AC

E3ABP30534M 000
E3AZP30554 000

Technical Training

JSS DATA PROCESSOR AND DISPLAY MAINTENANCE

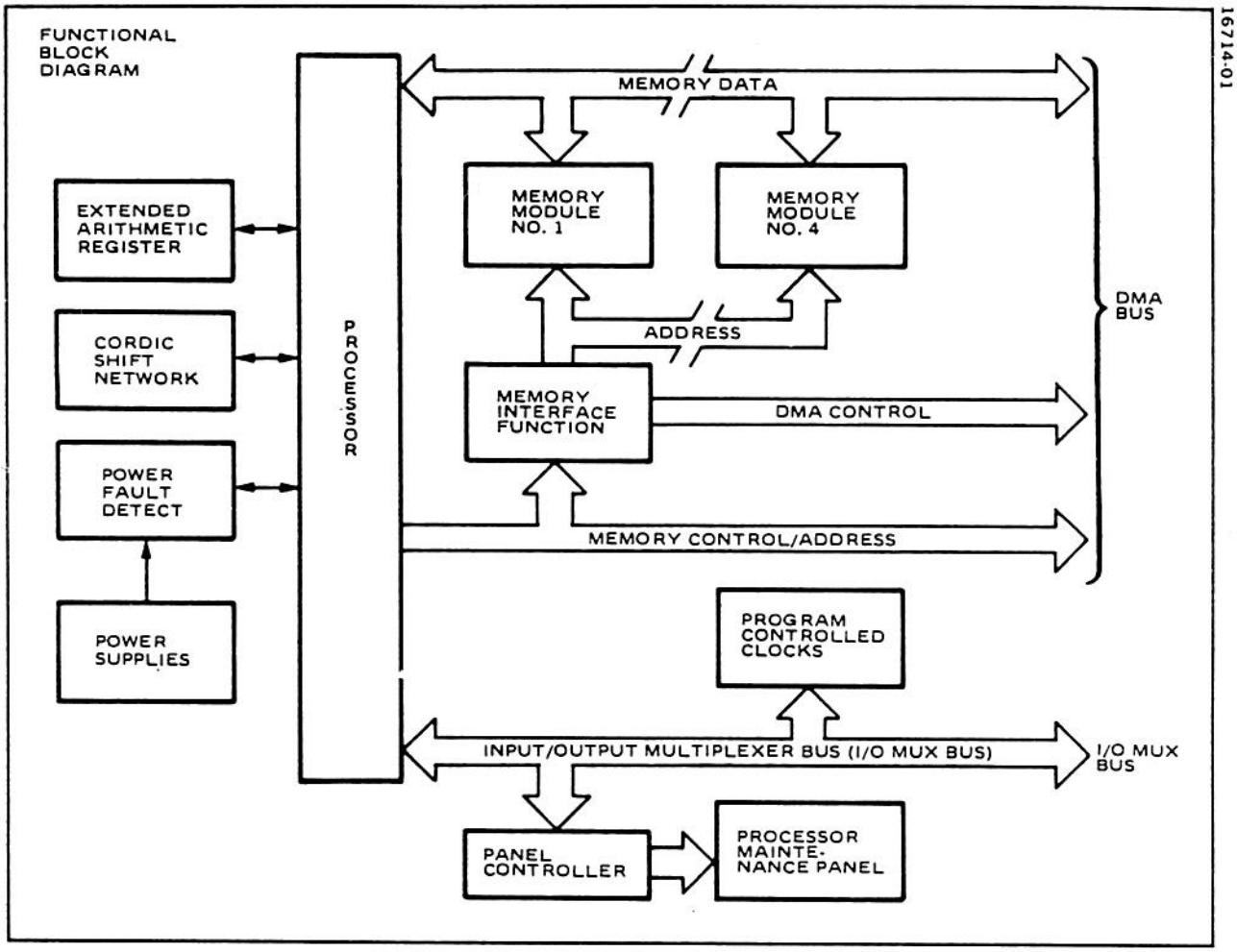
HMP-1116 INTRODUCTION

MAY 1986

USAF TECHNICAL TRAINING SCHOOL
3300th Technical Training WING
Keesler Air Force Base, Mississippi

HMP-1116 STUDENT GUIDE

TABLE OF CONTENTS



HMP-1116 Computer

HMP-1116 COMPUTER

CHARACTERISTICS

• ARITHMETIC	TWO'S COMPLEMENT, INTEGER
• DATA WORD LENGTHS	8, 16, 32, 48, AND 64 BITS
• DATA TYPES	FIXED-POINT AND FLOATING-POINT
• DATA FLOW	16-BIT PARALLEL (HALFWORDS)
• ALU WIDTH	32-BIT
• INSTRUCTION WORD LENGTH	16 AND 32 BITS
• GENERAL RESISTERS	SIXTEEN 16-BIT HARDWARE REGISTERS AND EIGHT 48-BIT FLOATING -POINT MEMORY REGISTERS
• STORAGE (READ/WRITE)	RANDOM ACCESS, DYNAMIC INTEGRATED CIRCUIT, LIMITED NON-VOLATILITY VIA BATTERY BACKUP
• STORAGE SIZE (MAXIMUM)	131,072 HALFWORDS (17 BITS EACH INCLUDING ONE PARITY BIT)

1.0 INTRODUCTION TO THE HMP-1116

The HMP-1116 minicomputer is a microprogrammable, general purpose, digital computer providing high-speed computation and data processing. The computer provides hardware capabilities for operating on binary data in field lengths of 8-bit bytes, 16-bit halfwords, and optional 32-bit full words, and 48-bit floating-point words. The computer is organized into functional units to achieve modularity at the card level for adding optional capabilities. The modularity allows various functional configurations without affecting existing card designs. The degree of flexibility includes variable memory size (128K halfwords maximum), ability to add, delete, and modify user instructions, Real Time Counter and Elapsed Time Counter option, extension to a 32-bit arithmetic logic unit (ALU), and Cordic shift network option (Trigonometric function).

The purpose of the HMP-1116 is to provide the capability to execute a program consisting of a sequence of software instructions and to communicate with external devices (peripherals). A software instruction is a binary word with a predefined format that the computer will recognize and execute. A program consists of a group of software instructions. The instructions in a program are located in consecutive memory locations.

1.1 Microsequencer. Figure 1-1 is a basic architecture of the HMP-1116 minicomputer. The central controlling function is the microsequencer function. This function consists of the Rom Address Register, Microprogram Memory and Rom Data Register. The Microprogram Memory is a read only memory (ROM) and contains up to 2,048 36-bit microinstructions. The Microprogram Memory is addressed by the Rom Address Register (RAR). The 36-bit microinstruction read from the Microprogram Memory is loaded into the Rom Data Register (RDR). Each microinstruction is broken into fields which are used to control the RAR, arithmetic operations, hex digit (4-bit) and byte (8-bit) manipulations, data movement between registers, input/output operations, memory and software execution. The Network Control is responsible for decoding the microinstruction in order to control these functions.

The microinstructions are arranged in sets within the Microprogram Memory. A set of microinstructions is called a microsequence. Each microsequence is responsible for performing a specific task.

While performing a specific microsequence, the RAR will count up by 1 every 200ns. To locate the next microsequence the RAR must be loaded with a new value. The RAR will be loaded 1) for a microprogram branch, 2) when a software instruction is to be executed or 3) on an interrupt.

The microprogram branch is implemented by taking a 12-bit field from the RDR and loading the RAR.

There is a microsequence in the Microprogram Memory for each software instruction. The start addresses for these microsequences are provided by the Instruction Decode Rom.

The Instruction Decode Rom also provides the start address for the interrupt service microsequence. Interrupts are generated from functions external to the microprocessor. The following analogy may promote further understanding:

> Let's say you are reading this document and your phone rings. This is an interrupt. When you answer the phone you have acknowledged the interrupt. Your conversation is service for the interrupt. Following your conversation you return to reading this document. The processor performs the same steps when receiving an interrupt. First it acknowledges the interrupt, then it performs the necessary service. Following this service the processor returns to its previous duties.

If the execution of a microinstruction takes more than 200ns (1 clock cycle), the RAR and RDR will be held until the operation is complete. Three cases in which this hold will occur are memory operations, input/output (I/O) operations and repeat operations.

Repeat operations are controlled by the Repeat Counter. This counter will allow a single microinstruction in the RDR to be repeated a maximum of 31 times.

1.2 Arithmetic Logic Unit (ALU). The ALU is a 16-bit arithmetic function which can be expanded to a 32-bit arithmetic function. The ALU is under direct control of the microprogram and is responsible for performing all logical and arithmetic operations. The ALU function consists of the Byte Manipulator, Flag Register and AM2901 large scale integrated circuits (LSI). Each AM2901 chip provides 4 bits of ALU. Therefore, a 16-bit ALU requires four AM2901s.

The AM2901s will either perform an arithmetic or logical function, or pass data unchanged to the A-bus output. The specific operation is dictated by the microprogram. The AM2901s also provide 16 scratch pad registers (internal registers) which are used by the microprogram as a work area. There is also a Q register which is used during multiply, divide and shift operations.

The Flag Register (FLR) is a four bit register used to indicate the magnitude of the arithmetic result. The four bits of the FLR are called the C, V, G and L flags.

The C (carry) flag is set when a carry out is generated from the AM2901.

The V (overflow) flag is set when an arithmetic overflow occurs. An example of an overflow is when two numbers with like signs are added and produce a result with a different sign.

The G (greater than) flag is set when the arithmetic result is greater than zero.

The L (less than) flag is set when the arithmetic result is less than zero.

The FLR may be examined by the microprogram in order to make decisions based on a previous operation.

The Byte Manipulator is under microprogram control. This function allows the following operations to be performed:

1) byte exchange
2) hex digit extraction
3) no change to data

A byte exchange operation means that the 8 MSBs and 8 LSBs of a 16-bit value on the B-bus are exchanged. The new 16-bit word is then applied to the direct input of the AM2901.

Example of Byte Exchange:

Input from the B-bus(In hex) = 'ABCD'
Output of the Byte Manipulator = 'CDAB'

Hex digit extraction allows any one of the 4 hex digits of a 16-bit word to be placed in the hex LSD position. The three hex MSDs are set to zero.

Example of Hex Digit Extraction:

Input to the Byte Manipulator(In hex) = 'ABCD'
Output of the Byte Manipulator for extracting MSD = 'OOOA'
Byte Manipulator Output for extracting second MSD = 'OOOB'
Byte Manipulator Output for extracting third MSD = 'OOOC'
Byte Manipulator Output for extracting the LSD = 'OOOD'

1.3 Data Movement Between Registers. Data can be transferred from an internal or external register to an internal or external register. The register sending data is called the source. The register receiving data is called the destination. The source and destination registers are specified by the microprogram.

Data is 16 bits of arithmetic or logical information. A 16-bit arithmetic value consists of data (15 LSBs) and a sign bit (MSB). Negative values are in two's complement and have a sign bit set high. Logical values consist of 16 bits of unsigned data.

A source register can be an internal register, the Q register, an external register or data from an I/O device (via the I/O transceivers). External registers include all registers outside the dotted lines on Figure 1-1. A list of external registers which can be specified by the microprogram as a source is shown below.

1) Rom Data Register (RDR)
2) Rom Address Register (RAR)
3) Program Status Register (PSR)
4) Any one of the 16x16 Register File
5) Instruction Register (IR)
6) Memory Address Register (MAR)
7) Memory Data Register (MDR)

A destination register can be an internal register, the Q register, an external register or data to an I/O device (via the I/O transceivers). A list of external registers which can be specified by the microprogram as a destination is shown below.

1) Rom Address Register
2) Repeat Counter (CTR)
3) Flag register (FLR)
4) Program Status Register
5) Any one of the 16x16 Register File
6) Instruction Register
7) Memory Data Register
8) Memory Address Register

An external source register is placed on the B-bus and passes through the Byte Manipulator to the AM2901. An external destination is loaded from data placed on the A-bus by the AM2901.

If data is moved from one internal register to another, the source register contents can be passed onto the A-bus, through the Rom Address B-Bus Mux, to the B-bus, through the Byte Manipulator to the D input of the AM2901 and loaded into the destination register. This operation will be performed when an internal register is to be modified by the Byte Manipulator.

The Rom Address B-Bus Mux is also used when the RDR or RAR is specified as the source. The RAR will be specified as a source if it needs to be saved in an internal register. The RDR is specified as the source if a field of data from the microinstruction is to be loaded into an internal register.

1.4 MOS RAM Memory. The MOS RAM Memory is used by the programmer to store his program and data. The memory in the HMP-1116 can be varied in size by addition or deletion of memory cards (32K each). The maximum memory configuration is 131,072 (128K, where 1K = 1,024) 16-bit words. For each location in memory there are 17 bits (16 data bits plus 1 parity bit). When a word is stored in memory, a parity bit is also stored to provide odd

parity (odd number of 1s). When data is read from memory this parity is checked. A memory cycle (time required to read or store data) is 600ns. Since the RAM Memory is MOS type it requires a refresh cycle every 14.8 microseconds.

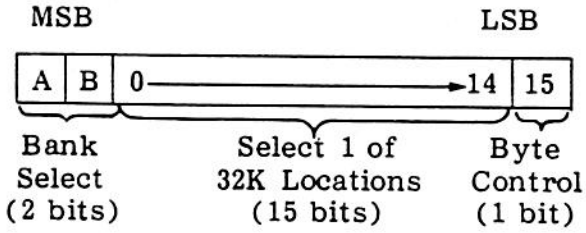
There are three functions that access memory. These are listed below from highest priority to lowest.

1) Refresh
2) DMA (Direct Memory Access)
3) Processor

Processor Memory Requests

The processor uses four registers for memory operations. These are the Memory Address Register (MAR), Memory Data Register (MDR), Instruction Register (IR), and Program Status Register (PSR).

The Memory Address Bus is used to locate one of 128K words in memory. The two MSBs (PSR8:9 during Instruction Fetch or PSR10:11 during Operand Read) select one of four memory banks (32K each). The LSB of the MAR is not sent to the bus, instead it is provided to the Byte Manipulator for control byte operations. The remaining 15 bits of the MAR are used to select one of 32K words within a memory bank.



A byte operation is when an 8-bit byte is extracted from a 16-bit source register, or when an 8-bit byte is inserted into 8 bits of a 16-bit destination register.

In an extract operation the MDR is the source. Either the 8 MSBs or the 8 LSBs can be selected as the byte operand. If the LSB of the MAR is 0, the Byte Manipulator selects the 8 MSBs. If the LSB of the MAR is 1, the Byte Manipulator selects the 8 LSBs.

Byte Extraction Example

MAR LSB	Byte Manipulator Input	Byte Manipulator Output
0	'ABCD'	'OOAB'
1	'ABCD'	'OOCD'

In an insert operation an internal register or the MDR is the destination. An 8-bit byte on the 8 LSBs of the B-bus is inserted into either the 8 MSBs or the 8 LSBs of the destination register, leaving 8 bits of the destination unchanged. If the LSB of the MAR is 0, a byte is inserted into the 8 MSBs of the destination. If the LSB of the MAR is 1, a byte is inserted into the 8 LSBs of the destination.

Byte Insertion Example:

Destination Before	Byte Manipulator Input from B-bus	MAR LSB	Destination After
'1234'	'OOAB'	0	'AB34'
'1234'	'OOAB'	1	'12AB'

The MDR is used to hold data read from memory or data to be written into memory.

The IR is loaded with a software instruction read from memory. This instruction is broken into fields. One of these fields, called the op-code, defines the type of instruction and the operation to be performed. The op-code is decoded by the Instruction Decode Rom to provide a microsequence start address, which is loaded into the RAR. At this point the software instruction will be executed by its unique microsequence residing in the Microprogram Rom. Another field from the software instruction allows the programmer access to the 16x16 Register File. These registers are usually called General Registers. When the microprogram uses the 16x16 Register File as a source or destination, the software instruction in the IR specifies which one of the 16 registers is to be used.

DMA Memory Requests

The Direct Memory Access (DMA) port allows an external device to address the memory and either read or write data. The DMA interface will be discussed in detail later in this chapter.

1.5 Program Status Word. The Program Status Word (PSW) consists of a 16-bit external register called the Program Status Register (PSR) and a 16-bit internal register which is called the Location Counter.

The Location Counter (LOC) is used by the microprogram to locate the next software instruction in memory to be executed. Each time an instruction is read from memory and loaded into the Instruction Register, the LOC is incremented to point to the next instruction.

The PSR contains information that the programmer can use to control software branching and to designate the proper memory banks in which his program and data are located. The programmer can change the PSR by using specific software instructions.

0	1 7	8 9	10 11	12 15	16 31
Wait	Status	Program Bank	Operand Bank	Condition Code	Location Counter (Int R_7 (ALU))

PSW see (2-2)

The PSR bit 0 (MSB) is called the Wait bit. When this bit is set, program execution halts.

PSR bits 1 through 7 (Status) control interrupt service.

Bits 8 and 9 of the PSR are the Program Bank address. They are used to select one of four banks when reading an instruction from memory.

PSR bits 10 and 11 are the Operand Bank address. They are used to select one of four banks when reading an operand from memory. The Program Bank address or Operand Bank address is loaded into the two MSBs of the MAR by the microprogram to provide the appropriate bank selection.

Bits 12 through 15 of the PSR are the Condition Code. This field is loaded from the Alarm Register (4 bits) or the Flag Register (4 bits).

PSW bits 16 through 31 are the Location Counter (LOC). The LOC is stored in the 16x16 Ram Scratchpad in the AM2901. The LOC defines the memory address of the next software instruction to be executed.

Alarm Register

Each bit of the Alarm Register represents a specific hardware fault. The four faults that cause a respective bit in the Alarm Register to be set are memory violation (bit 12), parity error when reading an operand (bit 13), parity error when reading an instruction (bit 14) and power failure (bit 15).

A memory violation can occur only if the computer is equipped with the Memory Protect Controller (MPC) option. The MPC option allows the programmer to designate portions of memory as protected. The software program and DMA will be unable to write into protected memory.

1.6 Input/Output Operations. The I/O Mux Bus allows the HMP-1116 to communicate with external devices (peripherals) such as magnetic tape units, card readers or line printers. The interface between the HMP-1116 and the peripheral is indirectly controlled by I/O software instructions. The direct control is provided by the microprogram.

By using appropriate software instructions the programmer can send commands, send or retrieve data and get status information from an I/O

device over the I/O Mux Bus. The software instructions specify General Registers and memory locations that are to be used for I/O transfers. The microprogram uses this information and sequentially controls I/O operations.

The HMP-1116 can communicate with one I/O device at a time. Each device is assigned a unique 8-bit device address. In order to begin communication, the I/O device must first be addressed by the HMP-1116.

Figure 1-2 represents the HMP-1116 I/O Mux Bus interface. The interface is common to all I/O devices connected to the HMP-1116. All information (device address, commands, status and data) is transferred via the bidirectional data bus ($\overline{T}$DOO-15). The type of information is defined by the flag set by the computer.

The Address Flag ($\overline{T}$ADRS) is set active when the 8 LSBs ($\overline{T}$D08-15) contain a device address. The device whose address is on the data lines will respond by setting the Sync ($\overline{T}$SYN) active. This informs the HMP-1116 microprogram that the device has received and decoded its address. This device will now be able to receive a command, send its status or transfer data.

The Command Flag ($\overline{T}$CMD) is set active by the microprogram when the 8 LSBs of the I/O Mux Bus ($\overline{T}$D08-15) contain a command for the addressed device. When the device receives this command it responds by setting $\overline{T}$SYN active.

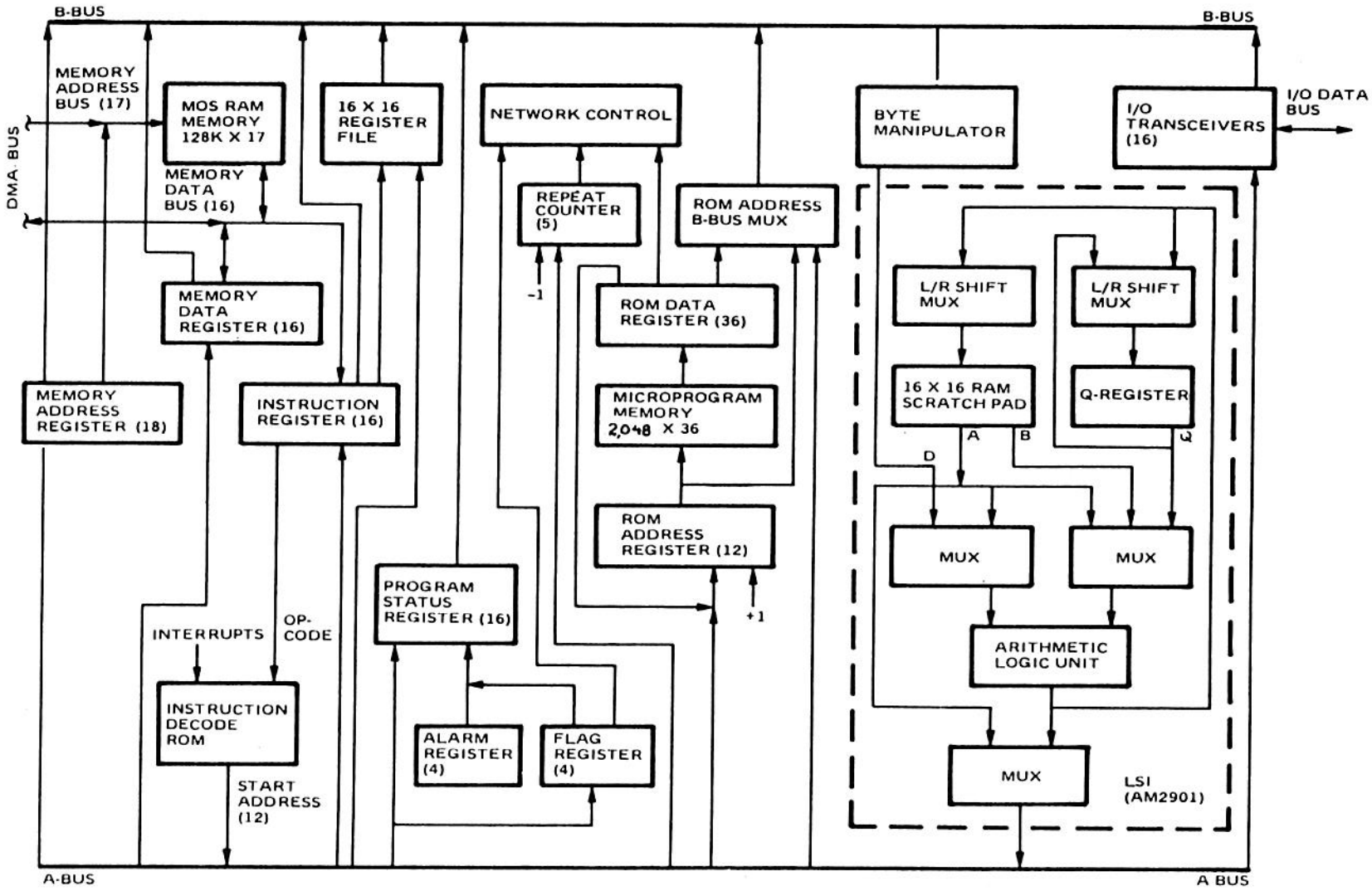
When the Status Flag ($\overline{T}$SR) is set active the addressed device will place its 8-bit status on the I/O Mux Bus LSBs and set $\overline{T}$SYN active.

