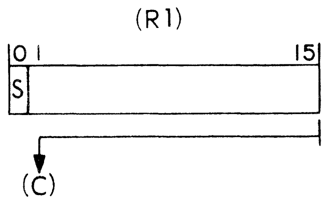
A shift specification of more than 15 bit positions will give meaningful results, since only the four least significant bits of the sum of A plus (X2) are used to specify the number of positions to be shifted.

2.6.1 Shift Left Halfword Arithmetic

SLHA R1, A(X2) [RS]

0 7	8 11	12 15	16 31
CF	R1	X2	A

The content of the first operand (R1) is shifted left the number of bit positions specified by the second operand. Bits 1 through 15 are shifted, the sign bit is unchanged. High order bits shifted out of position 1 are shifted thru the carry bit of the PSW and then lost. Zeros are shifted into position 15.



Resulting Condition Code:

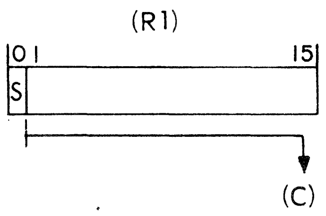
12	13	14	15	
C	V	G	L	
		0	0	Result is zero.
		0	1	Result is less than zero.
		1	0	Result is greater than zero.
0				Last bit that was shifted out was a zero.
1				Last bit that was shifted out was a one.

2.6.2 Shift Right Halfword Arithmetic

SRHA R1, A(X2) [RS]

0 7	8 11	12 15	16 31
CE	R1	X2	A

The content of the first operand (R1) is shifted right the number of bit positions specified by the second operand. Bits 1 through 15 are shifted, the sign bit is unchanged. Low order bits shifted out of position 15 are shifted thru the carry bit of the PSW and then lost. The sign bit is propogated right into position 1.



Resulting Condition Code:

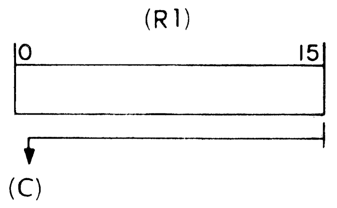
12	13	14	15	
C	V	G	L	
		0	0	Result is zero.
		0	1	Result is less than zero.
		1	0	Result is greater than zero.
0				Last bit that was shifted out was a zero.
1				Last bit that was shiftedout was a one.

2.6.3 Shift Left Halfword Logical

SLHL R1, A(X2) [RS]

0 7	8 11	12 15	16 31
CD	R1	X2	A

The content of the first operand (R1) is shifted left the number of positions specified by the second operand. All 16 bits of the halfword are shifted. High order bits shifted out of position 0 are shifted thru the carry bit of the PSW and then lost. Zeros are shifted into position 15.



Resulting Condition Code:

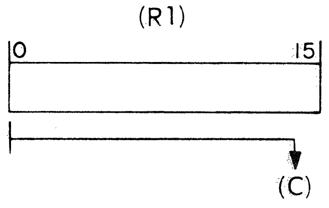
12	13	14	15	
C	V	G	L	
		0	0	Result is zero.
		0	1	Result is less than zero.
		1	0	Result is greater than zero.
0				Last bit that was shifted out was a zero.
1				Last bit that was shifted out was a one.

2.6.4 Shift Right Halfword Logical

SRHL R1, A(X2) [RS]

0 7	8 11	12 15	16 31
CC	R1	X2	A

The content of the first operand (R1) is shifted right the number of bit positions specified by the second operand. All 16 bits of the halfword are shifted. Low order bits shifted out of position 15 are shifted thru the carry bit of the PSW and then lost. Zeros are shifted into position zero.



Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	Result is zero.
		0	1	Result is less than zero.
		1	0	Result is greater than zero.
0				Last bit that was shifted out was a zero.
1				Last bit that was shifted out was a one.

2.7 BRANCH INSTRUCTIONS

Branch instructions are programmed decisions providing entry to subprograms, as well as testing the result of arithmetic logical, or indexing operations.

Many Processor operations result in setting of the Condition Code in the Program Status Word (PSW (12:15)). The Branch on Condition instructions implement the testing of the Condition Code through use of a mask field contained in the instruction itself (M1 field).

The 4-bit M1 field is not a register address, but rather an image of the condition code to be tested.

2.7.1 Branch on True Condition*

BTCR M1, R2 [RR]

0 7	8 11	12 15
Ø2	M1	R2

BTC M1, A(X2) [RX]

0 7	8 11	12 15	16 31
42	M1	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the conditions specified by the mask field (M1). If any of the conditions tested are found to be true, a Branch is executed to the 16-bit address specified by the second operand. If none of the conditions tested are found to be true the next sequential instruction is executed.

Tested Condition True:

[PSW (16:31)] ←—— (R2) [RR]

Tested Condition Not True:

[PSW (16:31)] ←—— [PSW (16:31)] + 2

Tested Condition True:

[PSW (16:31)] ←—— A + (X2) [RX]

Tested Condition Not true:

[PSW (16:31)] ←—— [PSW (16:31)] + 4

Programming Note:

A logical AND is performed between each bit in the condition code and its corresponding bit in the M1 field. If any resultant bit is a one, the branch will occur. The condition code (PSW (12:15)) is not changed.

Example: (Branch occurs)

```
CC    1010
            AND
M1    1100
      ----
      1000   BTC result: Branch if
             bit remains after AND.
```

2.7.2 Branch on False Condition*

BFCR M1, R2 [RR]

0 7	8 11	12 15
Ø3	M1	R2

BFC M1, A(X2) [RX]

0 7	8 11	12 15	16 31
43	M1	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the conditions specified by the mask field (M1). If all conditions tested are found to be false, a Branch is executed to the 16-bit address specified by the second operand. If any of the conditions tested are found to be true, the next sequential instruction is executed.

Tested Condition False:

[PSW (16:31)] ←—— (R2) [RR]

Tested Condition Not false:

[PSW (16:31)] ←—— [(PSW (16:31)] +2

Tested Condition False:

[PSW (16:31)] ←—— A + (X2) [RX]

Tested Condition Not false:

[PSW (16:31)] ←—— [PSW (16:31)] +4

Programming Note:

A logical AND is performed between each bit in the condition code and its corresponding bit in the M1 field. If any resultant bit is a one, the branch will not occur. The condition code (PSW (12:15)) is not changed.

Example: (Branch does not occur)

```
CC    1010
            AND
M1    1100
      ----
      1000   BFC result: Branch
             if no bit remains after
             AND
```

*Refer to Section 2.8 for information on Entended Mnemonic Codes for conditional branch instructions.

2.7.3 Branch Unconditional

BR R2 [RR]

0 7	8 11	12 15
Ø3	Ø	R2

B A(X2) [RX]

0 7	8 11	12 15	16 31
43	Ø	X2	A

The 16-bit address specified by the second operand is transferred to the instruction address field of the Program Status Word (PSW (16:31)). The next instruction executed will be accessed from the location specified by the new instruction address.

$[PSW\ (16{:}31)] \longleftarrow (R2)$ [RR]

$[PSW\ (16{:}31)] \longleftarrow A + (X2)$ [RX]

Programming Note:

The Branch Unconditional instruction is a form of the Branch on False Condition instruction where no condition is specified for testing.