The addressed I/O device sets the Halfword Flag ($\overline{T}$HW) active if it can send or receive 16 bits of data at a time. If the device does not set $\overline{T}$HW active it can only send or receive 8 bits of data at a time. The microprogram examines THW prior to sending or receiving data.

The Data Available Flag ($\overline{T}$DA) is set active when the I/O Mux Bus contains data for the addressed device. Data will be either 16 bits (on $\overline{T}$D00-15) or 8 bits (on $\overline{T}$D08-15) depending on the state of $\overline{T}$HW. The device responds with $\overline{T}$SYN when it receives the data.

When the Data Request Flag ($\overline{T}$DR) is set active, the addressed device will place either 16 bits (on $\overline{T}$D00-15) or 8 bits (on $\overline{T}$D08-15) on the I/O Mux Bus (depending on the state of $\overline{T}$HW) and set $\overline{T}$SYN active.

An interrupt from an I/O device is a request for service. The Interrupt Flag ($\overline{T}$ATN) is common to all devices connected to the I/O Mux Bus. When $\overline{T}$ATN goes active, the computer must determine which device set the interrupt. To accomplish this the Acknowledge Flag ($\overline{T}$ACK) is set active by the HMP-1116. The highest priority I/O device that has an interrupt pending will place its own device address on $\overline{T}$D08-15 to identify itself as the interrupting device and sets $\overline{T}$SYN active.



16714-2

Figure 1-1. HMP-1116 Processor Architecture

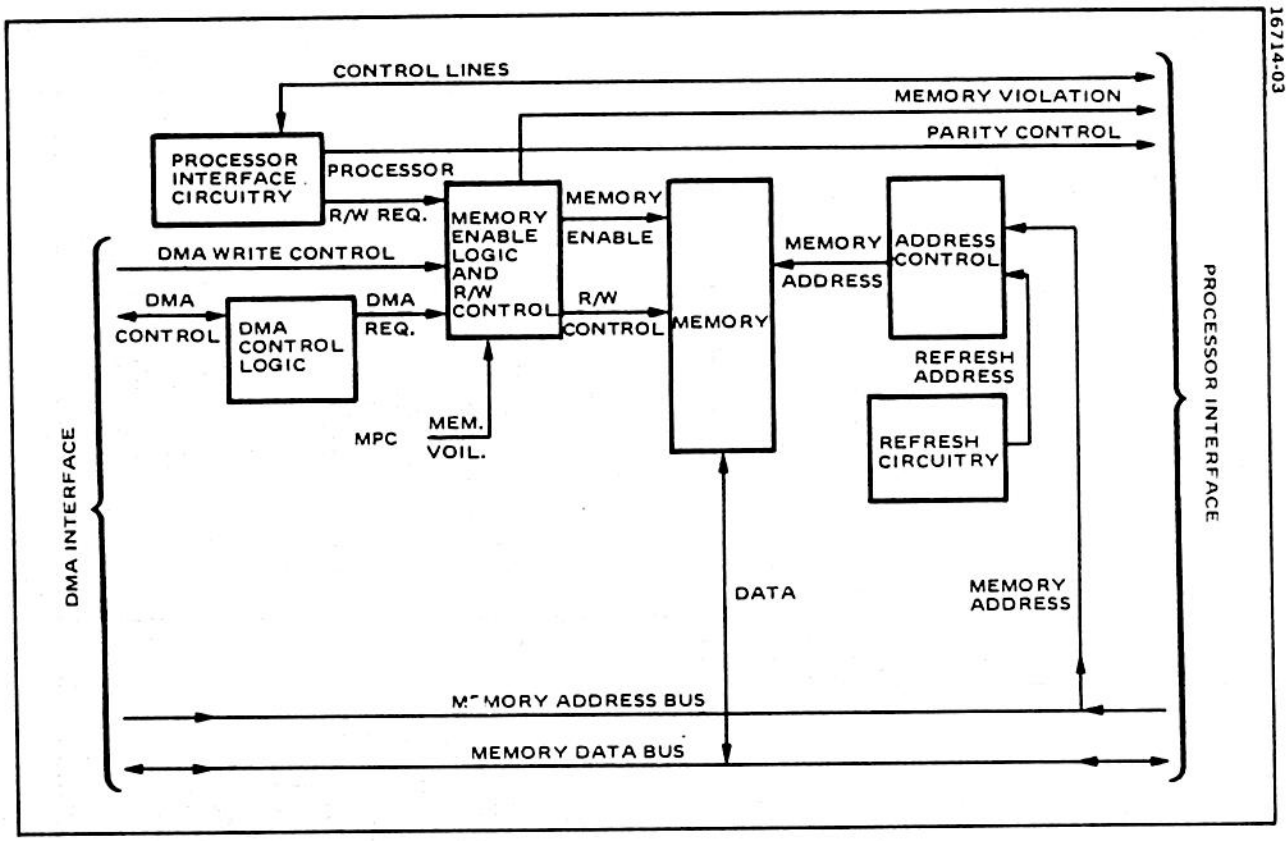


Figure 1-1A. HMP-1116 Memory Architecture

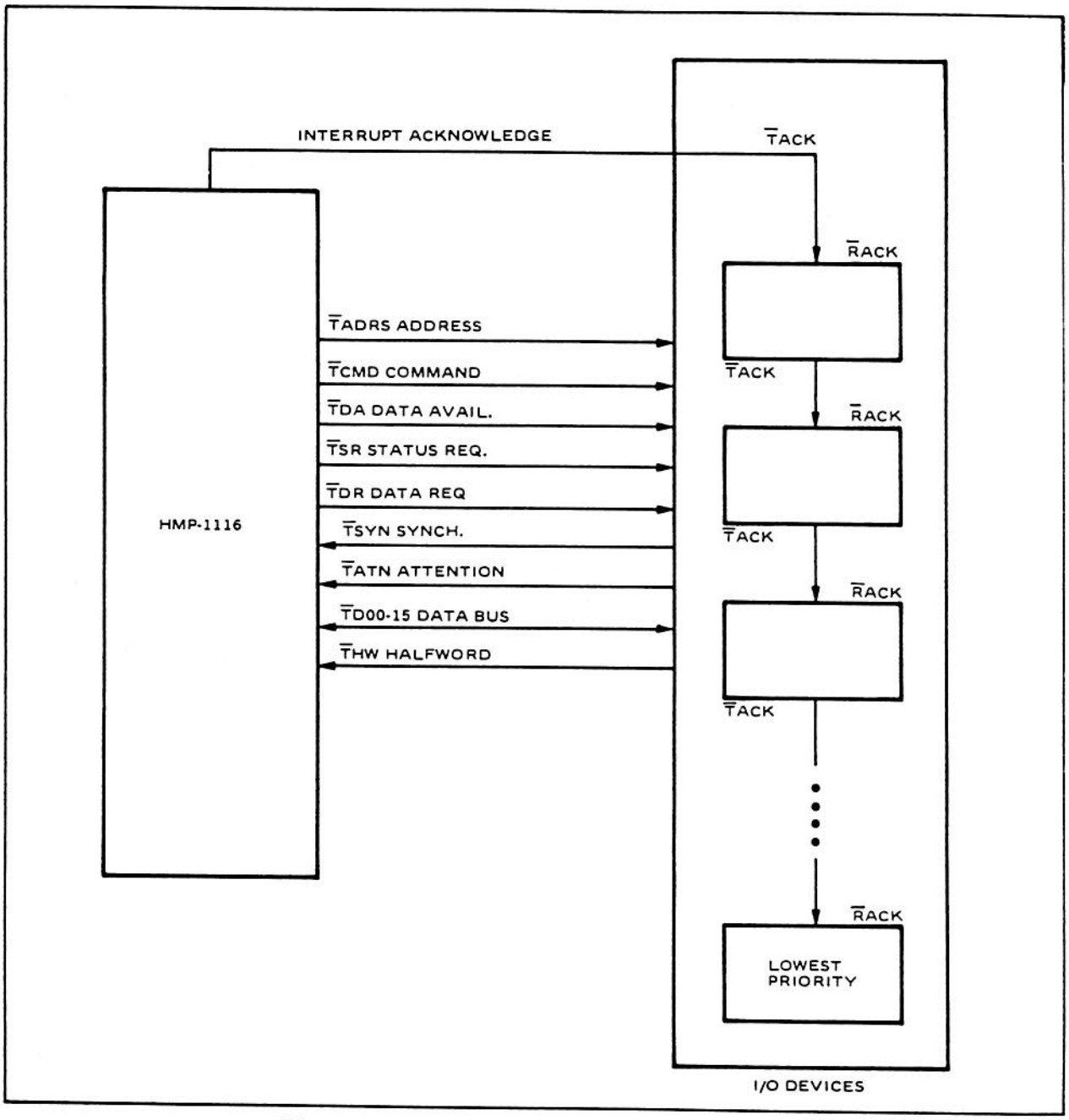


Figure 1-2. HMP-1116 I/O Mux Bus Interface

1.7 1.7 DMA Interface. The Direct Memory Access (DMA) Bus allows an external device to address HMP-1116 memory and read or write data. All DMA devices have an I/O Mux Bus interface also. The I/O interface provides initialization data (such as the starting address in the HMP-1116 memory) to be used by the DMA device during data transfers. The DMA device is responsible for updating the memory address and maintaining a word count when transferring a group of data.

Figure 1-3 represents the DMA Bus interface. The DMA Request (TDMAREQ) is set active by the DMA device when it wants to use memory. The HMP-1116 responds with DMA Acknowledge ($\overline{T}$EN).

DMA Read

When $\overline{T}$EN is received by the device it sends a 17-bit memory address ($\overline{T}$MAA, $\overline{T}$MAB, $\overline{T}$MA00-14) to the HMP-1116. It also indicates that it wants to read from memory by setting the Read/Write Control ($\overline{T}$DMAWRT) inactive. The DMA Strobe ($\overline{T}$DMAST) will indicate that memory data is ready to be taken from the Memory Data Bus ($\overline{T}$MD00-15). At this time the DMA device must accept the data and remove the memory address and Read/Write Control.

DMA Write

When $\overline{T}$EN is received by the DMA device it places memory address, Read/Write Control and memory data to be written into the HMP-1116 on the DMA Bus. The DMA Strobe will indicate that data has been written into memory. When the DMA Strobe is received, the device must remove memory address, Read/Write Control and data from the DMA Bus.

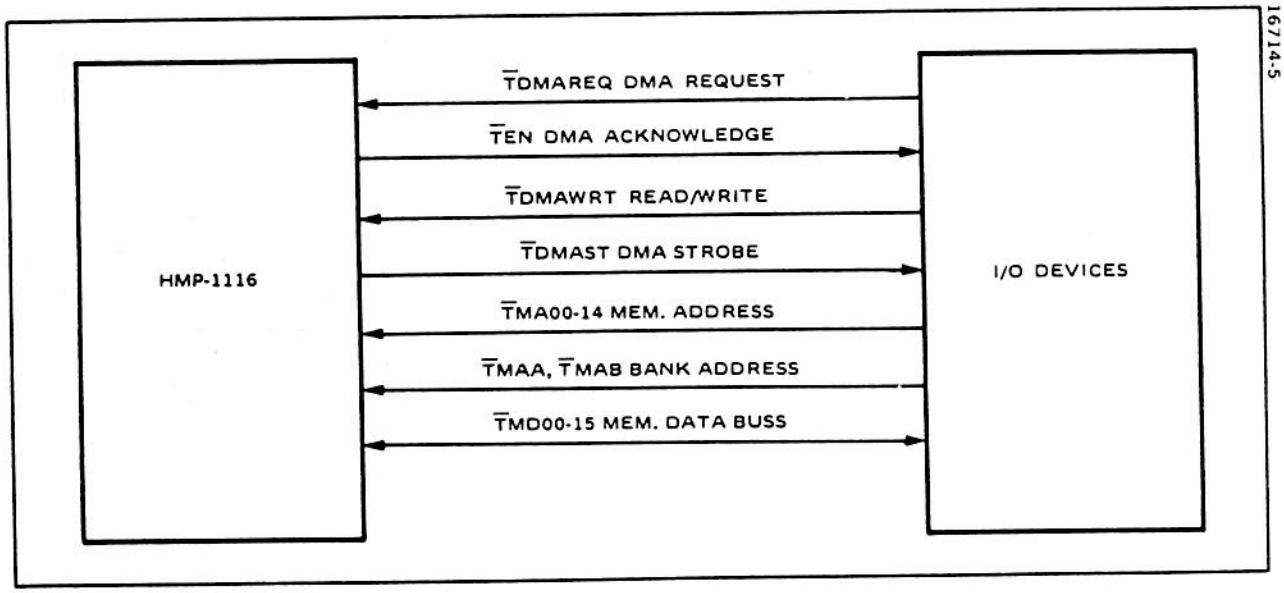


Figure 1-3. HMP-1116 DMA Bus Interface

Am2901

Four-Bit Bipolar Microprocessor Slice

DISTINCTIVE CHARACTERISTICS

- 16-word x 4-bit two-port RAM.
- High speed ALU.
- 9-bit microinstruction word.
- Advanced low-power Schottky processing.
- Four-bit slice cascadable to any number of bits with full carry look-ahead.
- Three-state outputs.
- Shift left, no shift, or shift right entry into RAM from ALU.
- Output multiplexer for direct RAM A-port access or ALU output.
- Status flags include carry-out, sign-bit (negative), overflow and zero detect.
- Four-bit Q-register for scratch pad or accumulator extension.
- Direct ALU entry to Q-register.
- Shift Q-register left or right.
- RAM-shift and Q-shift are easily cascadable.

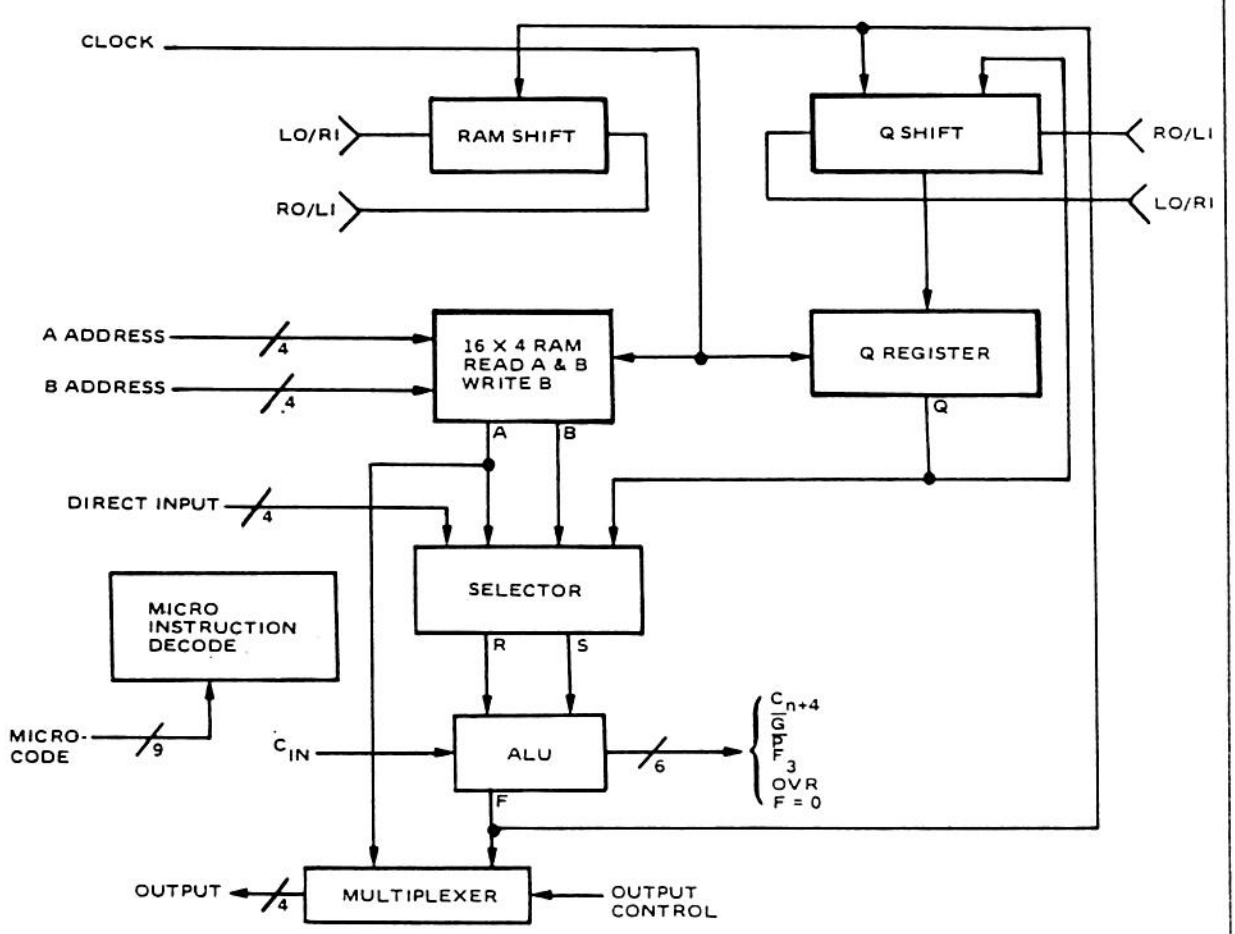
GENERAL DESCRIPTION

The four-bit bipolar microprocessor slice is designed as a high-speed cascadable element intended for use in CPU's, peripheral controllers, programmable microprocessors and numerous other applications. The microinstruction flexibility of the Am2901 will allow efficient emulation of almost any digital computing machine.

The device, as shown in the block diagram below consists of a 16-word by 4-bit two-port RAM, a high-speed ALU and the associated shifting, decoding and multiplexing circuitry. The nine-bit microinstruction word is organized into three groups of three bits each and selects the ALU source operands, the ALU function, and the ALU destination register. The microprocessor is cascadable with full look-ahead or with ripple carry, has three-state outputs, and provides various status flag outputs from the ALU. Advanced low-power Schottky processing is used to fabricate this 40-lead LSI chip.

16714-6

MICROPROCESSOR SLICE BLOCK DIAGRAM



ARCHITECTURE

A detailed block diagram of the bipolar microprogrammable microprocessor structure is shown in Figure 1. The circuit is a four-bit slice cascadable to any number of bits. Therefore, all data paths within the circuit are four bits wide. The two key elements in the Figure 1 block diagram are the 16-word by 4-bit 2-port RAM and the high-speed ALU.

Data in any of the 16 words of the Random Access Memory (RAM) can be read from the A-port of the RAM as controlled by the 4-bit A address field input. Likewise, data in any of the 16 words of the RAM as defined by the B address field input can be simultaneously read from the B-port of the RAM. The same code can be applied to the A select field and B select field in which case the identical file data will appear at both the RAM A-port and B-port outputs simultaneously.

When enabled by the RAM write enable (RAM EN), new data is always written into the file (word) defined by the B address field of the RAM. The RAM data input field is driven by a 3-input multiplexer. This configuration is used to shift the ALU output data (F) if desired. This three-input multiplexer scheme allows the data to be shifted up (right) one bit position, shifted down (left) one bit position, or not shifted in either direction.

The RAM A-port data outputs and RAM B-port data outputs drive separate 4-bit latches. These latches hold the RAM data while the clock input is LOW. This eliminates any possible race conditions that could occur while new data is being written into the RAM.

The high-speed Arithmetic Logic Unit (ALU) can perform three binary arithmetic and five logic operations on the two 4-bit input words R and S. The R input field is driven from a 2-input multiplexer, while the S input field is driven from a 3-input multiplexer. Both multiplexers also have an inhibit capability; that is, no data is passed. This is equivalent to a "zero" source operand.

Referring to Figure 1, the ALU R-input multiplexer has the RAM A-port and the direct data inputs (D) connected as inputs. Likewise, the ALU S-input multiplexer has the RAM A-port, the RAM B-port and the Q register connected as inputs.

This multiplexer scheme gives the capability of selecting various pairs of the A, B, D, Q and "0" inputs as source operands to the ALU. These five inputs, when taken two at a time, result in ten possible combinations of source operand pairs. These combinations include AB, AD, AQ, A0, BD, BQ, B0, DQ, D0 and Q0. It is apparent that AD, AQ and A0 are somewhat redundant with BD, BQ and B0 in that if the A address and B address are the same, the identical function results. Thus, there are only seven completely non-redundant source operand pairs for the ALU. The Am2901 microprocessor implements eight of these pairs. The microinstruction inputs used to select the ALU source operands are the I_0, I_1, and I_2 inputs. The definition of I_0, I_1, and I_2 for the eight source operand combinations are as shown in Figure 2. Also shown is the octal code for each selection.