2.7.4 No Operation

NOPR R2 [RR]

0 7	8 11	12 15
Ø2	Ø	R2

NOP A(X2) [RX]

0 7	8 11	12 15	16 31
42	Ø	X2	A

The second operand is ignored and therefore may assume any value. The (M1) field is zero. The instruction address field of the Program Status Word (PSW (16:31)) is incremented and the next sequential instruction is accessed for execution.

$[PSW\ (16{:}31)] \longleftarrow [PSW\ (16{:}31)] + 2$ [RR]

$[PSW\ (16{:}31)] \longleftarrow [PSW\ (16{:}31)] + 4$ [RX]

Programming Note:

The No Operation instruction is a form of the Branch on True Condition instruction where no condition is specified for testing. The No Operation instruction is useful to replace 16 or 32 bits of erroneous or redundant coding or to reserve memory locations within a program for anticipated future coding. This instruction may also be employed as an inactive instruction in timing sequences.

2.7.5 Branch on Index High

BXH R1, A(X2) [RS]

0 7	8 11	12 15	16 31
CØ	R1	X2	A

Prior to execution of this instruction, the General Register specified by the first operand (R1) must contain a 16-bit final address, R1 + 1 must contain a 16-bit decrement value, and R1 + 2 must contain a 16-bit comparand value (limit or starting address). All values may be signed.

Execution of this instruction causes the final address (R1) to be decremented by (R1 + 1) and logically compared to the limit (R1 + 2). As long as the count (R1) is greater than the limit (R1 + 2), the 16-bit address specified by the second operand is transferred to the instruction address field of the Program Status Word (PSW 16:31). The next instruction executed will be accessed from the location specified by the new instruction address.

When the count is not greater than the index limit, the instruction following Branch on Index High will be executed.

$(R1) \longleftarrow (R1) + (R1 + 1)$ [RS]

$(R1) : (R1 + 2)$

if $(R1) > (R1 + 2)$

$[PSW (16:31)] \longleftarrow A + (X2)$

if $(R1) \leq (R1 + 2)$;

$[PSW (16:31)] \longleftarrow [PSW (16:31)] + 4$

Programming Note:

General Register 13 is the maximum specification for the R1 field, since a block of three consecutive General Registers is required.

A logical comparison treats all 16-bits of the halfword as magnitude bits.

2.7.6 Branch on Index Low or Equal

BXLE R1, A(X2) [RS]

0 7	8 11	12 15	16 31
C1	R1	X2	A

Prior to execution of this instruction, the General Register specified by the first operand (R1) must contain a 16-bit count value (starting address), R1 + 1 must contain a 16-bit increment value, and R1 + 2 must contain a 16-bit comparand (limit or final address). All values may be signed.

Execution of this instruction causes the count (R1) to be incremented by (R1 + 1) and logically compared to the index limit. As long as the count (R1) is equal to or less than the limit (R1 + 2), the 16-bit address specified by the second operand is transferred to the instruction address field of the Program Status Word (PSW 16:31). The next instruction executed will be accessed from the location specified by the new instruction address. When the starting address is greater than the limit, the instruction following Branch on Index Low will be executed.

$(R1) \longleftarrow (R1) + (R1 + 1)$ [RS]

$(R1) : (R1 + 2)$

if $(R1) \leq (R1 + 2)$

$[PSW (16:31)] \longleftarrow A + (X2)$

if $(R1) > (R1 + 2)$;

$[PSW (16:31)] \longleftarrow [PSW (16:31)] + 4$

Programming Note:

General Register 13 is the maximum specification for the R1 field since a block of three consecutive General Registers is required.

A logical comparison treats all 16-bits of the halfword as magnitude bits.

2.7.7 Branch and Link

BALR R1, R2 [RR]

0 7	8 11	12 15
Ø1	R1	R2

BAL R1, A(X2) [RX]

0 7	8 11	12 15	16 31
41	P1	X2	A

The Branch and Link instruction is executed in two phases. The instruction address field of the Program Status Word [PSW (16:31)] is incremented and transferred to the General Register specified by the first operand. (R1). Then the second operand is loaded into the instruction address field [PSW (16:31)]. The next instruction executed will be accessed from the location specified by the new instruction address.

(R1) ⟵ [PSW (16:31)] + 2[RR]

[PSW (16:31)] ⟵ (R2)

(R1) ⟵ [PSW (16:31)] + 4[RX]

[PSW (16:31)] ⟵ A + (X2)

Programming Note:

The Branch and Link instruction is required for entry to sub-programs. It differs from the Branch Unconditional instruction in that the current instruction address field is preserved in a specified General Register to be used as the sub-program exit address. Exit from the sub-program is effected by a Branch Unconditional instruction through the General Register in which exit address has been maintained.

2.8 EXTENDED MNEMONIC CODES FOR BRANCH ON CONDITION

To simplify the coding of conditional branch instructions for the programmer, an extended set of mnemonic codes has been provided in the Symbolic Assembler. The most frequently used branch instructions have been provided with mnemonics which are not a part of the machine language instruction set, but are translated by the assembler into the proper operation code and M1 field combinations.

The extended mnemonic codes are for instructions in the RX format.

2.8.1 Branch on Zero

BZ A(X2) [RX]

0 7	8 11	12 15	16 31
43	3	X2	A

The Condition Code field of the Program Status Word [PSW (12:15)] is tested for the zero condition. If this condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Zero: [PSW (16:31)] ← A + (X2) [RX]

CC ≠ Zero: [PSW (16:31)] ← [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
		0	0	Branch
		0	1	No Branch
		1	0	No Branch

Valid After:

LH, LE
AH, ACH, SH, SCH, AE, SE, ME, DE
SLHA, SRHA, SLHL, SRHL
NH, OH, XH

2.8.2 Branch on Not Zero

BNZ A(X2) [RX]

0 7	8 11	12 15	16 31
42	3	X2	A

The Condition Code field of the Program Status Word [PSW (12:15)] is tested for the not zero condition. If this condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC ≠ Zero; [PSW (16:31)] ← A + (X2) [RX]

CC = Zero; [PSW (16:31)] ← [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
		0	0	No branch
		0	1	Branch
		1	0	Branch

Valid After:

LH, LE
AH, ACH, SH, SCH, AE, SE, ME, DE
SLHA, SRHA, SLHL, SRHL
NH, OH, XH

2.8.3 Branch on Plus

BP A(X2) [RX]

0 7	8 11	12 15	16 31
42	2	X2	A

The Condition Code field of the Program Status Word [PSW (12:15)] is tested for the plus condition. If this condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Plus; [PSW (16:31)] ← A+(X2) [RX]

CC ≠ Plus; [PSW (16:31)] ← [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
		0	0	} No branch
		0	1	
		1	0	Branch

Valid After:

LH, LE
AH, ACH, SH, SCH, AE, SE, ME, DE
SLHA, SRHA, SLHL, SRHL
NH, OH, XH

2.8.4 Branch on Not Plus

BNP A(X2) [RX]

0 7	8 11	12 15	16 31
43	2	X2	A

The Condition Code field of the Program Status Word [PSW (12:15)] is tested for the not plus condition. If this condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC ≠ Plus; [PSW (16:31)] ← A+(X2) [RX]

CC = Plus; [PSW (16:31)] ← [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
		0	0	} Branch
		0	1	
		1	0	No Branch

Valid After:

LH, LE
AH, ACH, SH, SCH, AE, SE, ME, DE
SLHA, SRHA, SLHL, SRHL
NH, OH, XH

2.8.5 Branch on Minus

BM A(X2) [RX]

0 7	8 11	12 15	16 31
42	1	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the minus condition. If the condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Minus; [PSW (16:31)] $\leftarrow$ A + (X2) [RX]

CC $\neq$ Minus; [PSW (16:31)] $\leftarrow$ PSW (16:31) + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
		0	0	No branch
		0	1	Branch
		1	0	No branch

Valid After:

LH, LE
AH, ACH, SH, SCH, AE, SE, ME, DE
SLHA, SRHA, SLHL, SRHL
NH, OH, XH

2.8.6 Branch on Not Minus

BNM A(X2) [RX]

0 7	8 11	12 15	16 31
43	1	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the not minus condition. If the condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC $\neq$ Minus; [PSW (16:31)] $\leftarrow$ A + (X2) [RX]

CC = Minus; [PSW (16:31)] $\leftarrow$ [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
		0	0	Branch
		0	1	No branch
		1	0	Branch

Valid After:

LH, LE
AH, ACH, SH, SCH, AE, SE, ME, DE
SLHA, SRHA, SLHL, SRHL
NH, OH, XH

2.8.7 Branch on Carry

BC A(X2) [RX]

0 7	8 11	12 15	16 31
42	8	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the carry condition. If the condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Carry;[PSW (16:31)] ← A + (X2) [RX]

CC ≠ Carry;[PSW (16:31)] ← [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
1				Branch
0				No Branch

Valid After:

AH, ACH, SH, SCH
SLHA, SRHA, SLHL, SRHL

2.8.8 Branch on Overflow

BO A(X2) [RX]

0 7	8 11	12 15	16 31
42	4	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the overflow condition. If the condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Overflow;[PSW (16:31)] ← A + (X2) [RX]

CC ≠ Overflow;[PSW (16:31)] ← [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
	1			Branch
	0			No Branch

Valid After:

AH, ACH, SH, SCH, AE, SE, ME, DE, LE

2.8.9 Branch on Low

BL A(X2) [RX]

0 7	8 11	12 15	16 31
42	8	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the low condition. If the condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Low; [PSW (16:31)] $\leftarrow$ A + (X2) [RX]

CC $\neq$ Low; [PSW (16:31)] $\leftarrow$ [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
1				Branch
0				No branch

Valid After:

CLH, CE

2.8.10 Branch on Not Low

BNL A(X2) [RX]

0 7	8 11	12 15	16 31
43	8	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the not low condition. If the condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Not low; [PSW (16:31)] $\leftarrow$ A + (X2) [RX]

CC $\neq$ Not low; [PSW (16:31)] $\leftarrow$ [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
0				Branch
1				No Branch

Valid After:

CLH, CE

2.8.11 Branch on Equal

BE A(X2) [RX]

0 7	8 11	12 15	16 31
43	3	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the equal condition. If the condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Equal; [PSW (16:31)] $\leftarrow$ A + (X2) [RX]

CC $\neq$ Equal; [PSW (16:31)] $\leftarrow$ [PSW (16:31)] + 4

Condition Code Tested:

12	13	14	15	
C	V	G	L	
		0	0	Branch
		0	1	No Branch
		1	0	

Valid After:

CLH, CE

2.8.12 Branch on Not Equal

BNE A(X2) [RX]

0 7	8 11	12 15	16 31
42	3	X2	A

The condition code field of the Program Status Word [PSW (12:15)] is tested for the not equal condition. If the condition is met, a Branch is executed to the 16-bit address specified by the second operand. If the condition is not met, the next sequential instruction is executed.

CC = Not equal; [PSW (16:31)] $\leftarrow$ A + (X2) [RX]

CC $\neq$ Not equal; [PSW (16:31)] $\leftarrow$ [(PSW(16:31)] +

Condition Code Tested:

12	13	14	15	
C	V	G	L	
		0	0	No Branch
		0	1	Branch
		1	0	

Valid After:

CLH, CE

2.9 DEVICE INTERRUPT AND CONTROL INSTRUCTIONS

The Interrupt and Control instructions provide for Processor interrogation and control of peripheral devices in the system.

2.9.1 Acknowledge Interrupt

AIR R1, R2 [RR]

0 7	8 11	12 15
9F	R1	R2

AI R1, A(X2) [RX]

0 7	8 11	12 15	16 31
DF	R1	X2	A

The address of the interrupting device replaces the content of the 16-bit General Register specified by the first operand (R1). The 8-bit device status byte replaces the content of the location specified by the second operand. The Condition Code is set equal to the right-most four bits of the device status byte. The device interrupt condition is then cleared.

[R1 (8:15)] ← Device address [RR]

[R1 (0:7)] ← Zero

[R2 (8:15)] ← Status byte

[R2 (0:7)] ← Zero

[PSW (12:15)] ← Status byte (4:7)

[R1 (8:15)] ← Device number [RX]

[R1 (0:7)] ← Zero

[A + (X2)] ← Status byte

[PSW (12:15)] ← Status byte (4:7)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
1	0	0	0	Device busy (BSY)
0	1	0	0	Examine status (EX)
0	0	1	0	End of medium (EOM)
0	0	0	1	Device unavailable (DU)

2.9.2 Sense Status

SSR R1, R2 [RR]

0 7	8 11	12 15
9D	R1	R2

SS R1, A(X2) [RX]

0 7	8 11	12 15	16 31
DD	R1	X2	A

The 16-bit General Register specified by the first operand (R1) contains the device address. The device is addressed and the 8-bit device status byte replaces the content of the location specified by the second operand. The Condition Code is set equal to the right-most four bits of the device status byte. The first operand is unchanged.

[R2 (8:15)] ← Status byte [RR]

[R2 (0:7)] ← Zero

[PSW (12:15)] ← Status byte (4:7)

[A + (X2)] ← Status byte [RX]

[PSW (12:15)] ← Status byte (4:7)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
1	0	0	0	Device busy (BSY)
0	1	0	0	Examine Status (EX)
0	0	1	0	End of Medium (EOM)
0	0	0	1	Device unavailable (DU)

2.9.3 Output Command

OCR R1, R2 [RR]

0 7	8 11	12 15
9E	R1	R2

OC R1, A(X2) [RX]

0 7	8 11	12 15	16 31
DE	R1	X2	A

The 16-bit General Register specified by the first operand (R1) contains the device address. The device is addressed and the 8-bit device command byte specified by the second operand is transmitted to the addressed device. Both operands remain unchanged.

Device $\longleftarrow$	[R2 (8:15)]	[RR]
Device $\longleftarrow$	[A + (X2)]	[RX]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	1	0	0	Examine Status (EX)

Programming Note:

The Examine Status bit is set if the device cannot complete the command action.