The two source operands not fully described as yet are the D input and Q input. The D input is the four-bit wide direct data field input. This port is used to insert all data into the working registers inside the device. Likewise, this input can be used in the ALU to modify any of the internal data files. The Q register is a separate 4-bit file intended primarily for multiplication and division routines but it can also be used as an accumulator or holding register for some applications.

The ALU itself is a high-speed arithmetic/logic operator capable of performing three binary arithmetic and five logic functions. The I_3, I_4, and I_5 microinstruction inputs are used to select the ALU function. The definition of these inputs is shown in Figure 3. The octal code is also shown for reference. The normal technique for cascading the ALU of several devices is in a look-ahead carry mode. Carry generate, $\overline{G}$, and carry propagate, $\overline{P}$, are outputs of the device for use with a carry-look-ahead-generator such as the Am2902 ('182). A carry-out, C_{n+4}, is also generated and is available as an output for use as the carry flag in a status register. Both carry-in (C_n) and carry-out (C_{n+4}) are active HIGH.

The ALU has three other status-oriented outputs. These are F_3, F = 0, and overflow (OVR). The F_3 output is the most significant (sign) bit of the ALU and can be used to determine positive or negative results without enabling the three-state data outputs. F_3 is non-inverted with respect to the sign bit output Y_3. The F = 0 output is used for zero detect. It is an open-collector output and can be wire OR'ed between microprocessor slices. F = 0 is HIGH when all F outputs are LOW. The overflow output (OVR) is used to flag arithmetic operations that exceed the available two's complement number range. The overflow output (OVR) is HIGH when overflow exists. That is, when C_{n+3} and C_{n+4} are not the same polarity.

The ALU data output is routed to several destinations. It can be a data output of the device and it can also be stored in the RAM or the Q register. Eight possible combinations of ALU destination functions are available as defined by the I_6, I_7, and I_8 microinstruction inputs. These combinations are shown in Figure 4.

The four-bit data output field (Y) features three-state outputs and can be directly bus organized. An output control ($\overline{OE}$) is used to enable the three-state outputs. When $\overline{OE}$ is HIGH, the Y outputs are in the high-impedance state.

A two-input multiplexer is also used at the data output such that either the A-port of the RAM or the ALU outputs (F) are selected at the device Y outputs. This selection is controlled by the I_6, I_7, and I_8 microinstruction inputs. Refer to Figure 4 for the selected output for each microinstruction code combination.

As was discussed previously, the RAM inputs are driven from a three-input multiplexer. This allows the ALU outputs to be entered non-shifted, shifted up one position (X2) or shifted down one position (÷2). The shifter has two ports; one is labeled RAM_0–LO/RI and the other is labeled RAM_3–RO/LI. Both of these ports consist of a buffer-driver with a three-state output and an input to the multiplexer. Thus, in the shift up mode, the RO buffer is enabled and the RI multiplexer input is enabled. Likewise, in the shift down mode, the LO buffer and LI input are enabled. In the no-shift mode, both the LO and RO buffers are in the high-impedance state and the multiplexer inputs are not selected. This shifter is controlled from the I_6, I_7, and I_8 microinstruction inputs as defined in Figure 4.

Similarly, the Q register is driven from a 3-input multiplexer. In the no-shift mode, the multiplexer enters the ALU data into the Q register. In either the shift-up or shift-down mode, the multiplexer selects the Q register data appropriately shifted up or down. The Q shifter also has two ports; one is labeled Q_0–LO/RI and the other is Q_3–RO/LI. The operation of these two ports is similar to the RAM shifter and is also controlled from I_6, I_7, and I_8 as shown in Figure 4.

The clock input to the Am2901 controls the RAM, the Q register, and the A and B data latches. When enabled, data is clocked into the Q register on the LOW-to-HIGH transition of the clock. When the clock input is HIGH, the A and B latches are open and will pass whatever data is present at the RAM outputs. When the clock input is LOW, the latches are closed and will retain the last data entered. If the RAM-EN is enabled, new data will be written into the RAM file (word) defined by the B address field when the clock input is LOW.

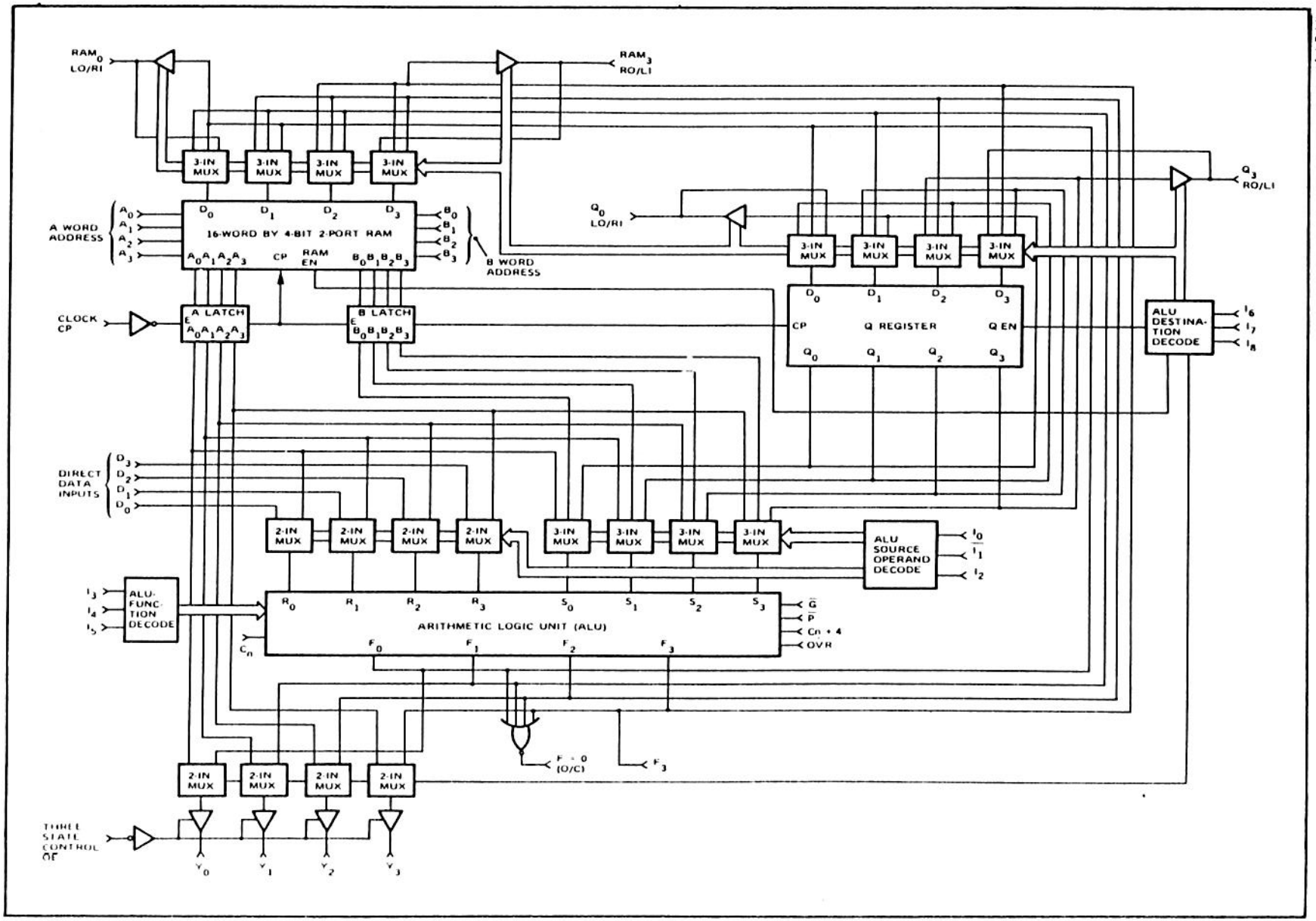


Figure 1. Detailed Am2901 Microprocessor Block Diagram

MICRO CODE				ALU SOURCE OPERANDS	
I_2	I_1	I_0	OCTAL CODE	R	S
L	L	L	0	A	Q
L	L	H	1	A	B
L	H	L	2	O	Q
L	H	H	3	O	B
H	L	L	4	O	A
H	L	H	5	D	A
H	H	L	6	D	Q
H	H	H	7	D	O

Figure 2. ALU Source Operand Control

MICRO CODE				ALU FUNCTION	SYMBOL
I_5	I_4	I_3	OCTAL CODE		
L	L	L	0	R PLUS S	R + S
L	L	H	1	S MINUS R	S − R
L	H	L	2	R MINUS S	R − S
L	H	H	3	R OR S	R ∨ S
H	L	L	4	R AND S	R ∧ S
H	L	H	5	$\overline{R}$ AND S	$\overline{R} \wedge S$
H	H	L	6	R EX-OR S	R ∀ S
H	H	H	7	R EX-NOR S	$\overline{R \forall S}$

Figure 3. ALU Function Control

MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER	
I_8	I_7	I_6	OCTAL CODE	SHIFT	LOAD	SHIFT	LOAD		RAM_0 LO/RI	RAM_3 LI/RO	Q_0 LO/RI	Q_3 LI/RO
L	L	L	0	—	—	NONE	ALU (F_i)	F	Z	Z	Z	Z
L	L	H	1	—	—	—	—	F	Z	Z	Z	Z
L	H	L	2	NONE	ALU (F_i)	—	—	A	Z	Z	Z	Z
L	H	H	3	NONE	ALU (F_i)	—	—	F	Z	Z	Z	Z
H	L	L	4	LEFT (DOWN)	ALU (F_{i+1})	LEFT (DOWN)	Q-REG (Q_{i+1})	F	F_0	IN_3	Q_0	IN_3
H	L	H	5	LEFT (DOWN)	ALU (F_{i+1})	—	—	F	F_0	IN_3	Q_0	IN_3
H	H	L	6	RIGHT (UP)	ALU (F_i-1)	RIGHT (UP)	Q-REG (Q_{i-1})	F	IN_0	F_3	IN_0	Q_3
H	H	H	7	RIGHT (UP)	ALU (F_{i-1})	—	—	F	IN_0	F_3	IN_0	Q_3

Z = HIGH-IMPEDANCE

Figure 4. ALU Destination Control

+ = PLUS; − = MINUS; ∨ = OR; ∧ = AND; ∀ = EX-OR

R, S / I_5 I_4 I_3 C_n	I_2 I_1 I_0	L L A, Q L	L L A, B H	L H O, Q L	L H O, B H	H L O, A L	H L D, A H	H H D, Q L	H H D, O H
		0	1	2	3	4	5	6	7
L L L L R PLUS S H		A+Q A+Q+1	A+B A+Q+1	Q Q+1	B B+1	A A+1	D+A D+A+1	D+Q D+Q+1	D D+1
L L H L S MINUS R H		Q−A−1 Q−A	B−A−1 B−A	Q−1 Q	B−1 B	A−1 A	A−D−1 A−D	Q−D−1 Q−D	−D−1 −D
L H L L R MINUS S H		A−Q−1 A−Q	A−B−1 A−B	−Q−1 −Q	−B−1 −B	−A−1 −A	D−A−1 D−A	D−Q−1 D−Q	D−1 D
L H H L R OR S H		AVQ	AVB	Q	B	A	DVA	DVQ	D
H L L L R AND S H		A∧Q	A∧B	0	0	0	D∧A	D∧Q	0
H L H L $\overline{R}$ AND S H		$\overline{A}$∧Q	$\overline{A}$∧B	Q	B	A	$\overline{D}$∧A	$\overline{D}$∧Q	0
H H L L R EX-OR S H		A∀Q	A∀B	Q	B	A	D∀A	D∀Q	D
H H H L R EX-NOR S H		$\overline{A \forall Q}$	$\overline{A \forall B}$	$\overline{Q}$	$\overline{B}$	$\overline{A}$	$\overline{D \forall A}$	$\overline{D \forall Q}$	$\overline{D}$

Figure 5. Source Operand and ALU Function Matrix

SOURCE OPERANDS AND ALU FUNCTIONS

As discussed earlier, there are eight source operand pairs available to the ALU as selected by the I_0, I_1, and I_2 instruction inputs. The ALU can perform eight functions; five logic and three arithmetic. The I_3, I_4, and I_5 instruction inputs control this function selection. The carry input, C_n, also affects the ALU results when in the arithmetic mode. The C_n input is inhibited in the logic mode. When I_0 through I_5 and C_n are viewed together, the matrix of Figure 5 results. This matrix fully defines the ALU/source operand function for each state.

The ALU functions can also be examined on a "task" basis, i.e., add, subtract, AND, OR, etc. In the arithmetic mode, the carry will affect the function performed while in the logic mode, the carry will have no bearing on the ALU output. Figure 6 defines the various logic operations that the Am2901 can perform and Figure 7 shows the arithmetic functions of the device. Both carry-in LOW (C_n = 0) and carry-in HIGH (C_n = 1) are defined in these operations.

I_5 I_4 I_3 I_2 I_1 I_0	OCTAL I_{543},I_{210}	GROUP	FUNCTION
1 0 0 0 0 0	40	AND	A ∧ Q
0 0 1	41		A ∧ B
1 0 1	45		D ∧ A
1 1 0	46		D ∧ Q
0 1 1 0 0 0	30	OR	A ∨ Q
0 0 1	31		A ∨ B
1 0 1	35		D ∨ A
1 1 0	36		D ∨ Q
1 1 0 0 0 0	60	EX-OR	A ∀ Q
0 0 1	61		A ∀ B
1 0 1	65		D ∀ A
1 1 0	66		D ∀ Q
1 1 1 0 0 0	70	EX-NOR	$\overline{A \forall Q}$
0 0 1	71		$\overline{A \forall B}$
1 0 1	75		$\overline{D \forall A}$
1 1 0	76		$\overline{D \forall Q}$
1 1 1 0 1 0	72	INVERT	$\overline{Q}$
0 1 1	73		$\overline{B}$
1 0 0	74		$\overline{A}$
1 1 1	77		$\overline{D}$
1 1 0 0 1 0	62	PASS	Q
0 1 1	63		B
1 0 0	64		A
1 1 1	67		D
0 1 1 0 1 0	32	PASS	Q
0 1 1	33		B
1 0 0	34		A
1 1 1	37		D
1 0 0 0 1 0	42	"ZERO"	0
0 1 1	43		0
1 0 0	44		0
1 1 1	47		0
1 0 1 0 0 0	50	MASK	$\overline{A}$ ∧ Q
0 0 1	51		$\overline{A}$ ∧ B
1 0 1	55		$\overline{D}$ ∧ A
1 1 0	56		$\overline{D}$ ∧ Q

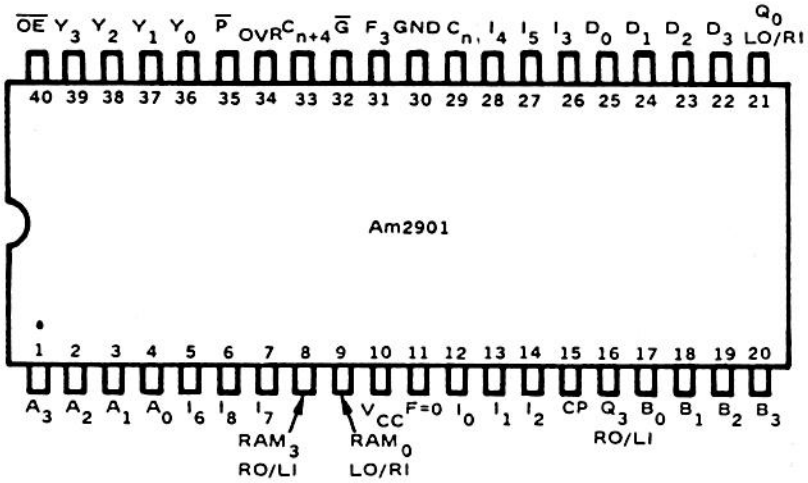
FIGURE 6. ALU LOGIC MODE FUNCTIONS

I_5 I_4 I_3 I_2 I_1 I_0	OCTAL I_{543},I_{210}	C_n = 0		C_n = 1	
		GROUP	FUNCTION	GROUP	FUNCTION
0 0 0 0 0 0	0 0	ADD	A+Q	ADD PLUS ONE	A+Q+1
0 0 1	0 1		A+B		A+B+1
1 0 1	0 5		D+A		D+A+1
1 1 0	0 6		D+Q		D+Q+1
0 0 0 0 1 0	0 2	PASS	Q	INCREMENT	Q+1
0 1 1	0 3		B		B+1
1 0 0	0 4		A		A+1
1 1 1	0 7		D		D+1
0 0 1 0 1 0	1 2	DECREMENT	Q−1	PASS	Q
0 1 1	1 3		B−1		B
1 0 0	1 4		A−1		A
0 1 0 1 1 1	2 7		D−1		D
0 1 0 0 1 0	2 2	1's COMP.	−Q−1	2's COMP.	−Q
0 1 1	2 3		−B−1		−B
1 0 0	2 4		−A−1		−A
0 0 1 1 1 1	1 7		−D−1		−D
0 0 1 0 0 0	1 0	SUBTRACT (1's COMP)	Q−A−1	SUBTRACT (2's COMP)	Q−A
0 0 1	1 1		B−A−1		B−A
1 0 1	1 5		A−D−1		A−D
1 1 0	1 6		Q−D−1		Q−D
0 1 0 0 0 0	2 0		A−Q−1		A−Q
0 0 1	2 1		A−B−1		A−B
1 0 1	2 5		D−A−1		D−A
1 1 0	2 6		D−Q−1		D−Q

FIGURE 7. ALU ARITHMETIC MODE FUNCTIONS

16714-9

CONNECTION DIAGRAM
TOP VIEW



NOTE: PIN 1 IS MARKED FOR ORIENTATION

FINAL DATA. THE PRELIMINARY DATA SHEET PRINTED 3-75 HAD PINS 5 AND 6 INTERCHANGED AND PINS 12 AND 14 INTERCHANGED WHEN COMPARED WITH THE ABOVE CONNECTION DIAGRAM.

16714-10

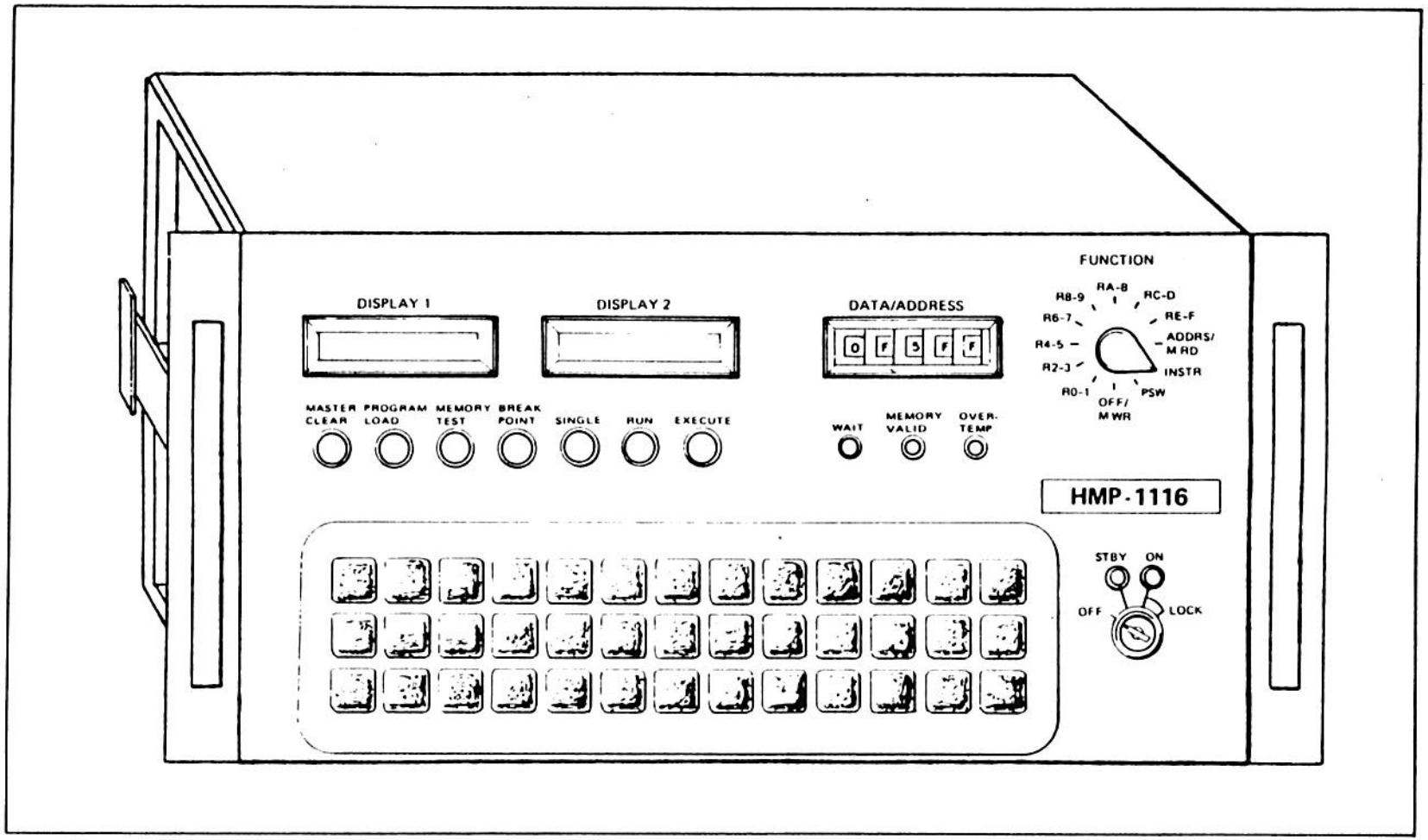
ORDERING INFORMATION

PACKAGE TYPE	TEMPERATURE RANGE	ORDER NUMBER
HERMETIC DIP	0°C TO +70°C	AM2901DC
DICE	0°C TO +70°C	AM2901XC
HERMETIC DIP	-55°C TO +125°C	AM2901DM
DICE	-55°C TO +125°C	AM2901XM

FURTHER INFORMATION

IF ADDITIONAL PRELIMINARY TECHNICAL INFORMATION IS REQUIRED, CONTACT THE NEAREST AMD SALES OFFICE OR CALL THE FACTORY IN SUNNYVALE, CALIFORNIA AND ASK FOR

JOHN SPRINGER
MARKETING MANAGER
BIPOLAR MICROPROCESSOR CIRCUITS
(408) 732-2400 OR TOLL FREE FROM OUTSIDE CALIFORNIA (800) 538-7904 OR 538-7989



HMP-1116 Maintenance Panel

HMP-1116 MAINTENANCE PANEL OPERATION

A. TURN-ON PROCEDURE

1. Turn Key Switch to ON
2. All pushbuttons (MEMORY TEST, PROGRAM LOAD, BREAKPOINT, SINGLE and RUN) should be off (unlit)
3. Press MASTER CLEAR

B. SPECIFYING A MEMORY ADDRESS

1. All pushbuttons off, FUNCTION SWITCH to ADRS/M RD
2. Enter desired address in the DATA/ADDRESS switches
3. Press EXECUTE; DISPLAY 1 = Program Status Register,
   DISPLAY 2 = Location Counter

C. WRITING TO MEMORY

1. Select address to be written to by performing steps 1 thru 3 of "Specifying a Memory Address"
2. SINGLE on, FUNCTION SWITCH to OFF/M WR
3. Enter desired data in the DATA/ADDRESS switches
4. Press EXECUTE; DISPLAY 1 = Address + 2 written to,
   DISPLAY 2 = Data written
5. To write into sequential memory locations, repeat steps 3 and 4

D. READING FROM MEMORY

1. Select address to be read from by performing steps 1 thru 3 of "Specifying a Memory Address"
2. SINGLE on
3. Press EXECUTE; DISPLAY 1 = Address + 2 read from,
   DISPLAY 2 = Data read
4. To read from sequential memory locations, repeat step 3

E. READ GENERAL REGISTER OR PSW

1. All pushbuttons off
2. Select pair of Registers or PSW on FUNCTION SWITCH
3. Press EXECUTE; DISPLAY 1 = Even numbered Register or Program Status Reg.
   DISPLAY 2 = Odd numbered Register or Location Counter

F. READ FULLWORD (INSTRUCTION) FROM MEMORY

1. Select address to be read from by performing steps 1 thru 3 of "Specifying a Memory Address"
2. FUNCTION SWITCH to INSTR
3. Press EXECUTE; DISPLAY 1 = Contents of memory location specified by address
   DISPLAY 2 = Contents of memory location specified by address+2

G. RUN A PROGRAM

1. Specify the start address of the program by performing steps 1 thru 3 of "Specifying a Memory Address"
2. RUN on, FUNCTION SWITCH to any position except ADRS/M RD or OFF/M WR
3. Press EXECUTE

H. RUN A PROGRAM IN SINGLE STEP

1. Perform steps 1 and 2 of "Run a Program"
2. SINGLE on
3. Press EXECUTE; DISPLAYS 1 and 2 will contain whatever the FUNCTION SWITCH specified (Register Pair, Updated PSW or next Instruction)
4. To execute successive instructions repeat step 3. The FUNCTION SWITCH can be changed between EXECUTE switch actions to select what will be displayed after the next instruction is executed.

I. RUN A PROGRAM IN BREAKPOINT

1. Perform steps 1 and 2 of "Run a Program"
2. BREAKPOINT on, enter instruction address in DATA/ADDRESS switches
3. Press EXECUTE; DISPLAY 1 = Specified breakpoint address
   DISPLAY 2 MSD = 3 (if breakpoint was reached)
   DISPLAY 2 LSDs = Instruction at breakpoint address

NOTE: Displays described in step 3 will only occur if the breakpoint address is reached; the MSD of DISPLAY 2 is the breakpoint indicator. Processor will stop before instruction at breakpoint address is executed.

J. PROGRAM LOAD USING THE 50-SEQUENCE LOADER

1. Perform operations required to prepare loader device (i.e. Put tape on Mag Tape Unit (MTU) and place MTU On-Line.)
2. PROGRAM LOAD on, all other pushbuttons off
3. Set DATA/ADDRESS switches to 'XXYY' where 'XX' is the loader device address, and 'YY' is command for device (i.e. '85A1' for MTU)
4. Press MASTER CLEAR
5. PROGRAM LOAD off, RUN on
6. Press EXECUTE; Program will load from specified loader device. Remainder of procedure will be determined by the program loaded.

K. PROGRAM LOAD OPTION USING DEVICE ADDRESS '00'

1. Perform operations required to prepare loader device
2. PROGRAM LOAD and RUN on, all other pushbuttons off, FUNCTION SWITCH to PSW
3. Set DATA/ADDRESS switches to '000X' to specify loader device (see table below)
4. Press MASTER CLEAR; Program will load and run. Remainder of procedure will be determined by the program loaded.

'X'	DEVICE SPECIFIED
0	Device Status Check
1	Mag Tape
2	
3	Floppy Disc
4	Disc Load
5	
6	
7	

MEMORY TEST

The Memory Test consists of four tests; 1) DATA, 2) GALPAT, 3) ADDRESS, and 4) $\overline{\text{ADDRESS}}$.

The DATA test writes a fixed data pattern in all tested memory locations. The data pattern to be used is initially in DISPLAY 2.

The GALPAT test generates a pseudo-random data pattern from the value initially in DISPLAY 2. This text writes to a memory location and then changes adjacent locations in memory to see if the tested location is affected.

The ADDRESS test writes the address of a location into that location. For example, load location '0123' with '0123'.

The $\overline{\text{ADDRESS}}$ test writes the one's complement of the address of a location into that location. For example, load location '0123' with 'FEDC'.

Executing the memory test is broken into five steps; 1) enter the test mode, 2) select portion of memory to be tested, 3) select test data (for DATA and GALPAT tests only), 4) select and start test (either DATA, GALPAT, ADDRESS or $\overline{\text{ADDRESS}}$) and 5) run test in either the "Continuous" or the "Stop on Error" mode.

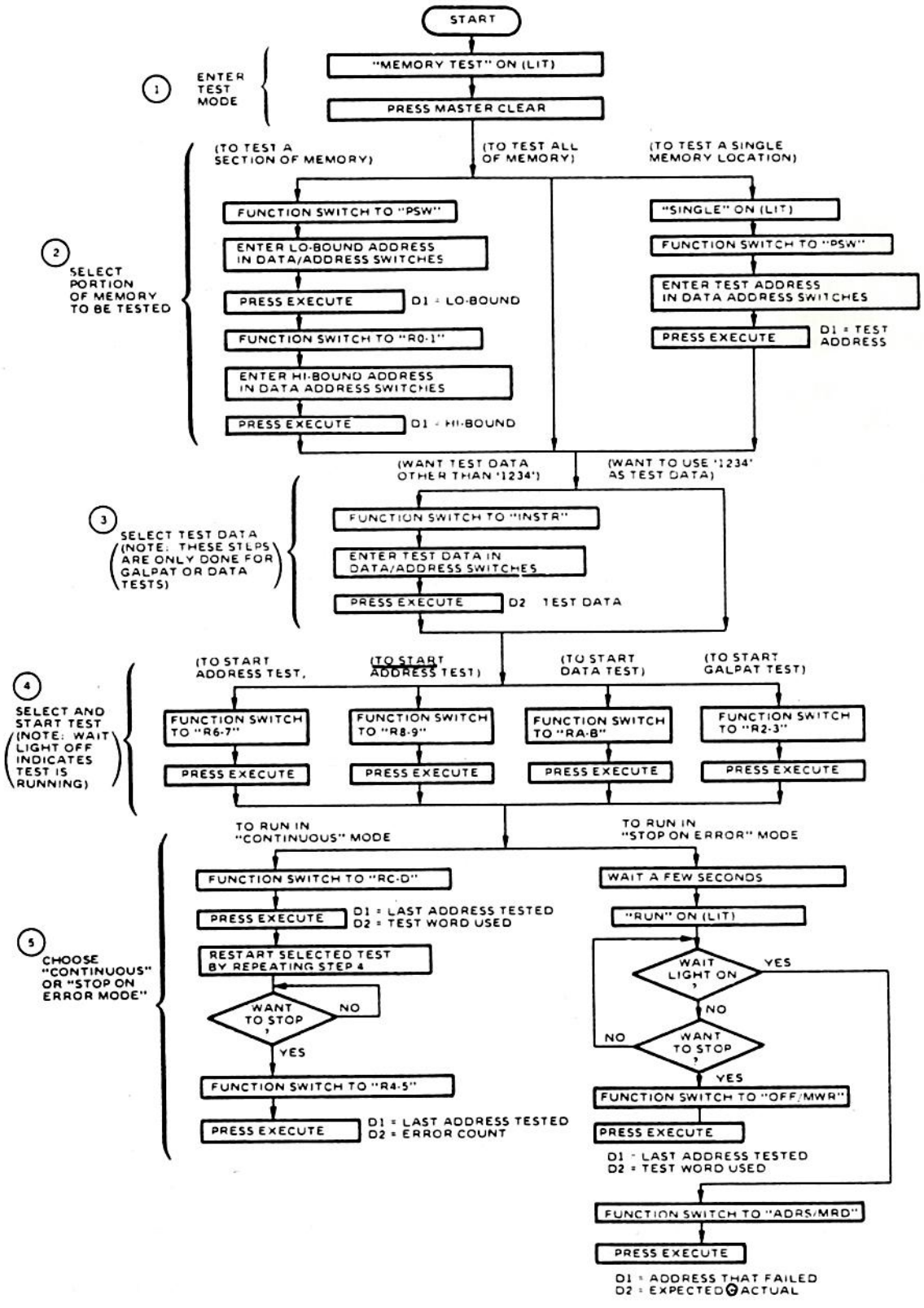
When you enter the test mode, DISPLAY 1 will indicate the size of memory as shown in the table below:

CONTENTS OF DISPLAY 1	NUMBER OF MEMORY CARDS	SIZE OF MEMORY (1K = 1,024 Halfwords)
10000	1	32K
20000	2	64K
30000	3	96K
00000	4	128K

The "Continuous" mode continually tests the selected memory locations and maintains an error count.

The "Stop on Error" mode tests the selected memory locations until the first error occurs and then halts.

The memory test performs the following operation on each location tested, "read-check-write". Therefore, on the first pass through memory using new test data every "read-check-write" will result in an error being detected. This is why you wait a few seconds when running the "Stop on Error" mode, you are allowing time enough for the new test pattern to fill memory before allowing a stop on error to occur. This is also why you stop and restart in the "Continuous" mode, this is to clear out the error count after the new data has filled memory.



HMP-1116 Memory Test

2.0 PROGRAMMING IN MACHINE LANGUAGE

2.1 Introduction. A program is a set of software instructions. These software instructions are arranged in a logical manner in order to perform the data manipulations necessary to solve a problem. The software instructions are loaded into consecutive memory locations. The Program Status Word (PSW) controls which instruction will be executed next and contains the status of the program. The format of the PSW is:

0	1 7	8 9	10 11	12 15	16 31
Wait	Status	Program Bank	Operand Bank	Condition Code	Location Counter

When the PSW is loaded the Wait bit (PSW bit 0) will determine if the software program will be executed. Wait = 1 means halt. Wait = 0 means execute the software program.

The Status bits (PSW bits 1 through 7) control interrupts.

The Condition Code consists of four flags; the Carry flag (PSW bit 12), the Overflow flag (PSW bit 13), the Greater Than Zero flag (PSW bit 14) and the Less Than Zero flag (PSW bit 15). The four flags are normally abbreviated as C, V, G and L, respectively. The Condition Code is set by a previously executed software instruction.

The computer has an 18-bit memory address. Since the basic word length in the computer is 16 bits, special control is provided for the two MSBs of the memory address. These two bits are specified by either bits 8 and 9, or bits 10 and 11 of the PSW. They are called the memory bank address.

The next instruction to be executed is located in the bank specified by the Program Bank address (PSW bits 8 and 9), and the location within that bank is specified by the Location Counter (PSW bits 16 through 31). If a software instruction is using a memory location as an operand, the bank that operand is located in is specified by the Operand Bank address (PSW bits 10 and 11).

Memory is organized in 16-bit words. When the processor accesses memory 16 bits of data are transferred. The processor can handle a word from memory as a 16-bit halfword or as two 8-bit bytes. To aid the programmer in handling data as bytes, the processor assigns two memory addresses to each memory location. The two addresses assigned to a location are an even/odd pair of numbers. For example, the first halfword in memory is assigned memory addresses '00000' and '00001'. Using either address will result in accessing the same 16 bits in memory. If the programmer is using the first halfword in memory as a 16-bit operand, he would refer to it as memory location '00000'. If the programmer is using the first halfword in memory as two 8-bit operands, he would refer to the 8 MSBs as '00000' and to the 8 LSBs as '00001'.

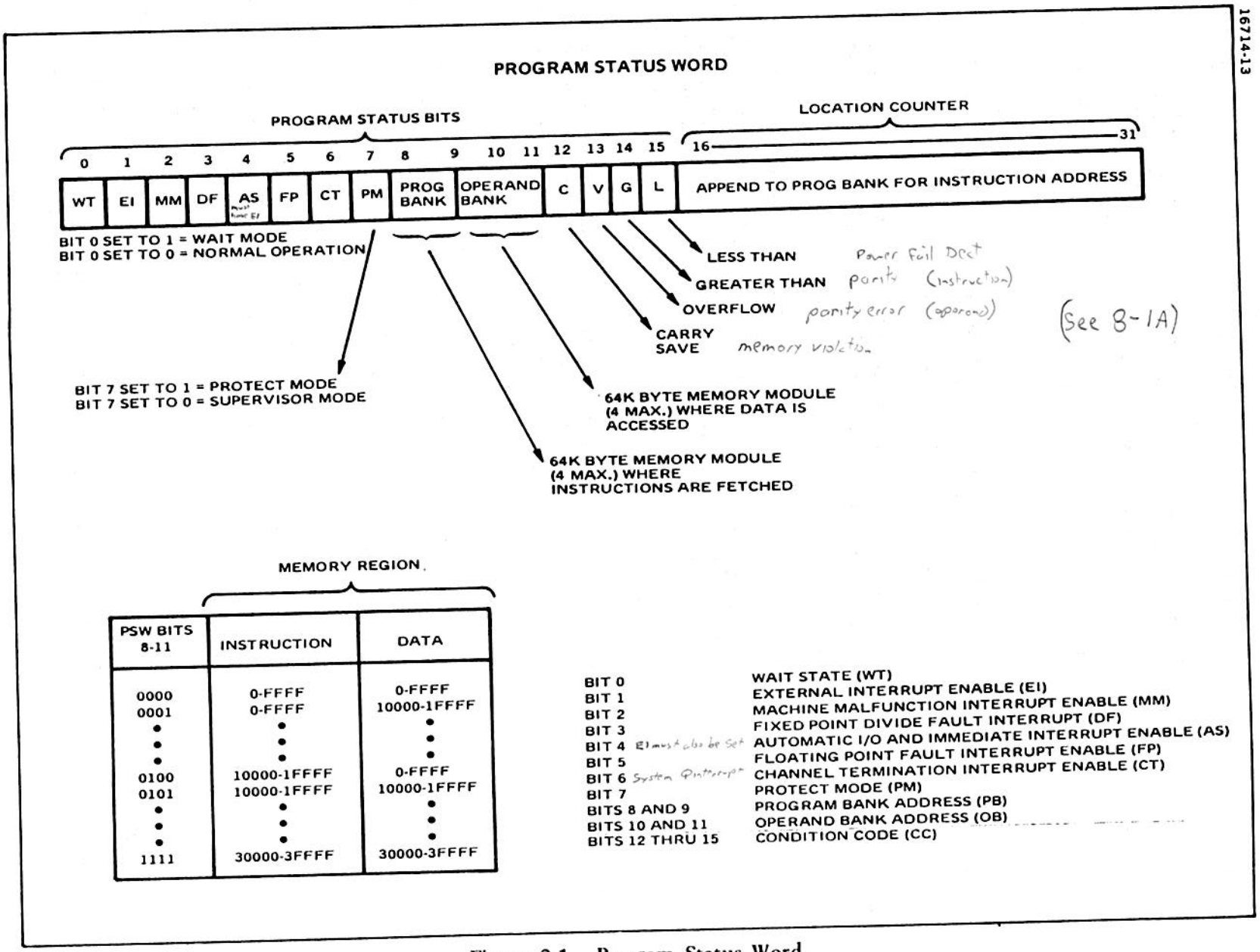
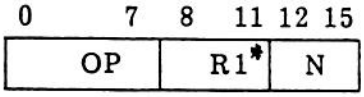


Figure 2-1. Program Status Word

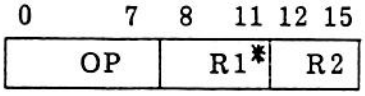
2.2 Software Instruction Formats. A software instruction is either a 16-bit halfword or a 32-bit fullword. The instruction is divided into groups of bits called fields. There are four software instruction formats. The 8 MSBs of all instructions is called the op-code (OP). The OP field specifies the instruction format and the operation to be performed. These halfword or fullword instructions are called the computer's machine language.

SHORT FORMAT (SF)



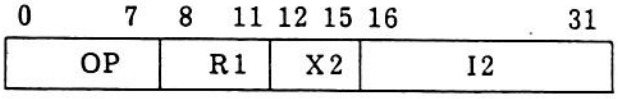
The operands in an SF instruction are a General Register in File 1 specified by the R1 field and an immediate operand specified by the N field. A 16-bit immediate operand is created by setting the 3 MSDs (Most Significant Hex Digits) to zero and using N as the LSD (Least Significant Hex Digit). The General Register specified by R1 is the destination (where the result is placed).

REGISTER TO REGISTER (RR)



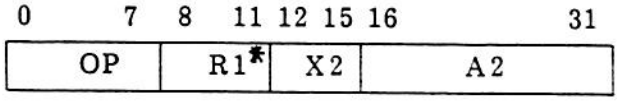
The operands in an RR instruction are two General Registers in File 1 specified by the R1 and R2 fields. The General Register specified by R1 is normally the destination.

REGISTER IMMEDIATE (RI)



The operands in an RI instruction are a General Register in File 1 specified by the R1 field and an immediate operand dictated by the I2 and X2 fields. The 16-bit immediate operand is calculated by adding the 16-bit I2 field to the General Register specified by the X2 field. The register specified by the X2 field is called an index register. If the X2 field is zero, I2 alone is used as the immediate operand; in other words, no indexing will occur. The General Register specified by R1 is the destination.

REGISTER INDEXED MEMORY (RX)



The operands in an RX instruction are a General Register in File 1 specified by the R1 field and a memory location dictated by the A2 and X2 fields. The address of the memory location to be used is calculated by adding the 16-bit A2 field to the General Register (Index Register) specified by X2. If the X2 field is zero, no indexing will occur. Normally the General Register specified by R1 is the destination.

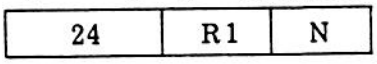
* In some branch instructions R1 is replaced by the M1 field. The M1 field is compared bit for bit with the Condition Code to determine if the branch is to be taken. When an instruction branches, the instruction specifies the location of the next instruction to be executed.

2.3 Basic Instruction Set. For a complete description of the software instruction set, see Chapter 2 of the Technical Manual.

In the following examples the notation used to describe the operation performed is:

(Reg. X)	is read "the contents of General Register X"
PSW(8:9)	is read "PSW bits 8 through 9"
'1234'	is a Hexadecimal quantity
['2000']	is read "the contents of memory location 2000"
←——	is read "is replaced with"

LOAD IMMEDIATE SHORT (LIS)



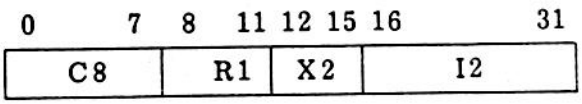
(SF)

This instruction causes the General Register specified by R1 to be loaded with '000N'.

Example: | 24 | 7 | 2 | (Reg. 7)←—'0002'

Load General Register 7 with '0002'.

LOAD HALFWORD IMMEDIATE (LHI)



(RI)

This instruction causes the General Register specified by R1 to be loaded with an immediate field specified by I2 plus the General Register specified by X2.

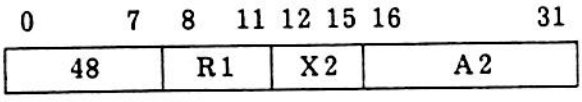
Example: | C8 | 2 | 0 | 0040 | (Reg. 2)←—'0040'

Load General Register 2 with '0040'. This instruction has no indexing.

Example: | C8 | 5 | 1 | 12AB | (Reg. 5)←—'12AB' + (Reg. 1)

Load General Register 5 with the sum of '12AB' and the contents of General Register 1.

LOAD HALFWORD (LH)



(RX)

This instruction causes the General Register specified by R1 to be loaded from a memory location specified by A2 and X2. The bank in memory used is specified by PSW(10:11).

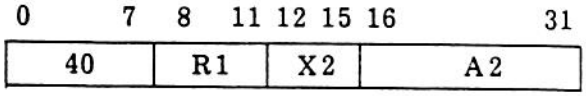
Example: | 48 | A | 0 | 1000 | (Reg. A)←—['1000']

Load General Register A from memory location '1000'. No indexing.

Example: | 48 | 0 | A | 2000 | (Reg. 0)←—['2000' + (Reg. A)]

Load General Register 0 from memory. The address of the memory location is calculated as '2000' plus the contents of General Register A.

STORE HALFWORD (STH)



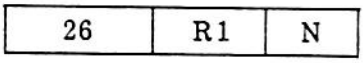
(RX)

This instruction cause the General Register specified by R1 to be stored in a memory location specified by A2 and X2. The bank in memory used is specified by PSW(10:11).

Example: | 40 | E | F | 02D0 | ['02D0' + (Reg. F)]←—(Reg. E)

Store General Register E in memory. The memory location is determined by adding the index register (General Register F) to '02D0'.

ADD IMMEDIATE SHORT (AIS)



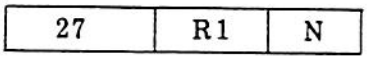
(SF)

This instruction causes the immediate operand '000N' to be added to the value currently in the General Register specified by R1.