2.10 INPUT/OUTPUT INSTRUCTIONS

The Input/Output instructions provide for transfer of 8-bit byte information between the Processor and peripheral devices in the system.

2.10.1 Read Data

RDR R1, R2 [RR]

0 7	8 11	12 15
9B	R1	R2

RD R1, A(X2) [RX]

0 7	8 11	12 15	16 31
DB	R1	X2	A

The 16-bit General Register specified by the first operand (R1) contains the device address. The device is addressed and a single 8-bit data byte is transmitted from the device replacing the content of the location specified by the second operand.

[R2 (8:15)] ←—— Data byte [RR]

[R2 (0:7)] ←—— Zero

[A + (X2)] ←—— Data byte [RX]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
	1			Examine Status (EX)

2.10.2 Write Data

WDR R1, R2 [RR]

0 7	8 11	12 15
9A	R1	R2

WD R1, A(X2) [RX]

0 7	8 11	12 15	16 31
DA	R1	X2	A

The 16-bit General Register specified by the first operand (R1) contains the device address. The device is addressed and a single 8-bit data byte is transmitted to the device. Both operands remain unchanged.

[R2 (8:15)] ——→ (Device) [RR]

[A + (X2)] ——→ (Device) [RX]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
	1			Examine Status (EX)

2.10.3 Read Block

RBR R1, R2 [RR]

0 7	8 11	12 15
97	R1	R2

RB R1, A(X2) [RX]

0 7	8 11	12 15	16 31
D7	R1	X2	A

The 16-bit General Register specified by the first operand (R1) contains the device address. The 16-bit second operand location, (R2) or [A + (X2)] contains the starting address of the data buffer to be transferred. The next sequential halfword, (R2 + 1) or [A + (X2) + 2] contains the ending address of the data buffer. The starting address must be equal to, or less than, the ending address. Data transfer is inclusive of the buffer limits.

The READ BLOCK instruction causes transfer of 8-bit data bytes from a device to consecutive memory locations. No other instructions are executed during transfer of the data block. The condition code portion of the Program Status Word [PSW (12:15)] will be set to zero after a normal transfer. In the event of an abnormal block data transfer, the condition code will not be zero.

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	0	0	0	Block data transfer completed correctly.
1				Device busy (BSY)
	1			Examine status (EX)
		1		End of medium (EOM)
			1	Device unavailable (DU)

2.10.4 Write Block

WBR R1, R2 [RR]

0 7	8 11	12 15
96	R1	R2

WB R1, A(X2) [RX]

0 7	8 11	12 15	16 31
D6	R1	X2	A

The 16-bit General Register specified by the first operand (R1) contains the device address. The 16-bit second operand location, (R2) or [A + (X2)] contains the starting address of the data buffer to be transferred. The next sequential halfword (R2 + 1) or [A + (X2) + 2] contains the ending address of the data buffer. The starting address must be equal to, or less than, the ending address. Data transfer is inclusive of the buffer limits.

The WRITE BLOCK instruction causes transfer of 8-bit data bytes from consecutive memory locations to a device. No other instruction are executed during transfer of the data block. The condition code portion of the Program Status Word [PSW (12:15)] will be set to zero after a normal transfer. In the event of an abnormal block data transfer, the condition code will not be zero.

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	0	0	0	Block data transfer completed correctly.
1				Device busy (BSY)
	1			Examine status (EX)
		1		End of medium (EOM)
			1	Device unavailable (DU)

CHAPTER 3
CONSOLE OPERATION AND DISPLAY

3.1 INTRODUCTION

The discussion which follows pertains to a typical Display Panel, shown on Figure 3-1, and the operating controls associated with it. Different models may vary.

The control console is comprised of six distinct elements:

1. Control Switches: POWER, INITIALIZE, and EXECUTE.
2. MODE CONTROL rotary switch.
3. SPEED CONTROL rotary switch.
4. REGISTER DISPLAY rotary switch.
5. Sixteen Data/Address switches.
6. Display of two 16-bit halfword registers.
7. Stripes above the display lights are memory aids for displayed information.

each of the elements is described in the following sections. Console operating procedures are provided following the descriptions.

3.2 CONTROL SWITCHES

The latching POWER switch applies power to the Processor and device controllers. An indicator lamp is associated with the POWER switch.