Example: | 26 | B | 5 | (Reg. B)←—(Reg. B) + '0005'

Add '0005' to the contents of General Register B and store the result in General Register B.

SUBTRACT IMMEDIATE SHORT (SIS)



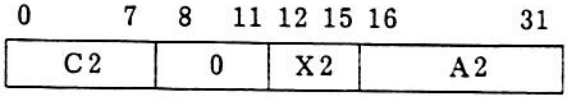
(SF)

This instruction causes the immediate operand '000N' to be subtracted from the value currently in the General Register specified by R1.

Example: | 27 | C | F | (Reg. C)←—(Reg. C) - '000F'

Subtract '000F' from the contents of General Register C and store the result in General Register C.

LOAD PROGRAM STATUS WORD (LPSW)



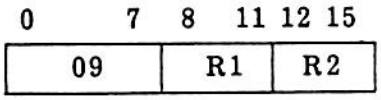
(RX)

This instruction causes the PSW to be loaded from memory. The memory location is specified by the X2 and A2 fields. The R1 field is set to zero and has no effect on this instruction.

Example: | C2 | 0 | 0 | F008 | PSW(0:15)←—['F008']
PSW(16:31)←—['F00A']

Load the PSW from two consecutive memory locations. The first memory location is specified by 'F008' and it is loaded into the 16 MSBs of the PSW. The second memory location is 'F00A' and it is loaded into the 16 LSBs of the PSW.

COMPARE HALFWORD (CHR)



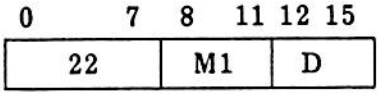
(RR)

This instruction algebraically compares the 16-bit operands in the General Registers specified by R1 and R2. The data in the two registers is unchanged as a result of this instruction. The G and L flags in the Condition Code indicate if the contents of the register specified by R1 are greater or less than the contents of the register specified by R2.

Example: | 09 | 4 | 9 | (Reg. 4) : (Reg. 9) (: means compared to)

This instruction causes the contents of General Register 4 to be compared to the contents of General Register 9. The contents are treated as two's complement operands. Both operands are unchanged. The G flag indicates that register 4 was greater than register 9. The L flag indicates that register 4 was less than register 9.

BRANCH ON FALSE BACKWARDS SHORT (BFBS)



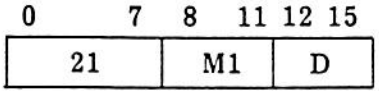
(SF)

The M1 field is compared bit for bit with the Condition Code. If none of the bits set in M1 match flags set in the Condition Code the condition is called false and a branch will be taken. When the branch is taken the branch address is calculated as the location of the BFBS instruction minus two times the D field. This is called relative addressing because the branch address is calculated relative to the location of the branch instruction.

Example: | 22 | 2 | 4 | If G flag not set, PSW(16:31)←PSW(16:31) - '0008'
If G flag is set, PSW(16:31)←PSW(16:31) + '0002'
(No branch)

The M1 (mask) in binary is 0010, and it is compared bit for bit with the flags CVGL in the Condition Code. If a flag set to 1 in the Condition Code corresponds to a 1 set in the mask, the condition is called true. Since only one bit is set in the mask field in this instruction only one flag will determine if the branch is to be taken, the G flag. If the G flag is not set, then the condition is false and the branch address is calculated as the Location Counter minus two times the D field ('4'). If the G flag is set, then the condition is true and the Location Counter is incremented by two to point at the next consecutive instruction. (Note: In all instructions the Location Counter is automatically incremented to point at the next consecutive instruction.)

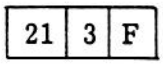
BRANCH ON TRUE FORWARDS SHORT (BTFS)



(SF)

The M1 field is compared bit for bit with the Condition Code. If any of the bits set in M1 match flags set in the Condition Code the condition is called true and a branch will be taken. The relative branch address is calculated as the Location Counter plus two times the D field.

Example:



If G or L set, PSW(16:31)←PSW(16:31) + '001E'
If G and L not set, PSW(16:31)←PSW(16:31) + '0002'
(No branch)

Either the G or L flags being set will be a true condition and a relative branch will be calculated as the Location Counter plus '001E' (two times '00F').

LSD \ MSD	0	1	2	3	4	5	6	7	9	B	C	D	E	F
0		A	X		X	X	X	X	H	X	X	X		
1	X	A	X		X	A	B		H	X	X	X	Y	X
2	X	A	X		X	A			X	P	Y	X	X	P
3	X		X		X	A			X	Q	D	X	X	Q
4	C		A		C		J	A	X	O	C	C	X	O
5	E	E	A		E	E	J	A	Y		E	Z		
6	C		B		C		K	A	Z		C	Z	H	
7	C		B		C		K	A	Z		C	Z	H	
8	A	A	L	L	A	A	L	L	Z		A	Z	!	X
9	F	F	M	M	F	F	M	M	Z		F	Z	!	X
A	B	B	N	N	B	B	N	N	Z		B	Z	C	X
B	B	B	N	N	B	B	N	N	Z		B	Z	C	X
C	X	X	N	N	X	X	N	N	X		H	X	H	
D	X	X	N	N	X	X	N	N	Z		H	Z	H	
E	B		X		B				Z		!	Z	!	
F	B		X		B				Z		!	Z	!	

TO FIND WHAT EFFECT AN INSTRUCTION HAS ON THE CONDITION CODE, USE THE TWO HEX DIGITS OF THE OP CODE TO FIND THE APPROPRIATE ENTRY IN THE MAP. THE LETTER FOUND IN THE MAP CORRESPONDS TO ONE OF THE DESCRIPTIONS BELOW.

A.

C	V	G	L	
X	X	0	0	OPERAND IS ZERO
X	X	0	1	OPERAND IS LESS THAN ZERO
X	X	1	0	OPERAND IS GREATER THAN ZERO

B.

C	V	G	L	
X	X	0	0	RESULT IS ZERO
X	X	0	1	RESULT IS LESS THAN ZERO
X	X	1	0	RESULT IS GREATER THAN ZERO
X	1	X	X	ARITHMETIC OVERFLOW
1	X	X	X	CARRY / BARROW

C.

C	V	G	L	
X	X	0	0	LOGICAL RESULT IS ZERO
X	X	0	1	LOGICAL RESULT IS NOT ZERO
X	X	1	0	

D.

C	V	G	L	
X	X	0	0	NONE OF THE BITS OF THE RESULT SET
X	X	0	1	BIT 0 OF THE RESULT SET
X	X	1	0	ONE OR MORE OF BITS 1 THRU 15 OF THE RESULT SET

E.

C	V	G	L	
X	X	0	0	FIRST OPERAND EQUAL TO SECOND OPERAND
X	X	0	1	FIRST OPERAND NOT EQUAL TO SECOND OPERAND
X	X	1	0	
1	X	X	X	FIRST OPERAND LESS THAN SECOND OPERAND
0	X	X	X	FIRST OPERAND EQUAL TO OR GREATER THAN SECOND OPERAND

O. V-FLAG INDICATES Y = Z = 0 OR |Y| > |Z|

P. V-FLAG INDICATES RESULT > '7FFF FFFF'

Q. V-FLAG INDICATES |X| < |Y|

X. FLAGS UNCHANGED

Y. NEW FLAGS DETERMINED BY PSW LOADED

Z. I/O INSTRUCTION V FLAG NORMALLY INDICATES TIMEOUT (FALSE SYNC) 35 μSEC

F.

C	V	G	L	
X	X	0	0	FIRST OPERAND EQUAL TO SECOND OPERAND
X	X	0	1	FIRST OPERAND LESS THAN SECOND OPERAND
X	X	1	0	FIRST OPERAND GREATER THAN SECOND OPERAND
1	X	X	X	FIRST OPERAND LESS THAN SECOND OPERAND
0	X	X	X	FIRST OPERAND EQUAL TO OR GREATER THAN SECOND OPERAND

G.

C	V	G	L	
0	X	0	0	FIRST OPERAND EQUALS SECOND OPERAND
1	X	0	1	FIRST OPERAND LESS THAN SECOND OPERAND
1	X	1	0	
0	X	0	1	FIRST OPERAND GREATER THAN SECOND OPERAND
0	X	1	0	

H.

C	V	G	L	
	0	0	0	RESULT IS ZERO
	0	0	1	RESULT IS NOT ZERO
	0	1	0	RESULT IS NOT ZERO
0				LAST BIT THAT WAS SHIFTED OUT WAS A ZERO
1				LAST BIT THAT WAS SHIFTED OUT WAS A ONE

I.

C	V	G	L	
	0	0	0	RESULT IS ZERO
	0	0	1	RESULT IS LESS THAN ZERO
	0	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

J.

C	V	G	L	
0	1	0	0	LIST OVERFLOW
0	0	0	0	ELEMENT ADDED SUCCESSFULLY

K.

C	V	G	L	
0	1	0	0	LIST WAS ALREADY EMPTY
0	0	0	0	LIST IS NOW EMPTY
0	0	1	0	LIST IS NOT YET EMPTY

L.

C	V	G	L	
X	0	0	0	ZERO
X	0	0	1	LESS THAN ZERO
X	0	1	0	GREATER THAN ZERO
X	1	0	0	EXPONENT UNDERFLOW

M.

C	V	G	L	
0	X	0	0	OPERANDS EQUAL
1	X	0	1	FIRST LESS THAN SECOND
0	X	1	0	FIRST GREATER THAN SECOND

N.

C	V	G	L	
0	0	0	0	RESULT IS ZERO
0	0	0	1	RESULT LESS THAN ZERO
0	0	1	0	RESULT GREATER THAN ZERO
0	1	0	1	OVERFLOW (NEGATIVE)
0	1	1	0	OVERFLOW (POSITIVE)
0	1	0	0	UNDERFLOW
1	1	X	X	DIVISOR EQUAL TO ZERO*

* DIVISION ONLY

16714-14

CHANGING PSW PROCEDURES

NOTE: There are times when you cannot execute your Program in the HMP-1116 due to a PSW that causes your Program to not run, i.e. OPERAND BANK ADDRESS set to wrong bank. There are ways to insure this does not happen.

1. Write zeros into the entire memory prior to loading your program (MEM Test with '0000' as test data).

2. Always check your PSW prior to loading your program to insure proper Bank address (PSW10-11).

3. If you do not do these first two steps and determine that the PSW bits 10-11 are not set properly after you loaded your program, you can change your PSW without destroying your program.

   Step 1: Load the instruction 'E100' into a memory location not being used, for example address '1000'.

   Step 2: Load your desired PSW into memory location '009A' (normally '0000').

   Step 3: Load the address of the instruction in step 1 into memory location '009C', in this example '1000'.

   Step 4: Execute the instruction loaded in step 1.

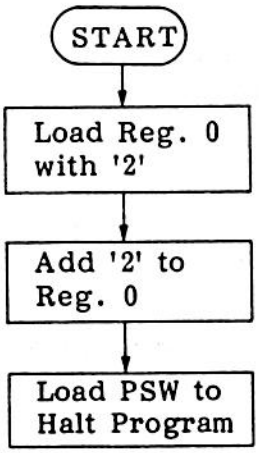
   You now have changed your PSW by executing a supervisor call instruction.

2.4 Sample Programs in Machine Language

PROGRAM #1

Write a program that will add 2 plus 2.

The first step in writing a program is making a flowchart of the software instructions required to solve the given problem. A flowchart is extremely helpful in organizing your approach to solving a problem and in explaining to others what your program is doing.



A Load Immediate Short (LIS) can be used to place the first '2' in General Register 0.

An Add Immediate Short (AIS) can now be used to add '2' to the '2' in General Register 0.

When a program has completed solving a problem it must either halt or go to another program.

MEMORY LOCATION	MEMORY CONTENTS	INSTRUCTION MNEMONIC	COMMENTS
1000	2402	LIS	(Reg. 0)←'0002'
1002	2602	AIS	(Reg. 0)←(Reg. 0) + '0002'
1004	C200	LPSW	PSW(0:15)←['1008']
1006	1008		PSW(16:31)←['100A']
1008	8000	- - -	Wait bit set will halt program
100A	1000	- - -	New Location Counter

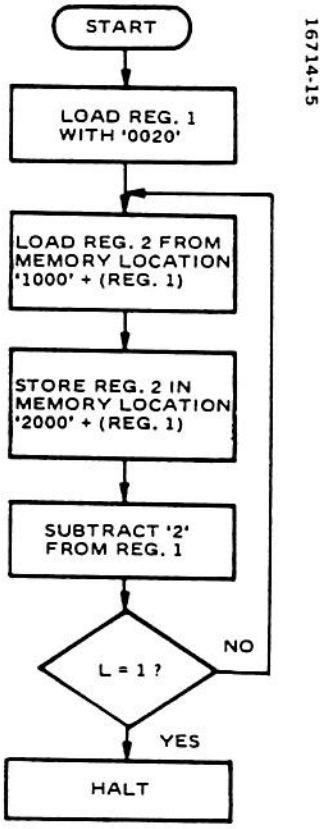
PROGRAM #2

Write a program to transfer memory locations '1000' through '1020' to memory locations '2000' through '2020', respectively.

A total of 17 memory locations are to be transferred. This could be done with 17 load and 17 store instructions, but that would be rather cumbersome. A simpler solution is to use indexed memory addressing and branching so that one load and one store instruction can be used for all 17 transfers.

In this program I initialize the index to '0020' and decrement it by '0002' after each data transfer. The first transfer will be of memory location '1020' into '2020', and the last data transfer will be '1000' into '2000'. After the last transfer the index register will contain zero, when it is decremented by '0002' it will become 'FFFE'. Since the result is a negative number the L flag will be set. Therefore, I will use the L flag as an indicator that the data transfers have been completed.

Flowchart for Program #2



Initialize the index register to '0020' Use LHI instruction.

Load General Register 2 from the memory location specified by '1000' plus the contents of General Register 1 (the index register). Use LH.

Store General Register 2 in the memory location specified by '2000' plus the contents of General Register 1 (the index register). Use STH.

Subtract '2' from the index register. Use SIS.

If the L flag is set all transfers are finished so halt. Since we want to branch backwards when the L flag is not set we use a BFBS instruction.

Use LPSW.

MEMORY LOCATION	MEMORY CONTENTS	INSTRUCTION MNEMONIC	COMMENTS
1000	C810	LHI	(Reg. 1) ← '0020'
1002	0020		
1004	4821	LH	(Reg. 2) ← ['1000' + (Reg. 1)]
1006	1000		
1008	4021	STH	['2000' + (Reg. 1)] ← (Reg. 2)
100A	2000		
100C	2712	SIS	(Reg. 1) ← (Reg. 1) - '0002'
100E	2215	BFBS	If L = 0, branch to '1004'
1010	C200	LPSW	PSW(0:15) ← ['1014']
1012	1014		PSW(16:31) ← ['1016']
1014	8000	- - -	Halt Program
1016	1000	- - -	

2.5 Basic Instruction Set Summary

I. Load Instructions

A. Load Byte - LB, LBR
B. Load Complement Short - LCS
C. Load Halfword - LH, LHI, LHR, LH0, LH1, LH2, LH3
D. Load Immediate Short - LIS
E. Load Multiple - LM

II. Store Instructions

A. Store Byte - STB, STBR
B. Store Halfword - STH, STH0, STH1, STH2, STH3
C. Store Multiple - STM

III. PSW Control

A. Exchange Program Status - EPSR
B. Exchange Operand Bank Address - EPOR
C. Exchange Program Address - EPPR
D. Load Program Status Word - LPSW

IV. Fixed-Point Arithmetic

A. Add Halfword - AH, AHI, AHR, AHM
B. Add Immediate Short - AIS
C. Add with Carry Halfword - ACH, ACHR
D. Divide Halfword - DH, DHR
E. Multiply Halfword - MH, MHR
F. Multiply Halfword Unsigned - MHU, MHUR
G. Subtract Halfword - SH, SHI, SHR
H. Subtract Immediate Short - SIS
I. Subtract with Carry Halfword - SCH, SCHR

V. Shift and Rotate

A. Rotate Left Logical - RLL
B. Rotate Right Logical - RRL
C. Shift Left Arithmetic - SLA, SLHA
D. Shift Left Logical - SLL, SLHL, SLLS
E. Shift Right Arithmetic - SRA, SRHA
F. Shift Right Logical - SRL, SRHL, SRLS

VI. Logical

A. And Halfword - NH, NHI, NHR
B. Compre Halfword - CH, CHI, CHR
C. Compare Logical Halfword - CLH, CLHI, CLHR
D. Compare Logical Byte - CLB
E. Exclusive-Or Halfword - XH, XHI, XHR
F. Or Halfword - OH, OHI, OHR
G. Test Halfword Immediate - THI

VII. Branch

A. Branch and Link - BAL, BALR
B. Branch on False - BFC, BFCR, BFBS, BFFS
C. Branch on True - BTC, BTCR, BTBS, BTFS
D. Branch on Index High - BXH
E. Branch on Index Low or Equal - BXLE

VIII. Input/Output

A. Acknowledge Interrupt - AI, AIR
B. Autoload - AL
C. Output Command - OC, OCR
D. Read - RB, RBR, RD, RDR, RH, RHR
E. Sense Status - SS, SSR
F. Write - WB, WBR, WD, WDR, WH, WHR

IX. List Processing

A. Add to List - ABL, ATL
B. Remove from List - RBL, RTL

X. Miscellaneous

A. Exchange Byte - EXBR
B. Supervisor Call - SVC
C. Simulate Interrupt - SINT

Note on how to read the basic instruction set summary:

X. Function - Instruction Mnemonic(s)

2.6 General Notes

1. The PSW format is described on Page 2-2.

2. When doing a byte oriented RX instruction, an even address specifies the 8 MSBs of a halfword in memory, and an odd address specifies the 8 LSBs of a halfword in memory.

3. BTFS op-code is '21'. BTBS op-code is '20'.

3.0 PROCESSOR BLOCK DIAGRAM DESCRIPTION

Introduction

Figure 3-1 is the Detailed Functional Block Diagram of the HMP-1116. Each block on Figure 3-1 has a number in the lower right corner. This is a reference to the Functional Schematics in Chapter 6. For example, the Rom Address Register on Figure 3-1 has a "2" in the lower right corner indicating that the RAR is on page 2 of the Functional Schematics. The three letter mnemonics in the bottom of each block indicate the circuit board in the HMP-1116 chassis that contains that function.

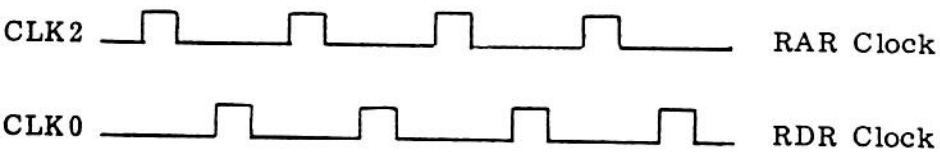
The basic HMP-1116 consists of a Microsequencer, 16-bit Arithmetic Logic Unit, Registers and Control Logic, Memory and Input/Output (I/O).

3.1 Microsequencer and Sequence Control. The Microsequencer controls the following major functions:

1) Software Instruction Decode
2) Arithmetic Logic Unit
3) Data Movement between Registers
4) Processor Memory Operations
5) I/O Operations

The microsequencer consists of the RAR, Microprogram ROM, RDR, RAR Branch and Load Control and Multiple Clock Cycle Control.

The RAR normally increments every 200ns to provide access to another microinstruction in the Microprogram ROM. The RDR is loaded every 200ns with the new microinstruction. The clocks that control incrementing the RAR and loading the RDR are shown below.



The microprogram consists of 36-bit microinstructions which are arranged in sets within the Microprogram ROM. Each set of microinstructions performs a specific function. Each microinstruction is divided into fields based on unique 4-bit Op-codes (QRD00-03). There are five microinstruction formats:

1) Register to Register (RR)
2) Immediate to Register (RI)
3) Control (CTL)
4) Input/Output (IO)
5) Branch (BR)

3.1.1 RAR Load Operations. The RAR is loaded when a new micro-sequence is required to perform a specific task (i.e. microbranch, RAR specified as destination, software instruction decode, interrupt). The Load Control (XLDRAR) from the RAR Branch and Load Control causes a 12-bit value to be loaded into the RAR. This 12-bit value is a Branch Address (QRD20-31) or A-bus data (XBA04-15). The selection of a Branch Address or A-bus data is also performed by the BAR Branch and Load Control. The Multiple Clock Cycle Control provides extra clock cycles if a microinstruction cannot be executed within the 200ns clock cycle.

3.1.1.1 Microbranch. Branch microinstructions may be conditional or unconditional. A conditional branch means that the FLR (Flag Register) is compared bit by bit to a 4-bit Mask field(QRD16-19) located within the BR microinstruction. If the condition is met a 12-bit Branch Address (QRD20-31) from the Branch microinstruction is loaded into the RAR. There are four types of Branch microinstructions, and each one has a unique 4-bit Op-code. The four types of BR microinstructions are:

1) Branch Condition True; if Mask $\wedge$ FLR = any true then QRD20-31 $\rightarrow$ RAR

2) Branch Condition False; if Mask $\wedge$ FLR = all false then QRD20-31 $\rightarrow$ RAR

3) Branch Indexed False; if Mask $\wedge$ FLR = all false then QRD20-31 + Scratch Reg. $\rightarrow$ RAR

4) Branch and Link; Unconditional, Current RAR $\rightarrow$ Scratch Reg
QRD20-31 $\rightarrow$ RAR

Branch Condition True/Branch Condition False

The RAR Branch and Load Control examines the microinstruction Op-code and compares the FLR and Mask. If the condition is met, the Load RAR (XLDRAR) is generated along with the RAR data select ($\overline{\text{X}}$SRARD) to allow QRD20-31 to be loaded into the RAR.

Branch Indexed False(BIF)

During this microinstruction execution the Mask and FLR are compared. If the condition is false, QRD20-31 are added to the contents of an internal register specified in the BR microinstruction. The sum is then loaded into the RAR from the A-bus. Since the RAR is the destination and an arithmetic operation must be performed in the AM2901, the operation requires an extra clock cycle for completion.