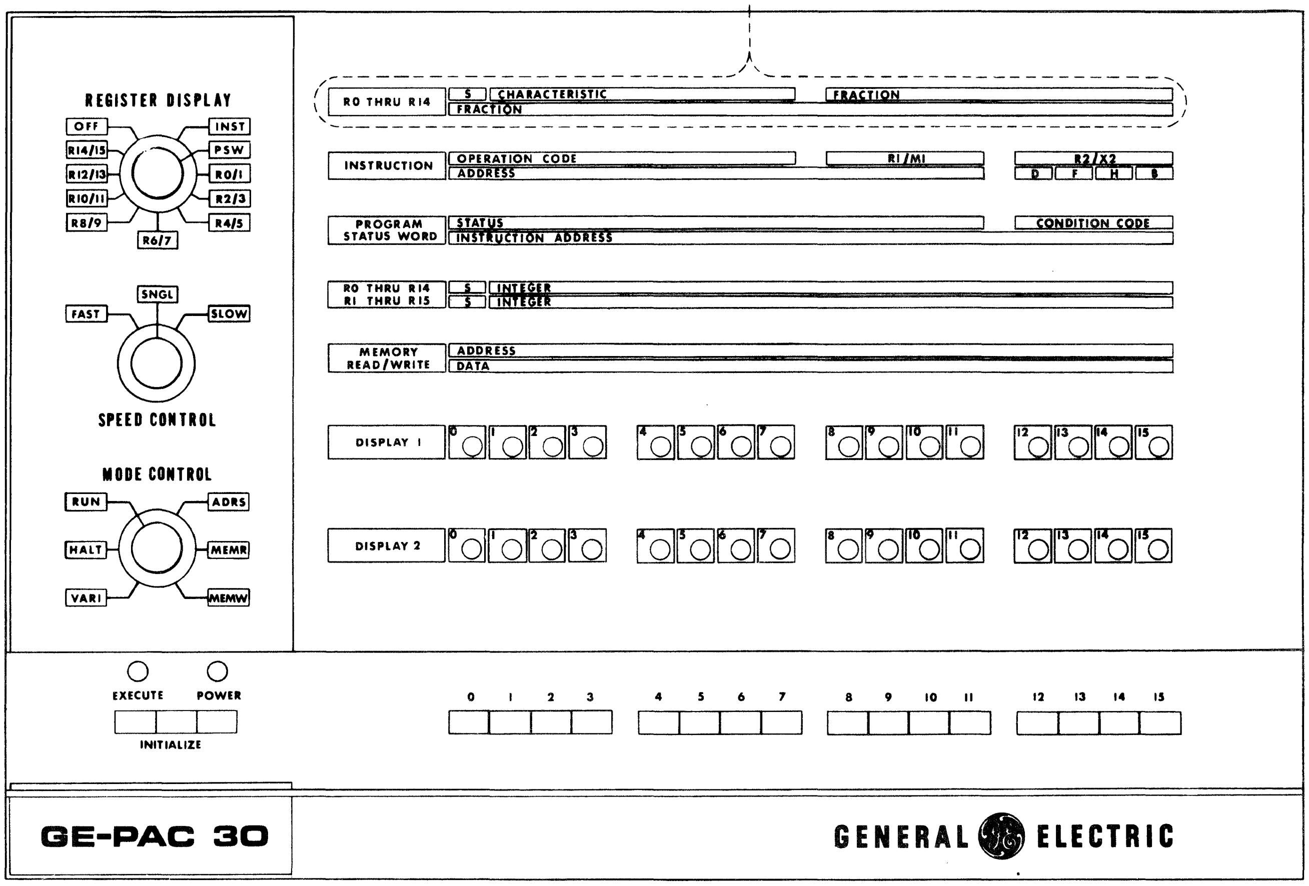


Figure 3-1. Display Panel

The momentary INITIALIZE switch resets peripheral device interrupts and certain other functions in the Processor. After initialization, the Processor is left in the Halt mode.

The momentary EXECUTE switch causes the Processor to perform the function selected by the MODE CONTROL switch. The associated indicator lamp is on when the Processor is in the interruptable Wait state or Halt mode; the lamp is off when the Processor is in the Run mode.

3.3 MODE CONTROL SWITCH

The rotary MODE CONTROL switch selects the following modes of operation which become effective when the EXECUTE switch is depressed:

RUN:	the Processor continuously executes instructions at rated speed.
HALT: (FIX)	instruction execution is stopped at the moment the EXECUTE switch is depressed and the Processor is placed in the Wait state. The register displays are operative in this mode.
HALT: (FLP)	The HALT FLP position is similar to the HALT (or HALT FIX) position except that in Processors equipped with optional floating-point hardware, the selected registers are displayed in the floating-point format.
VARI: (FIX)	the Processor executes instructions at the rate selected by the variable SPEED CONTROL. The register displays are operative in this mode.
VARI: (FLP)	The VARI FLP position is similar to the VARI (or VARI FIX) position except that in Processors equipped with the optional floating-point hardware, the selected registers are displayed in the floating-point format.
ADRS:	selects the instruction location address portion of the Program Status Word (PSW(16:31)). The new address is entered in the sixteen Address Switches below the register display.
MEMR:	the Memory Read mode permits display of memory data in the register display.
MEMW:	the Memory Write mode permits entry of data into memory from the sixteen Data Switches below the register display.

3.4 SPEED CONTROL SWITCH

The variable SPEED CONTROL switch provides a dynamically changing display when in the Variable mode. The rate of display can vary from 1 to 1000 cps by rotating the control clockwise from SLOW to FAST. When in the SNGL position, a single instruction is executed and displayed each time the EXECUTE switch is depressed.

3.5 REGISTER DISPLAY SWITCH

The REGISTER DISPLAY switch selects pairs of 16-bit registers for display in the lighted panel positions labeled DISPLAY 1 and DISPLAY 2. Beginning at the one o'clock position and moving clockwise, the registers displayed are:

INST:	(D1)	The current instruction.
	(D2)	The Address field of the current instruction if RX or RS format.
PSW:	(D1)	The Program Status and Condition Code.
	(D2)	The location address of the current instruction.
RO/1:	(D1)	General Register 0.
	(D2)	General Register 1.
		(Note: the seven succeeding pairs of General Registers are selected similarly.)
OFF:		(D1) and (D2) are blank.

3.6 DATA/ADDRESS SWITCHES

The 16 Input Register latching pushbutton switches provide a means of entering information manually. An address set in the switches is entered into the instruction location address portion of the Program Status Word (PSW (16:31)) when the ADRS mode is selected and the EXECUTE switch is depressed.

Data set in the switches is written into memory when the MEMW mode is selected and the EXECUTE switch is depressed. The halfword location written into is specified by the instruction address portion of the PSW.

3.7 REGISTER DISPLAY

The two 16-bit halfword register displays are operative when the VARIable Mode or when MEMR or MEMW have been selected. The display registers remain static when in the RUN mode.

3.8 CONSOLE OPERATING PROCEDURES

To bring up power and initialize the system:

1. Depress the latching POWER switch.
2. Depress the momentary INITIALIZE switch.

To shut down power to the system:

1. Set the Mode Control switch to HALT.
2. Depress the momentary EXECUTE switch.
3. Release the latching POWER switch.

To begin execution of a program:

The system must be in the Halt mode.

1. Set the Mode Control switch to ADRS.
2. Enter the program starting address in the 16 address switches.
3. Depress the momentary EXECUTE switch.
4. Set the Mode Control switch to RUN.
5. Depress the EXECUTE switch.

To halt execution of a program:

1. Set the Mode Control switch to HALT.
2. Depress the EXECUTE switch.

To read memory from display registers:

The system must be in the Halt mode.

1. Set the Mode Control switch to ADRS.
2. Enter the memory read starting address in the 16 address switches.
3. Depress the EXECUTE switch.
4. Set the Mode Control switch to MEMR.
5. Depress the EXECUTE switch.
6. The memory data is read from display register 2 (D2). The memory address of the data being displayed is in display register 1 (D1).
7. Depress the EXECUTE switch to display memory data from successive memory locations. The memory address is automatically incremented each time the EXECUTE switch is depressed.

To write into memory:

The system must be in the Halt mode.

1. Set the Mode Control switch to ADRS.
2. Enter the memory write starting address in the 16 address switches.
3. Depress the EXECUTE switch.

4. Set the Mode Control switch to MEMW.

5. Enter the data to be written into memory in the 16 data switches.

6. Depress the EXECUTE switch.

7. The memory data entered is displayed in display register 2 (D2). The memory address which was written into is displayed in display register 1 (D1). To write into successive memory locations repeat from Step 5. The memory address is automatically incremented with each depression of the EXECUTE switch.

To display the Instruction Register, Program Status Word or General Registers:

The system must be in the Halt mode.

1. Set the Register Display switch to select the registers desired for display.

2. Depress the EXECUTE switch. The registers selected for display will appear in D1 and D2.

To display registers in the VARIable speed mode:

The system must be in the Halt mode.

1. Set the Mode Control switch to ADRS.

2. Enter the starting memory location address in the 16 address switches.

3. Depress the EXECUTE switch.

4. Set the Mode Control switch to VARI.

5. Set the Speed Control switch to SINGL or to a SLOW - FAST setting.

6. Set the REGISTER DISPLAY switch to select the registers desired for display.

7. Depress the EXECUTE switch to begin operation of the program with display of the selected registers. If SNGL step was selected, the EXECUTE switch is depressed to cause single step execution of successive instructions.

8. The REGISTER DISPLAY switch setting can be changed during operation in the variable speed mode. The SPEED CONTROL switch can also be changed from SNGL to a SLOW-FAST setting without halting operations.

3.9 DISPLAY PANEL PROGRAMMING

The Display Panel may also be accessed by program as a peripheral device. The Data/Address Switches may be read (Byte 0 = Switches 8 through 15, Byte 1 = Switches 0 through 7) and the Display Registers may be loaded as follows: Byte 0 = Display Register 2-bits 8 through 15, Byte 1 = Display Register 2-bits 0 through 7, Byte 2 = Display Register 1-bits 8 through 15, and Byte 3 = Display Register-1 bits 0 through 7. Two modes of operation are available, Normal and Incremental. In Normal mode, Byte 0 is accessed each time the Display is addressed. In incremental mode, the Bytes are accessed successively by each Write Data or Read Data instruction. The status of the MODE CONTROL and REGISTER DISPLAY Switches may be read via a Sense Status instruction. See Appendix 4, page A4-1.

APPENDIX 1
SUMMARY OF INSTRUCTIONS - ALPHABETICAL BY NAME

INSTRUCTION	TYPE	MNEMONIC	OP CODE
Acknowledge Interrupt	RR	AIR	9F
Acknowledge Interrupt	RX	AI	DF
Add Halfword	RR	AHR	0A
Add Halfword	RX	AH	4A
Add Halfword Immediate	RS	AHI	CA
Add with Carry Halfword	RR	ACHR	0E
Add with Carry Halfword	RX	ACH	4E
AND Halfword	RR	NHR	04
AND Halfword	RX	NH	44
AND Halfword Immediate	RS	NHI	C4
Autoload	RX	AL	D5
Branch and Link	RR	BALR	01
Branch and Link	RX	BAL	41
Branch on False Condition	RR	BFCR	03
Branch on False Condition	RX	BFC	43
Branch on True Condition	RR	BTCR	02
Branch on True Condition	RX	BTC	42
Branch on Index Low or Equal	RS	BXLE	C1
Branch on Index High	RS	BXH	C0
Branch Unconditional	RR	BR	03
Branch Unconditional	RX	B	43
Branch on Overflow*	RX	BO	424
Branch on Zero*	RX	BZ	433
Branch on Not Zero*	RX	BNZ	423
Branch on Equal*	RX	BE	433
Branch on Not Equal*	RX	BNE	423

*Extended Mnemonics - See Section 2.8

INSTRUCTION	TYPE	MNEMONIC	OP CODE
Branch on Plus*	RX	BP	422
Branch on Not Plus*	RX	BNP	432
Branch on Low*	RX	BL	428
Branch on Not Low*	RX	BNL	438
Branch on Minus*	RX	BM	421
Branch on Not Minus*	RX	BNM	431
Branch on Carry*	RX	BC	428
Compare Logical Halfword	RR	CLHR	05
Compare Logical Halfword	RX	CLH	45
Compare Logical Halfword Immediate	RS	CLHI	C5
Divide Halfword	RR	DHR	0D
Divide Halfword	RX	DH	4D
Exclusive OR Halfword	RR	XHR	07
Exclusive OR Halfword	RX	XH	47
Exclusive OR Halfword Immediate	RS	XHI	C7
Floating-Point Add	RR	AER	2A
Floating-Point Add	RX	AE	6A
Floating-Point Compare	RR	CER	29
Floating-Point Compare	RX	CE	69
Floating-Point Divide	RR	DER	2D
Floating-Point Divide	RX	DE	6D
Floating-Point Load	RR	LER	28
Floating-Point Load	RX	LE	68
Floating-Point Multiply	RR	MER	2C
Floating-Point Multiply	RX	ME	6C
Floating-Point Store	RX	STE	60
Floating-Point Subtract	RR	SER	2B
Floating-Point Subtract	RX	SE	6B
Load Byte	RR	LBR	93
Load Byte	RX	LB	D3
Load Halfword	RR	LHR	08
Load Halfword	RX	LH	48
Load Halfword Immediate	RS	LHI	C8

*Extended Mnemonic - See Section 2.8

INSTRUCTION	TYPE	MNEMONIC	OP CODE
Load Multiple	RX	LM	D1
Load Program Status Word	RX	LPSW	C2
Multiply Halfword	RR	MHR	OC
Multiply Halfword	RX	MH	4C
No Operation	RR	NOPR	020
No Operation	RX	NOP	420
OR Halfword	RR	OHR	06
OR Halfword	RX	OH	46
OR Halfword Immediate	RS	OHI	C6
Output Command	RR	OCR	9E
Output Command	RX	OC	DE
Read Block	RR	RBR	97
Read Block	RX	RB	D7
Read Data	RR	RDR	9B
Read Data	RX	RD	DB
Shift Left Arithmetic	RS	SLHA	CF
Shift Left Logical	RS	SLHL	CD
Shift Right Arithmetic	RS	SRHA	CE
Shift Right Logical	RS	SRHL	CC
Store Byte	RR	STBR	92
Store Byte	RX	STB	D2
Store Halfword	RX	STH	40
Store Multiple	RX	STM	D0
Subtract Halfword	RR	SHR	0B
Subtract Halfword	RX	SH	4B
Subtract Halfword Immediate	RS	SHI	CB
Subtract with Carry Halfword	RR	SCHR	0F
Subtract with Carry Halfword	RX	SCH	4F
Sense Status	RR	SSR	9D
Sense Status	RX	SS	DD
Write Block	RR	WBR	96
Write Block	RX	WB	D6
Write Data	RR	WDR	9A
Write Data	RX	WD	DA

APPENDIX 2
SUMMARY OF INSTRUCTIONS - NUMERICAL BY OP CODE

OP CODE	TYPE	MNEMONIC	INSTRUCTION
01	RR	BALR	Branch and Link
02	RR	BTCR	Branch on True Condition
03	RR	BFCR	Branch on False Condition
04	RR	NHR	AND Halfword
05	RR	CLHR	Compare Halfword
06	RR	OHR	OR Halfword
07	RR	XHR	Exclusive OR Halfword
08	RR	LHR	Load Halfword
0A	RR	AHR	Add Halfword
0B	RR	SHR	Subtract Halfword
0C	RR	MHR	Multiply Halfword
0D	RR	DHR	Divide Halfword
0E	RR	ACHR	Add with Carry Halfword
0F	RR	SCHR	Subtract with Carry Halfword
28	RR	LER	Floating-Point Load
29	RR	CER	Floating-Point Compare
2A	RR	AER	Floating-Point Add
2B	RR	SER	Floating-Point Subtract
2C	RR	MER	Floating-Point Multiply
2D	RR	DER	Floating-Point Divide
40	RX	STH	Store Halfword
41	RX	BAL	Branch and Link
42	RX	BTC	Branch on True Condition
43	RX	BFC	Branch on False Condition
44	RX	NH	AND Halfword
45	RX	CLH	Compare Logical Halfword
46	RX	OH	OR Halfword
47	RX	XH	Exclusive OR Halfword
48	RX	LH	Load Halfword
4A	RX	AH	Add Halfword
4B	RX	SH	Subtract Halfword
4C	RX	MH	Multiply Halfword
4D	RX	DH	Divide Halfword
4E	RX	ACH	Add with Carry Halfword
4F	RX	SCH	Subtract with Carry Halfword
60	RX	STE	Floating-Point Store