The Branch Address is passed through the Rom Address B-bus Mux and on to the B-bus (12 LSBs). The data then passes through the Byte Manipulator to the direct input of the AM2901. The scratch register specified by the BIF microinstruction is added to the direct input, and the output of the AM2901 is loaded into the RAR via the A-bus. The RAR Branch and Load Control generates XLDRAR and, in order for the A-bus to be selected, it sets RAR Data Select inactive (-XSRARRD = H).

The extra clock cycle required is provided by the Multiple Clock Cycle Control. The RAR Branch and Load Control generates XRDSA when a BIF microinstruction is in the Rom Data Register (RDR). This signal is used by the Multiple Clock Cycle Control to set QSMTH (Single/Multiple Time Hold), and to generate ROM Data Stop ($\overline{\text{X}}$RDSTP). QSMTH inhibits the RAR count by disabling XCNTRAR (RAR Count Enable) and is used by the RAR Branch and Load Control to allow XLDRAR to be generated at the appropriate time.

Branch and Link (BLK)

To execute this microinstruction the current RAR and current FLR are saved in a scratch register specified by the BLK microinstruction, and the Branch Address (QRD20-31) is loaded into the RAR. This microinstruction is an unconditional branch (i.e. there is no Mask:FLR comparison). Since the RAR is the implied source and destination an extra clock cycle is required.

The RAR is gated through the Rom Address B-bus Mux and on to the B-bus 12 LSBs. The FLR (QCSV,QOVF,QG,QL) is gated on to the B-bus 4 MSBs at the same time. This information is passed through the Byte Manipulator to the D input of the AM2901 and is loaded into the scratch register specified by this BLK microinstruction.

The new RAR value QRD20-31 is selected and loaded into the RAR by the RAR Branch and Load Control.

The RAR Branch and Load Control generates XRDSA when decoding the BLK microinstruction. This signal is used by the Multiple Clock Cycle Control to generate $\overline{\text{X}}$RDSTP and QSMTH. QSMTH disables XCNTRAR and allows the RAR Branch and Load Control to generate XLDRAR at the appropriate time.

3.1.1.2 RAR Specified as Destination. A microinstruction may specify the RAR as a destination register. One example of this operation is a return statement. This type of microinstruction is used following execution of a microsequence located by a BLK microinstruction. During execution of a Branch and Link microinstruction the current value of the RAR is saved in a scratch register. In order to link back to the microinstruction immediately following the BLK microinstruction, the return statement increments the RAR value which was saved in a scratch register and loads this value into the RAR.

The RAR Branch and Load Control receives RAR specified (XXEQRAR,XYEQRAR) and selects the A-bus data and generates XLDRAR. Since the RAR is the destination an extra clock cycle is provided by the Multiple Clock Cycle Control.

3.1.1.3 Software Instruction Decode/Interrupts. The RAR is loaded when a software instruction is decoded. There are two steps required to execute a halfword (16-bit) software instruction and three steps required to execute a fullword (32-bit) software instruction. These steps are implemented by executing CTL microinstructions using the DC field (QRD18-19).

16-Bit Software Execution

When a 16-bit software instruction is executed the two steps are IFCH (check for interrupts) and SFDC (execute the software instruction).

Step 1

A microinstruction with DC = IFCH allows the Instruction Decode ROM to check for interrupts ($\overline{X}$INTRPT) from the Interrupt Control Logic. If an interrupt is detected the Instruction Decode ROM will place a 12-bit RAR value onto the A-bus which is the Start Address of the interrupt service microsequence. The RAR Branch and Load Control causes this value to be loaded into the RAR.

$\overline{X}$INTRPT is active as one of the following conditions occurs:

1) Machine Malfunction (XMALF)
   a. Parity error on read operation
   b. Power failure
   c. Memory Protect violation
2) Program Load Interrupt (XFST)
3) Primary Power Failure ($\overline{X}$PPF)
4) Panel EXECUTE pushbutton pressed (XCATN)
5) Panel in SINGLE mode (XNGL)
6) External Interrupt (TATN)

Step 2

If no interrupts are detected the Instruction Decode ROM produces a 12-bit RAR that is the Start Address of a two-microinstruction microsequence called the Common Instruction Fetch Point. The first microinstruction increments the Location Counter by 2. The second microinstruction is a CTL with DC = PFDC. This allows Instruction Register bit one (QIR01) to be examined. The second MSB of the software Op-code (QIR01) defines the length of the software instruction. In the case of 16-bit software instructions QIR01 = 0. This causes the Instruction Decode ROM to perform an SFDC decode which generates a 12-bit Start Address of the microsequence that will perform the data manipulation specified by the software instruction.

32-Bit Software Execution

When a 32-bit software instruction is executed the three steps are IFCH (check for interrupts), PFDC (fetch second half of software instruction from memory) and SFDC (execute the software instruction).

Step 1

Same as 16-bit software execution.

Step 2

As was stated in Step 2 of 16-bit software execution, the microprogram sets DC = PFDC. QIR01 is set for 32-bit software instructions. This causes the Instruction Decode ROM to perform a PFDC decode which generates the 12-bit Start Address of the microsequence that will get the second half of the software instruction from memory. The PFDC routine also increments the Location Counter by 2 and performs any required indexing and additional reads from memory necessary to set up the second operand of the software instruction.

Step 3

At the end of the PFDC routine a CTL microinstruction has DC = SFDC. This causes the Instruction Decode ROM to generate the Start Address of the microsequence used to execute the 32-bit software instruction.

3.1.1.4 Abort Type Interrupts. The Interrupt Control Logic generates QNPT (Abort Software Execution) if the microprogram has enabled Abort type interrupts and one of the following interrupts occurs:

1) Machine Malfunction
2) Program Load Interrupt
3) External Interrupt

Abort type interrupts are only enabled when a software instruction requires a long microsequence for execution (i.e. Trigonometric software instructions). This type of interrupt does not require a microinstruction with DC = IFCH, QNPT actually forces an IFCH decode. Since QNPT interrupts occur during software execution they cause the software instruction execution to be aborted. QNPT forces the Instruction Decode ROM to place the 12-bit Start Address of the QNPT service routine on the A-bus. QNPT also causes the RAR Branch and Load Control to generate XLDRAR.

The QNPT service routine sets the Location Counter back to point at the aborted software instruction and then services the interrupt. After the interrupt service is complete, the execution of the aborted software instruction is started over.

Interrupts are discussed further in Chapter 8.

3.1.2 RAR/RDR Control During Repeat Operations. The Repeat Counter provides control to allow a single microinstruction to be executed a maximum of 31 times. To perform a repeat operation the following steps are executed by the microprogram:

1) Load the Repeat Counter with the number of repeat cycles
2) CTL microinstruction sets repeat enable (QRD17)

The Repeat Counter and Control produces ROM Data Stop Enable ($\overline{X}$ERDSTP) when the Repeat Counter is enabled by QRD17. $\overline{X}$ERDSTP goes to the Multiple Clock Cycle Control to set ROM Data Stop ($\overline{X}$RDSTP) active. $\overline{X}$ERDSTP also sets QSMTH (Single Multiple Time Hold) to disable the RAR Count Enable (XCNTRAR). The RAR and RDR are held until the repeat operation is completed.

The Repeat Counter (CTR) can also be used to repeat a microsequence up to 31 times. The CTR is first loaded with the number of repeat cycles. The last microinstruction of the microsequence to be repeated is a Branch on Condition True (BCT) with a mask of zero. This BCT microinstruction will generate XBCM2 (Special Branch) if the CTR is greater than one, and it causes the CTR to be decremented by one. QRD17 is not used in this type of repeat operation.

3.1.3 ROM Address B-Bus Mux. The ROM Address B-bus Mux has four functions. Two were previously described; 1) placing the Branch Address (QRD20-31) on the B-bus for a BIF microinstruction and 2) placing the FLR and RAR on the B-bus when the RAR is the source of data (i.e. BLK microinstruction). The other two operations are 1) when executing an RI microinstruction and 2) placing the A-bus on the B-bus.

When an RI microinstruction is decoded the ROM Address B-bus Mux places QRD24-31 (Data field) on the 8 LSBs of the B-bus. Zeros are placed on the 8 MSBs of the B-bus.

The A-bus is enabled to the B-bus by the ROM Address B-bus Mux for a special operation which allows an internal register (in AM2901) to be modified by the Byte Manipulator and loaded back into another internal register. The data path begins at General Register File 2, the A output is sent directly to the A-bus, by-passing the Arithmetic Logic Unit (ALU). The data on the A-bus passes through the ROM Address B-bus Mux to the B-bus and then down to the Byte Manipulator. The output of the Byte Manipulator is placed on the D input of the AM2901 and is sent through the ALU and loaded into General Register File 2. Due to the excessive length of the data path used, two clock cycles (400ns) are allowed to complete this type of operation.

3.2 Arithmetic Logic Unit. The ALU in the basic HMP-1116 is 16 bits, but it is normally expanded to a 32-bit ALU. The arithmetic function includes the Byte Manipulator, A-Address Mux, ALU Input Code Decoder, AM2901 LSI circuit and Flag Register. All of these sub-functions are controlled by the microprogram.

3.2.1 Byte Manipulator. The Byte Manipulator controls modification of B-bus data before it is presented to the D input of the AM2901. The Byte Manipulator is controlled by the MOD field (QRD17-19) of an RR, RI or IO microinstruction. The MOD field can specify no change to the 16-bit operand or byte exchange, byte insertion, byte extraction or hex digit extraction operations.

3.2.2 A-Address Mux. The A-Address Mux allows one of two registers (X or Y fields) specified by a microinstruction to be used as the A address input to the AM2901. Normally the Y field is used as the A address and the X field provides the B address to select internal registers in the AM2901. The AM2901 specification in Chapter 1 defines the possible operations and operand selections available (page 1-17). The selectable operands do not include operations between a register specified by the B address and the D input. The A-Address Mux exchanges the roles of the X and Y fields to allow the microprogram to perform operations between the D input and (effectively) a register specified by the B address.

The A-Address Mux determines the operands selected by examining the ALU Input Control (AIC) field of a microinstruction. The AIC field (QRD04-08) defines the operands and arithmetic or logical function to be performed by the AM2901.

3.2.3 ALU Input Code Decoder. This sub-function receives the 5-bit AIC field and provides the 6-bit field (XALUIO-5) required by the AM2901 to select operands and arithmetic or logical operations.

3.2.4 AM2901 LSI Circuit. The basic HMP-1116 has a 16-bit ALU, but it is usually expanded to a 32-bit ALU. The expansion is accomplished by adding an ARC card. The ARC card contains a 16-bit ALU which is considered to be the 16 LSBs of the 32-bit ALU. The D input to the ARC AM2901 is taken directly off the B-bus.

The AM2901 receives ALU Output Control (AOC) from the microprogram. This field corresponds to the ALU destination control outlined in Chapter 1, page 1-16.

The signal XLRXX is the output enable of the AM2901. The RAR Branch and Load Control uses XLRXX to disable the ALU output when the Instruction Decode ROM places a 12-bit Start Address on the A-bus.

3.2.5 Flag Register (FLR). The FLR provides an indication of the relative magnitude of an arithmetic result. The four bits of the FLR are designated as C (Carry), V (Overflow), G (Greater than zero) and L (Less than zero). These flags can be examined by the microprogram to make decisions (i.e. microbranch).

3.3 Register and Control Logic. The Register Control Logic decodes the X field (QRD12-15) and the Y field (QRD27-30) of a microinstruction to select the appropriate external source and destination registers. Registers can be internal (in AM2901) or external. QRD16 set indicates the Destination is External (DE) and QRD31 set indicates that the Source is External (SE). When an external register is specified as a destination, it is loaded from the A-bus. An external register specified as the source places its contents on the B-bus.

3.3.1 General Register File 1. This register file provides 16 general purpose registers for use by the software programmer. The microprogram can select the file as source or destination. The specific register is selected by R1, R2 or X2 field of the software instruction being executed (QIR08-15).

3.3.2 Program Status Register (PSR)

The PSR is indirectly controlled by the software programmer. Figure 2-1 on page 2-2 of the Manual defines the 32-bit Program Status Word (PSW). The PSR contains the 16 MSBs of the PSW. The Condition Code field (bits 12-15) can only be loaded from the FLIR or the Alarm Register.

3.3.3 Flag Register (FLR). The FLR, as was previously stated, provides an indication of the relative magnitude of the result of an arithmetic operation. When a software instruction specifies an arithmetic operation, the microprogram will execute the software instruction by executing a specific microsequence. The microinstruction which actually performs the arithmetic operation sets the flags in the FLR and the last microinstruction in the microsequence causes the FLR to be loaded into the Condition Code of the PSR. This provides the programmer with an indication of the relative magnitude of the result of executing his software instruction.

The microprogram also has the capability of loading the FLR from the four LSBs of the A-bus (FLR is destination). This gives the microprogram the capability of inspecting any bit in any register by placing it into the FLR.

3.3.4 Alarm Register. The Alarm Register is a four bit register that is loaded into the Condition Code if one of the following failures occurs:

1) Memory Violation
2) Parity Error (Operand Read)
3) Parity Error (Instruction Read)
4) Early Power Failure

These failures are called Machine Malfunctions and can cause a Machine Malfunction Interrupt (XMALF).

3.3.5 Other External Registers. The Memory Address Register (MAR), Memory Data Register (MDR) and Instruction Register (IR) are part of the Processor to Memory Interface and are described in 3.4. The Repeat Counter is defined in 3.1.2.

3.4 Processor Memory Operations. The microprogram controls processor access to memory. The sub-functions shown in Figure 3-1 that are used by the processor during memory operations are:

1) Memory Address Register
2) Memory Bank Extension Control
3) Memory Data Register
4) Instruction Register
5) Memory Interface and Timing

3.4.1 Memory Address Register (MAR). The 15 MSBs of the MAR select 1 of 32K memory locations within a bank of memory. The LSB (QMAR15) is used for byte insert/byte extract operations, and is not used to address memory. The load control (XLDMAR) allows the A-bus to be loaded into the MAR.

3.4.2 Memory Bank Extension Control Logic. The Memory Bank Extension Control Logic provides memory bank controls (TMAA, $\overline{\text{TMAB}}$) to select 1 of 4 banks of memory. The bank bits ($\overline{\text{TMAA}}$, $\overline{\text{TMAB}}$) are set according to the operation to be performed.

If an operand is to be written or read to/from memory, PSR bits 10 and 11 (QPSW10-11) are loaded into the bank control logic.

If an instruction is to be read from memory, QPSW08-09 are selected.

When using the Maintenance Panel to enter a memory address, $\overline{\text{TDMA}}$-$\overline{\text{TDMB}}$ are loaded from the I/O Mux Bus.

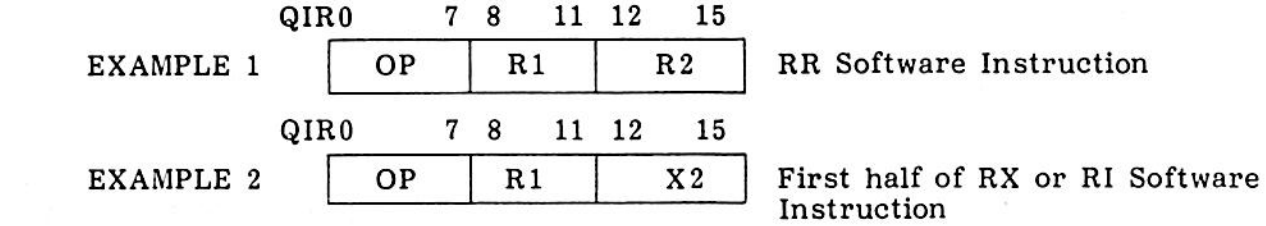
The store to bank and load from bank software instructions allow the programmer to access any bank of memory independent of the setting of QPSW10-11. During the execution of these software instructions, a micro-instruction will set the bank controls by loading them from XBA06-07.

3.4.3 Memory Data Register (MDR). The MDR is loaded with memory data during a read operation, and during a write operation the contents of the MDR are stored in memory.

The MDR load controls (XLDMDRU, XLDMDRL) allow byte insert operations (i.e. change only one byte in memory). These two load controls from the Register Control Logic allow the MSBs (XLDMDRU active) or LSBs (XLDMDRL active) to be loaded from the A-bus. If byte operations are not being performed and the MDR is to be loaded from the A-bus, both XLDMDRU and XLDMDRL will be active.

The MDR is loaded from the Memory Data Bus ($\overline{\text{TMD}}$00-15) during processor read operations by XLDMDRM from the Memory Interface and Timing.

3.4.4 Instruction Register (IR). The IR is loaded with instructions read from memory by XLDIRM. After the IR is loaded, the contents will represent either a 16-bit software instruction or the first half of a 32-bit software instruction.



The software instructions allow the programmer to specify arithmetic or logical operations by selecting a unique software Op-code. The programmer also specifies general registers to be used by the microprogram during execution of the software instruction.

The software Op-code (QIR00-07) is used by the Instruction Decode ROM to locate the appropriate microsequence in the microprogram. General Registers in File 1 are selected by QIR08-15.

A microinstruction can specify the IR as the destination register. When the Register Control Logic decodes that the IR is the destination register, it generates XLDIR which causes the IR to be loaded with data from the A-bus.

3.4.5 Memory Interface and Timing. There are three functions that can access the MOS RAM memory. These functions are (from highest to lowest priority) Refresh, DMA and Processor. The priority of memory access and basic memory timing are performed within the memory function (Figure 10-1, Chapter 10). The Memory Interface and Timing controls the processor functions associated with processor memory access. During DMA access these processor functions are disabled.

3.4.5.1 DMA Access. Unlike the processor, DMA devices do not use the MAR or MDR, but the Memory Address Bus and Memory Data Bus are shared by both functions. To prevent the MDR and MAR from affecting these busses during a DMA transfer, the Memory Interface and Timing disables Enable Memory Address Control ($\overline{\text{Q}}$EPMA inactive) and Enable Memory Data Control (QEPMD inactive). These controls are disabled when $\overline{\text{X}}$EPMAC1 (Disable Address and Data) is received from memory. DMA transfers are totally independent of microprogram control.

3.4.5.2 Processor Access. Processor access to memory is controlled by the microprogram. The Memory Control (MC) field (QRD20-22) of a CTL microinstruction defines the type of read or write operation. When a microinstruction generates a memory request, the RAR and RDR must be held until Memory Access is Granted (QMAG active). Also, since memory access time is 400ns, in some cases the RAR and RDR will be held until the memory operation is complete. The Memory Interface and Timing generates $\overline{X}$ERDSTP which is sent to the Multiple Clock Cycle Control.

3.4.5.3 Memory Parity. Each location of memory is 17 bits (16 data bits plus 1 parity bit). The Parity Bit ($\overline{T}$MD16) is generated during DMA or Processor write operations and is checked during Processor read operations.

PARITY GENERATION

Whenever DMA or Processor write into memory a Parity Bit ($\overline{T}$MD16) is generated by the Processor. The Parity Checker/Generator is part of the MDR function. The Parity of $\overline{T}$MD00-15 is checked and Even Parity (XEVNPTY) is set if the parity of the 16 bits is even. The Memory Interface and Timing uses XEVNPTY to generate $\overline{T}$MD16.

PARITY CHECKING

Whenever the processor reads from memory it checks parity. The Memory Interface and Timing places $\overline{T}$MD16 on XPTY16 which is sent to the MDR function. XPTY16 along with $\overline{T}$MD00-15 is checked at the MDR. If the parity is even (parity error) XEVNPTY is active. The Memory Interface and Timing uses XEVNPTY to set QEVNPTY which is sent to the Alarm Register. Since a read operation is being performed and even parity has been detected, the appropriate bit in the Alarm Register will be set.

3.5 I/O Mux Bus Interface. The I/O Interface consists of the Input Flag Control, Output Flag Control, I/O Mux Bus and Flag Timing Control and Transceivers (XCVRS). The I/O interface is controlled by IO microinstructions. All IO microinstructions contain an Input Flag field (QRD27-31), Output Flag field (QRD22-26) and a field to select an internal or external register. The Output Flag field selects $\overline{T}$CMD, $\overline{T}$SR, $\overline{T}$DR, $\overline{T}$DA, $\overline{T}$ADRS or $\overline{T}$ACK. The Input Flag field always selects $\overline{T}$SYN.

There are two types of IO microinstructions, Input and Output. Input microinstructions cause a register to be loaded with information from an I/O device. Output microinstructions cause the contents of a register to be sent to an I/O device.

OUTPUT MICROINSTRUCTION OPERATIONS

1) Send a device address from a register to an I/O device on the 8 LSBs of the I/O Data Bus (set $\overline{\text{T}}$ADRS and wait for $\overline{\text{T}}$SYN)

2) Send a device command from a register to an I/O device on the 8 LSBs of the I/O Data Bus (set $\overline{\text{T}}$CMD and wait for $\overline{\text{T}}$SYN)

3) Send data from a register to an I/O device on the I/O Data Bus (set $\overline{\text{T}}$DA and wait for $\overline{\text{T}}$SYN)

INPUT MICROINSTRUCTION OPERATIONS

1) Receive status byte on the 8 LSBs of the I/O Data Bus and load it into a register (set $\overline{\text{T}}$SR and wait for $\overline{\text{T}}$SYN)

2) Receive data on the I/O Data Bus and load it into a register (set $\overline{\text{T}}$DR and wait for $\overline{\text{T}}$SYN)

3) Receive interrupting device's address on the 8 LSBs of the I/O Data Bus and load it into a register (set $\overline{\text{T}}$ACK and wait for $\overline{\text{T}}$SYN)

For further information on the I/O Mux Bus, see Chapter 7.

4.0 MICROINSTRUCTION FORMATS

4.1 Introduction. The HMP-1116 is a microprogrammed minicomputer. The microprogram controls the microprogram address, arithmetic functions, memory operations, input/output operations, byte and hex digit modifications and software instruction decode.