OP CODE	TYPE	MNEMONIC	INSTRUCTION
68	RX	LE	Floating-Point Load
69	RX	CE	Floating-Point Compare
6A	RX	AE	Floating-Point Add
6B	RX	SE	Floating-Point Subtract
6C	RX	ME	Floating-Point Multiply
6D	RX	DE	Floating-Point Divide
92	RR	STBR	Store Byte
93	RR	LBR	Load Byte
96	RR	WBR	Write Block
97	RR	RBR	Read Block
9A	RR	WDR	Write Data
9B	RR	RDR	Read Data
9D	RR	SSR	Sense Status
9E	RR	OCR	Output Command
9F	RR	AIR	Acknowledge Interrupt
C0	RS	BXH	Branch on Index High
C1	RS	BXLE	Branch on Index Low or Equal
C2	RX	LPSW	Load Program Status Word
C4	RS	NHI	AND Halfword Immediate
C5	RS	CLHI	Compare Logical Halfword Immediate
C6	RS	OHI	OR Halfword Immediate
C7	RS	XHI	Exclusive OR Halfword Immediate
C8	RS	LHI	Load Halfword Immediate
CA	RS	AHI	Add Halfword Immediate
CB	RS	SHI	Subtract Halfword Immediate
CC	RS	SRHL	Shift Right Logical
CD	RS	SLHL	Shift Left Logical
CE	RS	SRHA	Shift Right Arithmetic
CF	RS	SLHA	Shift Left Arithmetic
D0	RX	STM	Store Multiple
D1	RX	LM	Load Multiple
D2	RX	STB	Store Byte
D3	RX	LB	Load Byte
D5	RX	AL	Autoload
D6	RX	WB	Write Block
D7	RX	RB	Read Block
DA	RX	WD	Write Data
DB	RX	RD	Read Data
DD	RX	SS	Sense Status
DE	RX	OC	Output Command
DF	RX	AI	Acknowledge Interrupt

APPENDIX 3

ARITHMETIC REFERENCES

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 45
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

TABLE OF POWERS OF SIXTEEN

16^n							n
						1	0
						16	1
						256	2
					4	096	3
					65	536	4
				1	048	576	5
				16	777	216	6
				268	435	456	7
			4	294	967	296	8
			68	719	476	736	9
		1	099	511	627	776	10
		17	592	186	044	416	11
		281	474	976	710	656	12
	4	503	599	627	370	496	13
	72	057	594	037	927	936	14
1	152	921	504	606	846	976	15

Decimal Values

HEXADECIMAL ADDITION TABLE

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	1
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	2
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	3
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	4
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	5
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	6
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	7
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	8
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	9
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	A
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	B
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	C
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	D
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	E
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

HEXADECIMAL MULTIPLICATION TABLE

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	2
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	3
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	4
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	5
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	6
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	7
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	8
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	9
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

APPENDIX 4

INPUT/OUTPUT REFERENCES

DISPLAY STATUS AND COMMAND BYTE DATA
HEX ADDRESS 01

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	MODE				REGISTER DISPLAY			
COMMAND BYTE	NORM	INC						

STATUS:

		0	1	2	3	4	5	6	7
MODE CONTROL SWITCH	VARI (FIX)	0	1	0	0				
	VARI FLT	0	1	1	0				
	RUN	1	0	0	0				
	HALT (FIX)	1	1	0	0				
	HALT FLT	1	1	1	0				
	MEM WRITE	0	0	0	1				
	MEM READ	0	0	1	0				
	ADRS	0	0	1	1				
	OFF					0	0	0	0
REGISTER DISPLAY SWITCH	REG DISPLAY					0	0	0	1
	INST					0	0	1	0
	PSW					0	1	0	0
	R0, R1					1	0	0	0
	R2, R3					1	0	0	1
	R4, R5					1	0	1	0
	R6, R7					1	0	1	1
	R8, R9					1	1	0	0
	R10, R11					1	1	0	1
	R12, R13					1	1	1	0
	R14, R15					1	1	1	1

COMMAND:

NORM	In the Normal Mode, Byte 0 of the registers or switches is accessed each time an I/O operation is directed to the Display Panel.
INC	In the Incremental Mode, subsequent I/O operations access subsequent bytes of the registers or switches.

TELETYPEWRITER STATUS AND COMMAND BYTE DATA HEX ADDRESS 02

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE			BRK		BSY	EX		DU
COMMAND BYTE	DISABLE	ENABLE	UNBLOCK	BLOCK	WRT	READ	PWR ON	PWR OFF

BRK	The Break bit is set when the Break key on the Teletypewriter is depressed, or the Teletypewriter is logically disconnected from the Controller.
BSY	The significance of the Busy bit depends upon whether a Read or a Write operation is in progress. During Write mode, BSY is normally low, and goes high only while data is being received by the device. During Read mode, BSY is normally high, and goes low only when data has been received from th device, but not yet been transferred to the Processor. During Read mode, BSY goes high again as soon as the Processor accepts the data.
EX	The Examine bit is set whenever BRK is set.
DU	The Device Unavailable bit is set whenever the Teletypewriter is in the OFF or LOCAL mode, or power is not connected to the Teletypewriter.
DISABLE	This command disables the Device Interrupt to the Processor from the Device Controller.
ENABLE	This command enables the Device Interrupt to the Processor from the Device Controller.
UNBLOCK	This command enables the printer to print data entered via either the keyboard or the tape reader.
BLOCK	This command disables the feature described above.
WRT READ	The Write and Read commands are used to define the significance of the BSY bit.
PWR ON PWR OFF	The Power On and Power Off commands are significant only with those Teletypewriters provided with an optional Power Control Box. The option permit switching Teletypewriter power under program control.

HIGH SPEED PAPER TAPE READER STATUS AND COMMAND BYTE DATA HEX ADDRESS 03

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	OVERFLOW			NMTN	BSY	EX		DU
COMMAND BYTE	DISABLE	ENABLE	STOP	RUN	INCR	SLEW	REV	FWD

OVERFLOW	The Overflow bit is available for use with paper tape readers which operate in the Slew mode. The bit is set if the next character is read before a Data Request (DR) is received for the present character.
NMTN	The No Motion bit is set any time the tape is not moving
BSY	The Busy bit is set anytime there is a character in the buffer and no Data Request (DR) has been received from the Processor.
EX	The Examine bit is set whenever either Overflow or NMTN is set.
DU	The Device Unavailable bit is set if Reader Power is off, or if the LOAD/READY lever on the reader is in the LOAD position.
DISABLE	This command disables the Device Interrupt.
ENABLE	This command enables the Device Interrupt.
STOP	The Stop command stops reader tape motion.
RUN	The Run command starts the reader tape motion.
INCR	The Increment command directs the reader to read in Increment mode. The tape is stepped to the next character after each character is input to the Processor.
SLEW	The Slew command applies only to readers capable of operation in the Slew mode. In Slew mode the tape is started and continues to run until a particular character or string of characters on the tape is sensed.
REV	The Reverse command applies only to bi-directional tape readers.
FWD	The Forward command directs the reader to move the tape forward.