The microprogram consists of 36-bit microinstructions which reside in the Microprogram Roms. Microinstructions conform to one of the five formats shown in Figure 4-1. The format of the microinstruction is determined by the op-code. Each of the fields within a microinstruction is defined in Tables 4-1 thru 4-17.

4.2 Microinstruction Formats Description

RR FORMAT

An RR microinstruction performs an arithmetic or logical operation on two registers, and places the result in the destination register. The registers are specified by the X and Y fields. The DE field determines whether the destination is internal or external. The SE field determines whether the source is internal or external. DE and SE together determine whether X or Y is the destination register.

The operation performed by an RR microinstruction is determined by the OP (controls carry in), AIC and AOC fields. The operation is performed by the ALU function. The DIS and SPL fields determine how the 16-bit ALUs on the ARB and ARC cards are used. One or both of the ALUs can be used. If the external data input to the ARB ALU is being used, the MOD field controls how the Byte Manipulator modifies the 16-bit operand on the B-bus.

The CSV and FLG fields control how the flags in the FLR will be set as a result of the operation. The CSV controls the setting of the C-flag, and the FLG controls the setting of the V-, G- and L-flags.

The SPM field can control either the setting of the memory bank address or special functions on the COR card.

RI FORMAT

An RI microinstruction performs an arithmetic or logical operation with a register and the 8-bit DATA field. The DE field determines if the destination register is internal or external.

The OP, AIC, AOC, DIS, MOD, SPM, SPL and FLG fields have the same functions as defined in the RR format. The C-flag remains unchanged as a result of an RI microinstruction.

CTL FORMAT

A CTL microinstruction performs an arithmetic or logical operation on two registers. The X, Y, DE, SE, OP, AIC, AOC, SPM and SPL fields have the same functions as defined in the RR format.

The RPT field when set enables the next microinstruction to be repeated until the Repeat Counter is decremented to zero.

The DC field controls the Instruction Decode Rom.

The MC field generates requests to Memory.

The STATUS field controls loading the Condition Code, and clearing the Alarm Register or FLR. It also can be used for enabling or disabling the QNPT interrupt function.

The LFG field when set causes the FLR to be loaded from the MSD of the A-bus.

IO FORMAT

The X and DE fields define the source for an output operation, or the destination for an input operation.

The OP, AIC and AOC fields control passing data through the ALU function unchanged. The MOD field can be used to modify incoming or outgoing data.

The BC field is not used.

The OF field selects which output flag is generated, defining what specific type of I/O operation is being performed.

The IF field is always '01' which enables either a Sync from a device or a False Sync Timeout to terminate the IO microinstruction.

The SPM and SPL fields are normally all zeroes for an IO microinstruction.

BR FORMAT

The BLK microinstruction will unconditionally branch to the address specified by the A field. The AIC and AOC fields allow the return address to pass unchanged to an internal register specified by the X field. The MSK field is not applicable to the BLK microinstruction.

The BCT and BCF microinstructions conditionally branch to the address specified by the A field. A BCT will branch if the condition specified by the MSK field is met. A BCF will branch if the condition specified by the MSK field is not met. The AIC, AOC and X fields are not applicable to BCT and BCF microinstructions.

The BIF microinstruction will branch if the condition specified by the MSK field is not met. If the branch is to be taken, the AIC and AOC fields will cause the A field to be added to the internal register specified by the X field. This indexed branch address is then loaded into the RAR.

The SPM and SPL fields are normally all zeroes for BR microinstructions.

4.3 Analyzing Microinstructions

EXAMPLE #1 (RR)

1000 OP	00111 AIC	011 AOC	0010 X	1 DE	000 MOD	0 DIS	000 FLG	000 CSV	1010 Y	1 SE	00 SPM	00 SPL

(In Hex: 83B280150)

When this microinstruction is in the Rom Data Register, the op-code of '8' defines the format to be RR. The fields within the microinstruction are decoded as follows:

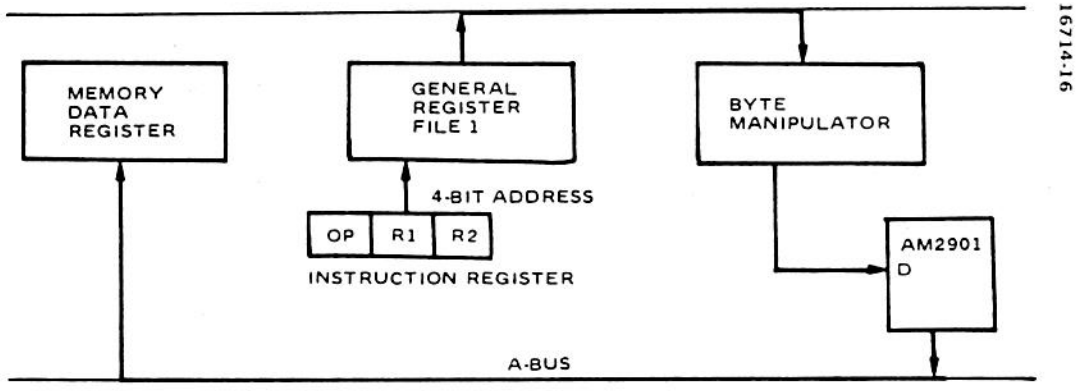
AIC(ALU Input Code) = '07' See Table 4-2		This field selects the D input of the AM2901. Note that Carry in = 0 (Table 4-1.), so the operation is not D + 1
AOC(ALU Output Control) = '3' See Table 4-3		Load RAM with ALU Output, Place ALU Output on A-bus. (If Dest. is internal, then the ALU is clocked. If Dest. is external, then data on A-bus is used)
DE(Dest. External) = 1	See Table 4-4	DE = 1, indicates destination is external
SE(Source External) = 1		SE = 1, indicates source is external
		ALU Clock Disabled
X = '2'	See Table 4-10	MDR is the Destination
Y = 'A'		The source is a Register in File 1, specified by the R1 field of a software instruction
MOD = '0' See Table 4-11		Direct input (D) to Am2901 is not Modified
DIS = 0 See Table 4-8		Destination Register is loaded
FLG = '0' See Table 4-6		No change to V-, G- or L-flags in FLR
CSV = '0' See Table 4-7		No change to C-flag in FLR

SPM = '0' See Table 4-9	No effect
SPL = '0' See Table 4-8	ARB ALU is enabled (If Dest. is internal, then ARB ALU clocks are enabled. If Dest. is external, then ARB ALU output data is placed on the A-bus.)

To determine the overall function of this microinstruction see Table 4-4. From the column titled "Implied Function and Source Restrictions" note the entry E(X)←ⓕ E(Y) which is applicable in this case. The notation " ⓕ " indicates a function may be performed. This function is dictated by the AIC. The entry is read "External register specified by the X field is loaded with a function of external register specified by the Y field".

In this example the contents of the Memory Data Register is loaded from a General Register in File 1. The specific register is specified by the R1 field of a software instruction. (MDR)←(GENERAL REGISTER).

The figure shown below indicates data flow as a result of executing this microinstruction.



EXAMPLE #2 (RI)

0110 OP	00111 AIC	011 AOC	0101 X	0 DE	001 MOD	0 DIS	000 FLG	1000 0000 DATA	00 SPM	00 SPL

(In Hex: 63B510800)

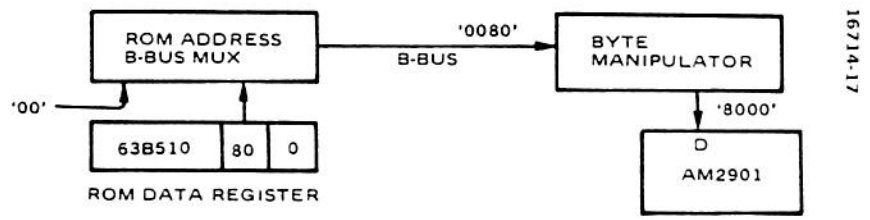
The op-code of '6' in this microinstruction defines the format as RI. The fields are decoded as follows:

AIC = '07'	See Table 4-2	Selects D input of AM2901 as operand.
AOC = '3'	See Table 4-3	Load RAM with ALU Output.
DE = 0	See Table 4-4	Destination is internal to AM2901.
X = '5'	See Table 4-10	Scratch register '5' is destination.
MOD = '1'	See Table 4-11	Exchange Bytes on direct input to AM2901.
DIS = 0	See Table 4-8	Destination register is loaded.
FLG = '0'	See Table 4-6	No change to V-, G- or L-flags in FLR.
DATA = '80'		Immediate data field.
SPM = '0'	See Table 4-9	No effect
SPL = '0'	See Table 4-8	ARB ALU is enabled

Referring to Figure 4-4 we see that the function performed is I(X)←I(X) ⓕ Im. This is read "Internal register specified by the X field is loaded with a function of the internal register specified by the X field and immediate data (DATA)."

The immediate field is '80'. A 16-bit word is applied to the input of the Byte Manipulator (see figure below). The '80' is the two LSDs and the two MSDs are forced to zero. Therefore, the input to the Byte Manipulator is '0080'. The MOD field dictates a byte exchange which results in '8000' being applied to the direct input of the AM2901.

The final result of executing this microinstruction is to load 8000 into internal register '5'.



EXAMPLE #3 (CTL)

1100 OP	00011 AIC	011 AOC	0001 X	1 DE	0 RPT	11 DC	001 MC	001 STATUS	0 LFG	1010 Y	0 SE	00 SPM	00 SPL

(In Hex: C1B1B2540)

The op-code 'C' defines the microinstruction as CTL format with a carry in of zero. The fields are decoded as follows:

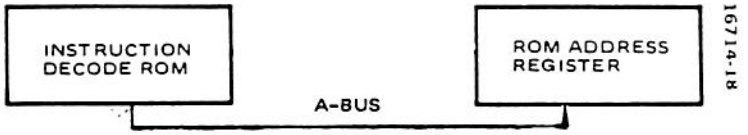
AIC = '03'	See Table 4-2	Selects B as the operand. This means an internal register specified by the B address on the AM2901
AOC = '3'	See Table 4-3	Place ALU Output on the A-bus
DE = 1	See Table 4-4	DE = 1 indicates that the destination is external
SE = 0		SE = 0 indicates that the source is internal Notice from Table 4-4 that the Y field specifies the destination and X specifies the source*
X = '1'	See Table 4-10	*Source is internal register 1, which is a data buffer
Y = 'A'		*Destination is General Register File 1, the specific register in File 1 is defined by the R1 field of a software instruction
RPT = 0		No repeat operation is to be performed
DC = '3'	See Table 4-13	Decode control is instruction fetch
MC = '1'	See Table 4-14	Read first half of software instruction
STATUS = '1'	See Table 4-12	The FLR is loaded into the Condition Code of PSW, and zeroes are loaded into the FLR
LFG = 0		
SPM = '0'	See Table 4-9	No effect
SPL = '0'	See Table 4-8	ARB ALU is enabled

*Normally the X field specifies the destination in RR and CTL microinstructions. The only exception is when DE = 1 and SE = 0, in that case the Y field specifies the destination.

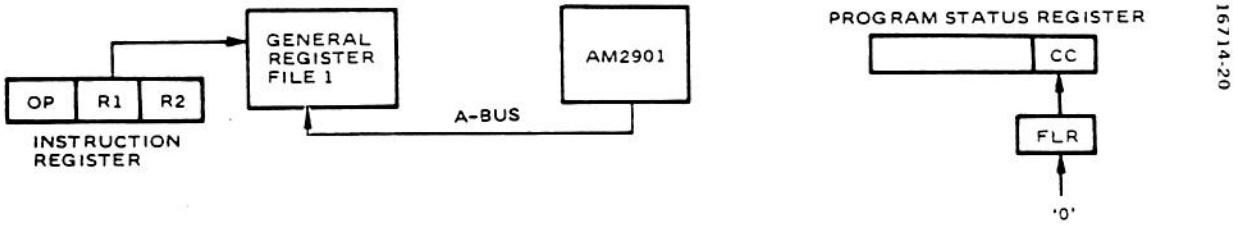
This CTL microinstruction actually specifies four separate operations. The AIC, AOC, X, Y, DE and SE fields specify an operation to be performed with two registers. The DC field specifies that a start address from the Instruction Decode Rom is to be loaded into the RAR. The MC field specifies that a halfword is to be read from memory. The STATUS field specifies that the FLR is to be loaded into the Condition Code and that the FLR is to be zeroed.

It takes 200ns to execute the microinstruction, it takes 400ns to access memory. The RAR is clocked halfway through the 200ns clock cycle, the destination register, FLR and PSR are clocked at the end of the 200ns clock cycle and the memory access is completed after an additional 200ns (during the next microinstruction). The next microinstruction has nothing to do with the loading of the MDR and IR, that is controlled by the hardware. The sequence of operations are as follows.

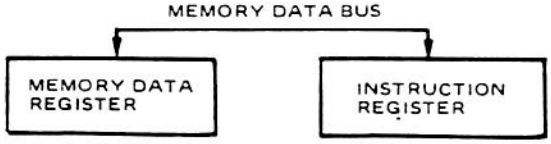
First, halfway through the 200ns clock cycle the RAR is loaded. If an interrupt is pending, the new RAR will point to the interrupt service routine. If no interrupt is pending, the RAR will point to a routine to begin decoding and executing the software instruction in the Instruction Register.



Second, at the end of the 200ns clock cycle, internal register '1' is loaded into General Register File 1. From Table 4-4, the entry E(Y) ← ⓕ I(X) defines the implied function. Also, the Condition Code is loaded from the FLR and '0' is loaded into the FLR.



Finally, after another 200ns the MDR and IR are loaded from memory.



EXAMPLE #4 (IO)

0101 OP	00011 AIC	011 AOC	1000 X	0 DE	000 MOD	00 BC	00001 OF	00001 IF	00 SPM	00 SPL

(In Hex: 51B800210)

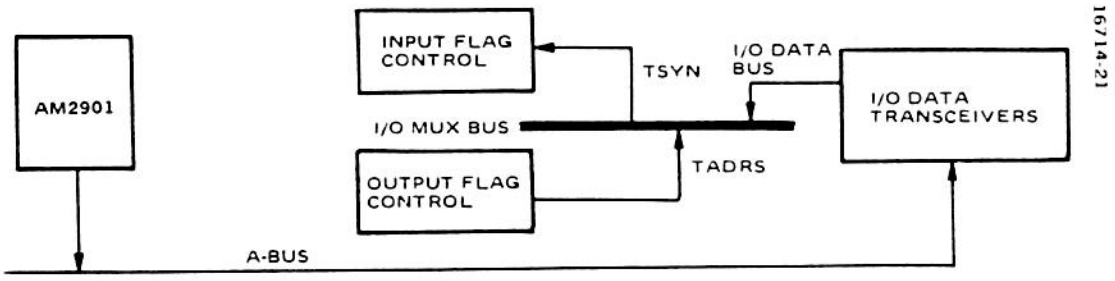
The op-code of '5' in this microinstruction defines the format as IO. An op-code of '5' is an output operation, the fields are defined as follows:

AIC = '03'	See Table 4-2	Selects internal register designated as B
AOC = '3'	See Table 4-3	Place ALU Output on A-bus
DE = 0	See Table 4-4	Source internal
X = '8'	See Table 4-10	Internal register '8' is source
MOD = '0'	See Table 4-11	No effect
BC = '0'		Not used
IF = '01'		Enable Sync to terminate microinstruction
OF = '01'	See Table 4-15	Set Address flag
SPM = '0'	See Table 4-9	No effect
SPL = '0'	See Table 4-8	ARB ALU is enabled

In Table 4-4 there are two statements for this IO microinstruction, I(X)←I/O and I/O←I(X). I(X)←I/O is for an input microinstruction. I/O←I(X) is for an output microinstruction. The second statement applies in this example since this is an output microinstruction. The term I/O means data on the I/O Mux Bus data lines.

In an IO microinstruction the data on the data bus is identified by the output flag set. In this case the OF field causes the Address flag to be set, indicating that the data on the 8 LSBs of the I/O data bus is a device address.

The final result of executing this microinstruction is to send the contents of scratch register '8' out on the I/O Mux Bus, set the Address flag and wait for the Sync flag. This is shown in the diagram below:



EXAMPLE #5 (BR)

0001	01011	011	0000	0010	0111 1111 1110	00	00
OP	AIC	AOC	X	MSK	A	SPM	SPL

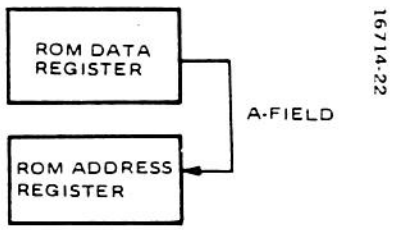
(In Hex: 15B027FE0)

The op-code of '1' in this microinstruction defines the format as BR. An op-code of '1' is a BCT(Branch on Condition True) microinstruction. The fields are decoded as follows:

AIC = '0B'		No effect
AOC = '3'		No effect
X = '0'		No effect
MSK = '2'	See Table 4-16	True branch condition is when G-flag is set
A = '7FE'		If branch condition is met, branch to '7FE'
SPM = '0'		No effect
SPL = '0'		No effect

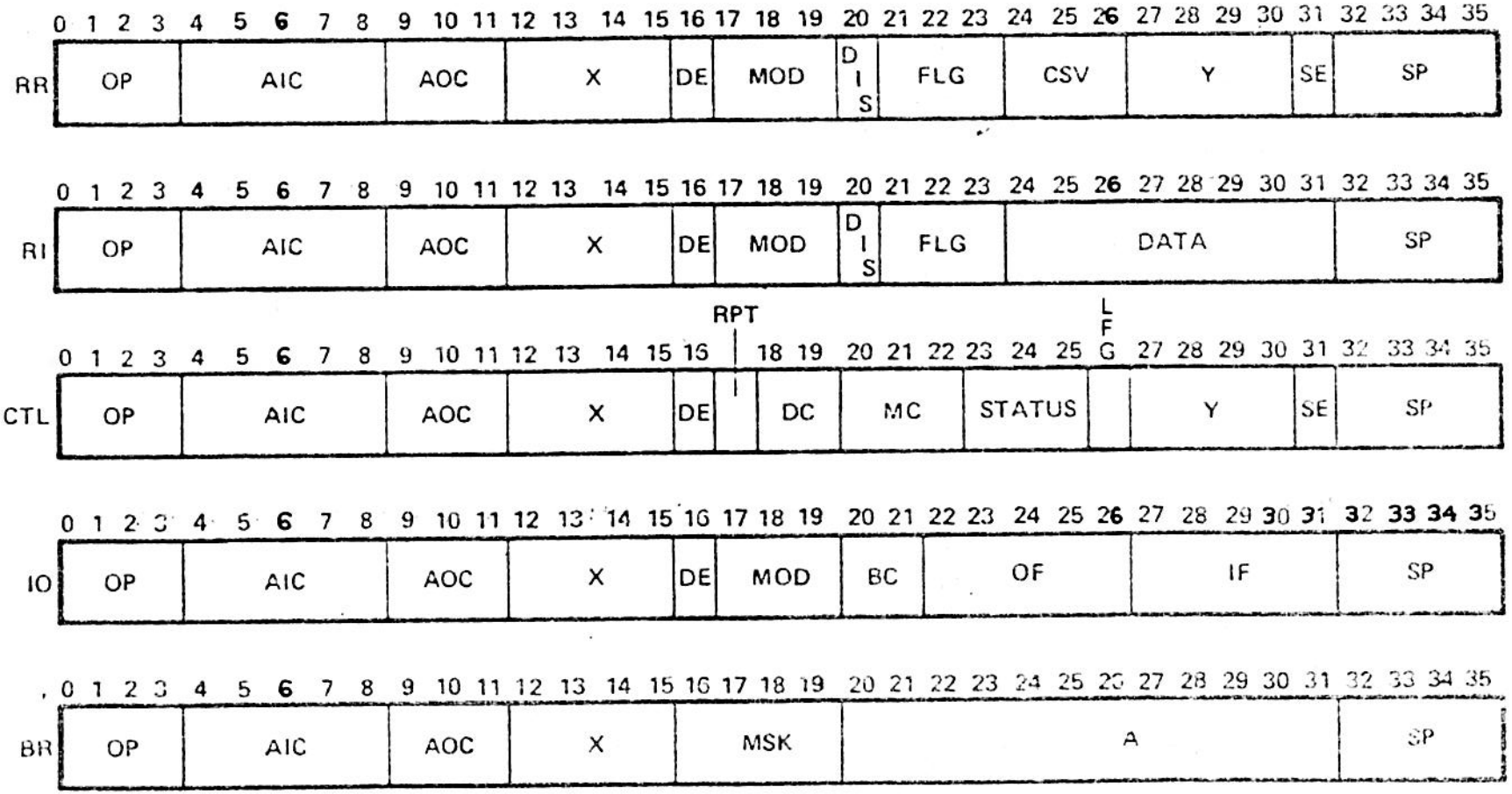
In Table 4-4 a BCT operation is described as RAR←A. This operation will only take place if the branch condition is satisfied. In this example, the RAR will be loaded only if the G-flag is set. If the G-flag is not set, the RAR will be incremented to point at the next consecutive microinstruction.