CARD READER STATUS AND COMMAND BYTE DATA (HEX ADDRESS 04)

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	EOV	TBL	HE	NMTN	BSY	EX	EOM	DU
COMMAND BYTE	DISABLE	ENABLE	FEED					

EOV	The EOV bit is set when the data is not taken from the Device Controller buffer before the next column of data arrives from the read station. This bit is reset by a FEED Command.
TBL/DU	These bits are set when the Card Reader fails to pick a card upon command, or when an error condition occurs in the Card Reader. The error conditions are: 1. Card Motion Error 2. Light Current Error 3. Dark Current Error These error conditions prevent the reading of any more cards until manually reset by the operator.
HE	The HE bit is set when the last card in the input hopper has been read. When HE sets, NMTN is set. The HE bit must be manually reset by the operator.
NMTN	The NMTN is set except for the time between a FEED command and the time it takes for a card to pass through the read station.
BSY	The BSY bit is set while the Device Controller is awaiting data from the Card Reader. It resets when the data is available to be transferred.
EX	The EX bit sets when any one of the upper four (4) bits of the Status byte is set.
EOM	The EOM bit is set whenever NMTN is set, and when the input hopper becomes empty. Reset when FEED command is issued.
DISABLE	This command disables the Card Reader Device Interrupt.
ENABLE	This command enables the Card Reader Device Interrupt.
FEED	This command initiates a new card feed cycle; however, no action occurs if TBL, DU, or HE is set.

TELETYPEWRITER/ASCII/HEX CONVERSION TABLE

HEX (MSD) →					8	9	A	B	C	D	E	F
(LSD) ↓	Teletype-writer Tape Channels → ↓			8	DEPENDS UPON PARITY							
				7	0	0	0	0	1	1	1	1
				6	0	0	1	1	0	0	1	1
				5	0	1	0	1	0	1	0	1
	4	3	2	1								
Ø	0	0	0	0	NULL	DC$_0$	SPACE	0	@	P		
1	0	0	0	1	SUM	X-ON	!	1	A	Q		
2	0	0	1	0	EOA	TAPE ON	"	2	B	R		
3	0	0	1	1	EOM	X-OFF	#	3	C	S		
4	0	1	0	0	EOT	TAPE OFF	$	4	D	T		
5	0	1	0	1	WRU	ERR	%	5	E	U		
6	0	1	1	0	RU	SYNC	&	6	F	V		
7	0	1	1	1	BELL	LEM	'	7	G	W		
8	1	0	0	0	FE$_0$	S$_0$	(	8	H	X		
9	1	0	0	1	HT/SK	S$_1$	)	9	I	Y		
A	1	0	1	0	LF	S$_2$	*	:	J	Z		
B	1	0	1	1	VT	S$_3$	+	;	K	[		
C	1	1	0	0	FF	S$_4$	,	<	L	\		ACK
D	1	1	0	1	CR	S$_5$	-	=	M	]		ALT. MODE
E	1	1	1	0	SO	S$_6$	.	>	N	↑		ESC
F	1	1	1	1	SI	S$_7$	/	?	O	←		DEL

ASCII/CARD CODE CONVERSION TABLE

GRAPHIC	8-BIT ASCII CODE	7-BIT ASCII CODE	CARD CODE	GRAPHIC	8-BIT ASCII CODE	7-BIT ASCII CODE	CARD CODE
SPACE	A0	20	0-8-2	@	C0	40	8-4
!	A1	21	12-8-7	A	C1	41	12-1
"	A2	22	8-7	B	C2	42	12-2
#	A3	23	8-3	C	C3	43	12-3
$	A4	24	11-8-3	D	C4	44	12-4
%	A5	25	0-8-4	E	C5	45	12-5
&	A6	26	12	F	C6	46	12-6
'	A7	27	8-5	G	C7	47	12-7
(	A8	28	12-8-5	H	C8	48	12-8
)	A9	29	11-8-5	I	C9	49	12-9
*	AA	2A	11-8-4	J	CA	4A	11-1
+	AB	2B	12-8-6	K	CB	4B	11-2
,	AC	2C	0-8-3	L	CC	4C	11-3
-	AD	2D	11	M	CD	4D	11-4
.	AE	2E	12-8-3	N	CE	4E	11-5
/	AF	2F	0-1	O	CF	4F	11-6
0	B0	30	0	P	D0	50	11-7
1	B1	31	1	Q	D1	51	11-8
2	B2	32	2	R	D2	52	11-9
3	B3	33	3	S	D3	53	0-2
4	B4	34	4	T	D4	54	0-3
5	B5	35	5	U	D5	55	0-4
6	B6	36	6	V	D6	56	0-5
7	B7	37	7	W	D7	57	0-6
8	B8	38	8	X	D8	58	0-7
9	B9	39	9	Y	D9	59	0-8
:	BA	3A	8-2	Z	DA	5A	0-9
;	BB	3B	11-8-6	[	DB	5B	12-8-2
<	BC	3C	12-8-4	\	DC	5C	11-8-1
=	BD	3D	8-6	]	DD	5D	11-8-2
>	BE	3E	0-8-6	↑	DE	5E	11-8-7
?	BF	3F	0-8-7	←	DF	5F	0-8-5

READER COMMENTS

The General Electric Company solicits your help in providing complete and accurate technical publications covering our Process Computer equipment. Please answer the questions listed here by checking the appropriate block. If your answer to any of these questions is "NO", please explain in "Comments" section below. Your comments and suggestions become the property of General Electric Company.

	YES	NO
• Is this publication adequate for your needs?	☐	☐
• Is the material		
Presented in clear text?	☐	☐
Conveniently organized?	☐	☐
Adequate detail?	☐	☐
Adequately illustrated?	☐	☐
Suitable for the technical level desired?	☐	☐

- What is your computer application? ______________________

  ____

- What is your position? (Supervisor, Programmer, Technician, etc.) ______________________
- How is this publication used:

Familiarization of the subject?	☐	As reference material?	☐
For training purposes?	☐	For maintenance of equipment?	☐

  Other (explain) ______________________________________

- Please give complete references (page number, line, etc.) with your comments. Please indicate if a reply is desired and include your proper mailing address.
- Your cooperation will be appreciated.

COMMENTS:

No postage necessary if mailed in the U. S. A.

Staple

YOUR ASSISTANCE, PLEASE

This document has been generated to help us serve you better. Your answers to the questions on the reverse side of this form, together with comments and recommendations, will be of great value to us in providing the best possible publications for your use. Your answers and comments will be carefully reviewed by the person who generated this publication, and may result in a revised publication. Your comments and recommendations become the property of General Electric Company.

Communications concerning Technical Publications should be directed to:

Manager, Technical Publications
GE Process Computer Department
2255 West Desert Cove Road
Phoenix, Arizona 85029

Fold — Fold

FIRST CLASS
Permit No. 4091
Phoenix, Arizona

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

GENERAL ELECTRIC COMPANY
PROCESS COMPUTER DEPARTMENT
2255 West Desert Cove Road
Phoenix, Arizona 85029

Attention: Technical Publications

Cut Along Line

Fold — Fold

Additional Comments:

Progress Is Our Most Important Product

GENERAL ELECTRIC

PROCESS COMPUTER DEPARTMENT

PHOENIX, ARIZONA