If the G-flag is set, the RAR is loaded as shown below:



MICROCODE FORMATS

HUGHES



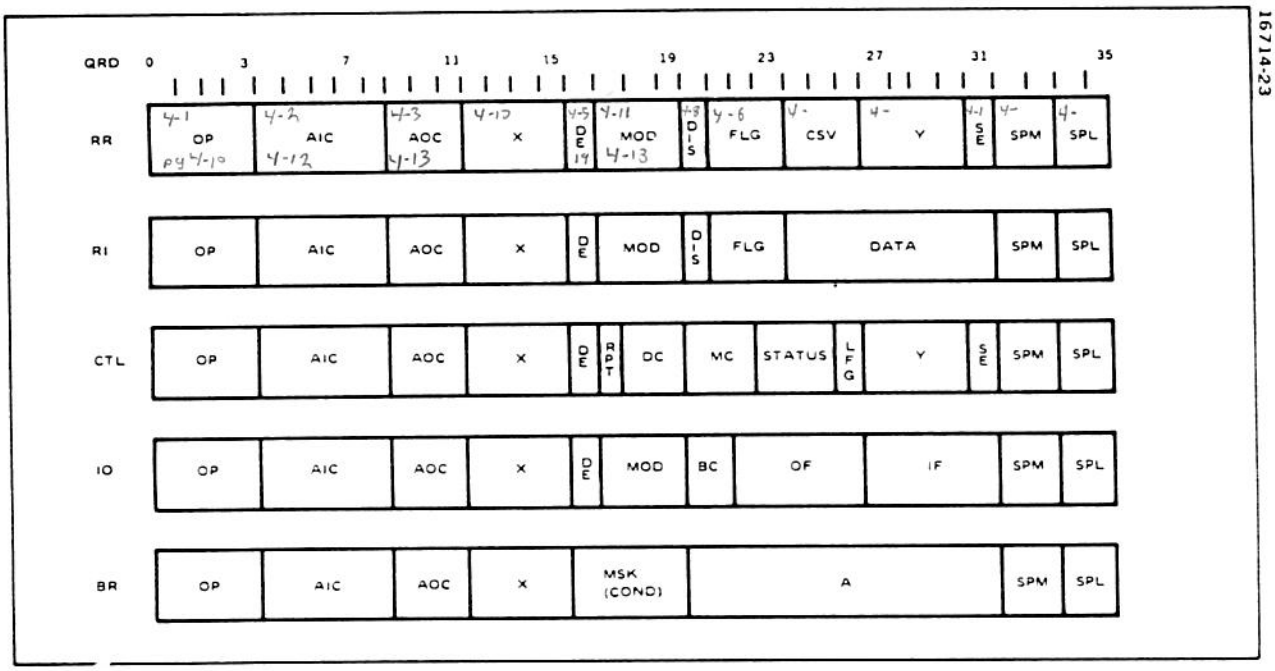


Figure 4-1. Microinstruction Formats

Table 4-1. OP-Code Formats

OP	FORMAT	DESCRIPTION	INPUT CARRY
0000	BR	Branch and link	0
0001		Branch on condition true	
0010		Branch on condition false	
0011		Branch indexed on false	
0100	IO	Input	0
0101		Output	
0110	RI	Immediate-to-register	0
0111			1
1000	RR	Register-to-register	0
1001			1
1010			CSV
1011			-CSV
1100	CTL	Register-to-register control	0
1101			1
1110	DIV	Divide step	-CSV
1111	MUL	Multiply step	-CSV

CSV = C-flag in FLR

MICROINSTRUCTION FIELD CROSS REFERENCE INDEX

FIELD	TABLE	FUNCTIONAL SCHEMATIC PAGE NUMBER
AIC	4-2	16
AOC	4-3	15
X or Y (Internal)	4-10	15, 16
X or Y (External)	4-10	8, 9, 10
DE and SE	4-4	8, 9, 10
MOD (Binary Position)	4-5	15
MOD (Byte Manipulator)	4-11	14
DIS	4-8	8
FLG	4-6	18
CSV	4-7	17
SPM	4-9	20, 32
SPL	4-8	15, 16, 17
DATA	---	5
RPT	---	13
DC	4-13	4
MC	4-14	30, 34
STATUS, LFG	4-12	17, 18, 24, 25
OF	4-15	22
IF	---	21
MSK	4-16	3
A	---	5

Table 4-2. AIC Field Definition

AIC	FUNCTION if C=0	FUNCTION if C=1
00000	A + Q	A + Q + 1
00001	A + B	A + B + 1
00010	Q	Q + 1
00011	B	B + 1
00100	A	A + 1
00101	D + B	D + B + 1
00110	D + Q	D + Q + 1
00111	D	D + 1
01001	B - A - 1	B - A
01010	Q - 1	Q
01011	B - 1	B
01100	A - 1	A
01101	B - D - 1	B - D
01111	$\overline{D}$	- D
10001	A - B - 1	A - B
10010	$\overline{Q}$	- Q
10011	$\overline{B}$	- B
10100	$\overline{A}$	- A
10101	D - B - 1	D - B
10110	D - Q - 1	D - Q
10111	D - 1	D
11000	A $\wedge$ B	A $\wedge$ B
11001	A $\vee$ B	A $\vee$ B
11010	A $\oplus$ B	A $\oplus$ B
11100	D $\wedge$ B	D $\wedge$ B
11101	D $\vee$ B	D $\vee$ B
11110	D $\oplus$ B	D $\oplus$ B

Key to Symbols:

\+ Addition
\- Subtraction
$\wedge$ Logical And
$\vee$ Logical Or
$\oplus$ Exclusive Or

$\overline{X}$=One's Complement of X

-X=Two's Complement of X

Table 4-3. AOC Field Definition

AOC	Load Control*	Shift Control	Output Data Control
000	Load Q	No shift	ALU output
001	(Not used)	(Not used)	(Not used)
010	Load RAM	No shift	RAM location A
011	Load RAM	No shift	ALU output
100	Load RAM and Q	Right shift	ALU output
101	Load RAM	Right shift	ALU output
110	Load RAM and Q	Left shift	ALU output
111	Load RAM	Left shift	ALU output

*Load will occur only if ALU is clocked.
The ALU will be clocked when the destination register is specified as internal.

Table 4-4. AIC and AOC Terminology Definition

Format	Source Control DE	Source Control SE	ALU Address A	ALU Address B	Implied Function and Source Restrictions	ALU Input Control AOC Restrictions
RR or CTL	0	0	Y	X	I(X)←I(X) ⓕ I(Y) or I(X)← ⓕ I(Y)	---
	0	1	*		I(X)←I(X) ⓕ E(Y) or I(X)← ⓕ E(Y)	
	1	0			E(Y)←E(Y) ⓕ I(X) or E(Y)← ⓕ I(X)	
	1	1		*	E(X)← ⓕ E(Y)	Unary operations only
RI	0	-	*	X	I(X)←I(X) ⓕ Im	---
	1			*	E(X)← ⓕ Im	Unary operations only
BLK	-	-	*	X	I(X)← 'RAR'; 'RAR'← A	---
BIF	-	-	*	X	'RAR'← I(X)+A	
IO	0	-	*	X	I(X)← I/O; I/O←I(X)	Unary operations only
	1			*	E(X)← I/O; I/O←E(X)	
BCT and BCF	-	-	*	*	RAR← A	---

I(X) = Register internal to ALU; E(X) = Register external to ALU;
- = Not specified;
* = ALU Address not applicable to operation;
Im = B-bus input modified by MOD-field, if any;
Unary operations = Invert, twos complement, transfer, increment, or decrement

Table 4-5. Binary Positioning Control

MOD	QRD-10	ALU MSB Input	Q LSB Input	Function
00X	0	0 (C)	-	Right logical shift; set ALU MSB to zero (when OP = 14 or 15, set ALU MSB to carry out)
	1	---	0	Left logical shift; set Q LSB to zero
01X	0	Sign Bit	---	Multiply; set ALU MSB to true sign (Sign ⊕ Overflow)
	1	---	ALU MSB Carry	Divide; set Q LSB to quotient bit (ALU carry)
10X	0	C	---	Right algebraic shift; set ALU MSB to CSV
	1	---	C	Left algebraic shift, set Q LSB to CSV
11X	0	Q LSB	---	Right rotate; set ALU MSB to Q LSB
	1	- -	ALU MSB	Left rotate; set Q LSB to ALU MSB

Table 4-6. FLG Field Definition

FLG	Status Saved in V Flag of FLR	Status Saved in G and L Flags of FLR
000	No Change	No Change
001	No Change	Single precision algebraic result
010	No Change	Multiprecision algebraic result
011	Algebraic overflow	Multiprecision algebraic result
100	Algebraic overflow	Single precision algebraic result
101	*V Flag or carry save exclusive ORed with left-shifted out bit of ALU	Single precision algebraic result
110	Algebraic overflow	No Change
111	*V Flag or carry save exclusive ORed with left-shifted out bit of ALU MSB	No Change

*Only for MOD code of algebraic shift or rotate, otherwise set to zero.

Table 4-7. CSV Field Definition

CSV	Status Saved in C Flag of FLR
XX0	No change
001	Carry from ALU MSB
011	Borrow from ALU MSB
101	Left-shifted out bit from ALU MSB
111	Right-shifted out bit from Q LSB

Table 4-8. ARB and ARC Enable Control

DIS	SPL	FUNCTION
0	00	Single Precision, Disable ARC Dest. Clock
0	01	Transfer from ARC to ARB, Disable ARC Dest. Clock
0	11	Double Precision
1*	01	Double Precision, Disable ARB and ARC Dest. Clocks
1*	10	Transfer from ARB to ARC, Disable ARB Dest. Clock.
1*	11	Single Precision, Disable ARB Dest. Clock

*Also Disables Dest. Clocks to External Registers

Table 4-9. SPM Field Definition

SPM	XMARD=1,LOAD QMARA-B WITH	XMARD=0,COR CARD OPERATION	**
00	QMARA-B*	No Operation	
01	Zeroes	Increment Iteration Counter	
10	PSW10-11	Load Shift Counter, Shift Algebraic	
11	PSW08-09	Load Shift Counter, Shift Logical	

*If Input Microinstruction, Load With TDMA-B.
**XMARD=1 when the MAR is the Destination.

Table 4-10. Register Definitions

X or Y	INTERNAL (ARB)	EXTERNAL
0000	Abort Return Address (Not Used)	Not Used
0001	TEMPYD	Repeat Counter (CTR)
0010	Constant '00FF'	Memory Data Reg. (MDR)
0011	Constant '0004'	Instruction Reg. (IR)
0100	Constant '0002'	Memory Address Reg. (MAR)
0101	Constant '8000'	Program Status Reg. (PSW)
0110	Temporary Save for LOC	EXT Reg.*
0111	Location Counter (LOC)	Cordic Fixed Memory (Source), Iteration Counter (Dest.)
1000	Scratch Reg.	Reg. File 1, R2+1 (YSP1)
1001	Scratch Reg.	Reg. File 1, R2 (YS)
1010	Scratch Reg.	Reg. File 1, R1 (YD)
1011	Scratch Reg.	Reg. File 1, R1+1 (YDP1)
1100	Scratch Reg.	Reg. C (Source), Reg. A (Dest.)
1101	Scratch Reg.	Reg. D (Source), Reg. B (Dest.)
1110	Scratch Reg.	Flag Register (FLR)
1111	Scratch Reg.	Rom Address Reg. (RAR)

*EXT Reg. is Defined in Table 4-17.

Table 4-11. MOD Field Definition (Byte Manipulator Control)

D_0	D_1	D_2	D_3	B-bus input

MOD	QMAR15	Resulting Word				Function
000	X	D_0	D_1	D_2	D_3	Parallel load
001	X	D_2	D_3	D_0	D_1	Byte exchange
010	0	D_2	D_3	P	P	Byte insert
	1	P	P	D_2	D_3	
011	0	0	0	D_0	D_1	Byte extract
	1	0	0	D_2	D_3	
100	X	0	0	0	D_0	Hex digit 0 extract
101	X	0	0	0	D_1	Hex digit 1 extract
110	X	0	0	0	D_2	Hex digit 2 extract
111	X	0	0	0	D_3	Hex digit 3 extract

X = Don't care; P = Previous contents; 0 = Zero

Table 4-12. Status Field Definition

STATUS	LFG	FUNCTION
000	0	No Operation
000	1	Load FLR from A-bus MSD
001	0	FLR → CC, Zero FLR
011	0	No Change to FLR, FLR → CC
100	0	Alarm Reg → CC, Zero Alarm Reg
101	0	Zero FLR
110	0	Reset Interrupt Enable
110	1	Load FLR from A-bus MSD, Reset Interrupt Enable
111	0	Set Interrupt Enable

Table 4-13. DC Field Definition

Φ_1 Φ_{1B} $\Phi_{1'}$	DC	FUNCTION
X 0 0	00	No Operation
1 0 1	01	Primary Function Decode (PFDC) (Second Fetch)
X 1 0	10	Secondary Function Decode (SFDC)
X 1 1	11	Instruction Fetch (IFCH) (Interrupt Check)

Table 4-14. MC Field Definition

MC	FUNCTION
001	Read First Half of Instruction
010	Read Second Half of Instruction
011	Read Operand
100	Not Used (Read)
101	Write
110	Write Priveleged
111	Not Used (Write)
000	No Operation

Table 4-15. OF Field Definitions

0F	I/O Operation
00001	Address
00010	Command
00011	Data Available
00100	Data Request
00101	Acknowledge Interrupt
00111	Status Request

Table 4-16. MSK Field Definition

MSK	TRUE BRANCH CONDITION	FALSE BRANCH CONDITION
0000	Special Branch Cond. (Repeat Counter Status)	Unconditional
0001	L	-L
0010	G	-G
0011	G or L	-G and -L
0100	V	-V
0101	V or L	-V and -L
0110	V or G	-V and -G
0111	V or G or L	-V and -G and -L
1000	C	-C
1001	C or L	-C and -L
1010	C or G	-C and -G
1011	C or G or L	-C and -G and -L
1100	C or V	-C and -V
1101	C or V or L	-C and -V and -L
1110	C or V or G	-C and -V and -G
1111	C or V or G or L	-C and -V and -G and -L

C = Carry Flag; V = Overflow Flag; G = Greater than Zero Flag;
L = Less than Zero Flag

Table 4-17. EXT Register

Bit	Source Definition	Dest. Definition
00	Execute Pushbutton(XCATN)	Not used
01	Single Mode(XSNGL)	Not used
02	Machine Malfunction(XMALF)	Not used
03	Power Failure(XPPF)	Not used
04	Remote Program Load(XFST)	Not used
05	I/O Inerrupt(TATN)	Bank Address Force Enable (XBA05)*
06	Not used	Bank Address MSB Force Bit(XBA06)
07	Not used	Bank Address LSB Force Bit(XBA07)
08	Panel Lock(SLOCK)	Not used
09	Memory Test Pushbutton(SMTST)	Not used
10	Program Load Pushbutton(SPRLD)	Not used
11	Memory Operational(QMOP)	Not used
12	Bank Address MSB(QMARA)	Reset Remote Program Load (XBA12)
13	Bank Address LSB(QMARB)	Set Memory Operational Indicator(XBA13)
14	Not used	Set Wait Indicator(XBA14)
15	Halfword Flag(THW)	Reset Wait Indicator(XBA15)

*The SPM field must be set to zero to enable XBA06-07 to be forced into the Memory Bank Address.

EXTERNAL REGISTER LOCATION IN FUNCTIONAL SCHEMATICS

REGISTER	FUNCTIONAL SCHEMATIC PAGE NUMBER
Repeat Counter	13
Memory Data Reg.	11
Instruction Reg.	11
Memory Address Reg.	12
Program Status Reg.	12
EXT Reg. (Source)	24
EXT Reg. (Destination)	32, 33
Cordic Fixed Memory Iteration Counter	20
Reg. File 1	12
Reg. C Reg. A	20
Reg. D Reg. B	20
Flag Register	17, 18, 19
Rom Address Reg.	2

5.0 READING THE MICROPROGRAM LISTING

5.1 Introduction. The HMP-1116 processor is based on a standard processor design which has many applications. The features which determine how this general purpose processor will be used are its microprogram and its hardware options. The external register definitions and I/O flag definitions are examples of hardware options.

A standard format is used for writing the microprogram. First, the microprogrammer gives names to operands (registers and immediate values) and defines the fields of the microinstruction which control hardware options. Second, the microprogrammer writes the microprogram in assembly language. One statement in assembly language will represent the operation to be performed by one microinstruction. The formats of the assembly statements are given in Appendix B of the Tech. Manual.

A computer is fed the definitions and assembly statements and it converts the assembly statements into 36-bit microinstructions. The computer then prints out the Microprogam Listing.

The Microprogram Listing provides a complete description of the microprogram of the HMP-1116. The Listing is divided into three major sections:

1) Definitions-Define hardware options unique to this microprogram.

2) Microprogram-The assembly statements and their associated 36-bit microinstructions.

3) Cross Reference Index-A list of all defined operands and microprogram branch addresses, where in the Listing they are defined and every assembly statement which references them.

5.2 Reading Assembly Statements. In the Microprogram Listing there is an assembly statement and a 36-bit code (in hexadecimal) for every microinstruction in the HMP-1116. There are two ways of determining the operation performed by a microinstruction. In Chapter 4 we learned the first method; write out the 36 bits of the microinstruction in binary and break them into fields. The second method is to read the assembly statement in the Listing. Reading an assembly statement is much faster and easier than decoding 36 ones and zeros. The objective of Chapter 5 is to demonstrate how to read the assembly statements in the Microprogram Listing.

A typical portion of the Microprogram Listing is shown below.

```
 LOC  OBJECT CODE      STMT  SOURCE STATEMENT

00012F C1B8B2540      3542        LR    YD,TEMP1,ST=JMC,DC=IFCH,MC=MRI1
                      3544 *
                      3545 *
                      3546 ** STORE HALFWORD                              (RXAD)
                      3547 *
000130 83B280150      3548 STH     LR    MDR,YD
000131 C2300A000      3550         CMD   MC=MW
000132 9237000CC      3552 P3EXITX IR    LOC,LOCX,SP=PINS
000133 C23033400      3554         CMD   ST=CLR,DC=IFCH,MC=MRI1
                      3556 *
                      3557 *
                      3558 **  ADD IMMEDIATE SHORT                         (SF)
                      3559 *
000134 82B174270      3560 AIS     AR    TEMPYD,IR,CVF,MOD=EXT3
000135 C1B1B2540      3562         LR    YD,TEMPYD,ST=JMC,DC=IFCH,MC=MRI1
                      3564 *
```

Each line of this part of the Listing is divided into four columns, 1) LOC, 2) OBJECT CODE, 3) STMT and 4) SOURCE STATEMENT. Each line with an entry in the columns labeled LOC and OBJECT CODE represents one microinstruction. Lines with no entries in LOC and OBJECT CODE are spacing or comments. The purpose of the spacing and comments is to make the Listing easier to read.

The column labeled LOC contains the hexadecimal address of the microinstruction.

The column labeled OBJECT CODE contains the 36-bit microinstruction in hexadecimal (9 hex characters).

The column labeled STMT contains statement numbers. A statement number is assigned to every line in the Listing. The Cross Reference Index refers to lines in the Listing by using these numbers.

The SOURCE STATEMENT on a line which has a LOC and an OBJECT CODE is an assembly statement. The assembly statement was written by the microprogrammer to represent the operation to be performed by a microinstruction. A computer converted the assembly statement into a 36-bit microinstruction (the OBJECT CODE) and assigned it a location in Microprogram Memory (the LOC).

Appendix B provides a description of the assembly statements. Three steps are required to read an assembly statement:

1) Find the format of the assembly statement in Table B-1.
2) Find the function of the assembly statement in Table B-2.

3) Look up the definitions of the operands and modifiers which describe the details of the operation to be performed. Some of the definitions are found in Table B-3. Any operand or modifier not defined exactly in Table B-3 will be defined in the Definitions in the Microprogram Listing.

EXAMPLE #1

```
000130 83B260150 ..... 3548 STH ..... LR ... MDR,YD .....
```

This assembly statement has been assigned the branch label STH. Branch labels are optional and are only assigned to assembly statements to which other assembly statements will branch.

By inspecting the MSD of the OBJECT CODE we can immediately tell that this is an RR microinstruction (microinstruction op-code '8'). Knowing the microinstruction format in advance will usually aid in reading the assembly statement.

The assembly statement op-code is LR. The format of this assembly statement is found on page B-2, it is the first format in Table B-1. Its format is:

LR regd,regs,flr,MOD=mod,DEST=INHIB

In our example regd is MDR and regs is YD, and all the other options of this microinstruction are not specified which means they are not implemented. The underlined portion of the statement is the only part of the statement that can be left unspecified (defaulted). The regd is the destination register and regs is the source register. MDR and YD are names the microprogrammer assigned to two of the registers in the HMP-1116.

The function performed by the assembly statement is found on page B-4, it is the first entry in Table B-2. It reads:

LR Load Register regd←MOD(regs)

This tells us that this microinstruction will load the destination register with data from the source register that is modified by the Byte Manipulator. The MOD in the assembly statementis specifying Byte Manipulator control. The meaning of MOD is defined in Table B-3 on page B-8. Since it is unspecified in our assembly statement (default), data will pass through the Byte Manipulator unchanged.

In Table B-3 regd and regs are defined only as register operands. The specific definitions of registers in the HMP-1116 are found in the Definitions in the Listing. Both of the registers in this assembly statement are found in the list of external registers.

The external register definitions are shown below:

```
                    2440 ### EXTERNAL REGISTERS
                    2441 #
                    2442          EXTRN CTR,MDR,IR,MAR,PSW,EXT,YSP1,YS,YD,YDP1,FLR,RAR
                    2443          REGEQU (CTR,MDR,IR,MAR,PSW,EXT,YSP1,YS,YD,YDP1,FLR,RAR),
                                         (1,2,3,4,5,6,8,9,10,11,14,15)
00001               2444+#CTR     EQU   1
00002               2445+#MDR     EQU   2
00003               2446+#IR      EQU   3
00004               2447+#MAR     EQU   4
00005               2448+#PSW     EQU   5
00006               2449+#EXT     EQU   6
00008               2450+#YSP1    EQU   8
00009               2451+#YS      EQU   9
0000A               2452+#YD      EQU   10
0000B               2453+#YDP1    EQU   11
0000E               2454+#FLR     EQU   14
0000F               2455+#RAR     EQU   15
```

In statement number 2445 "MDR EQU 2" tells us that MDR is the name the microprogrammer has assigned to external register '2'.

In statement number 2452 "YD EQU 10" tells us that YD is the name the microprogrammer has assigned to external register 'A'.

In table 4-10 (in Chapter 4) the external registers are defined. External register '2' is the Memory Data Register, and external register 'A' is the General Register in File 1 specified by the R1 field of a software instruction.

This microinstruction will load the Memory Data Register with data from the General Register in File 1 specified by the R1 field of a software instruction.

In Chapter 4 on page 4-3 we analyzed the same microinstruction by breaking it into binary fields. Compare the hex code of the microinstruction on page 4-3 with the hex code of the microinstruction we just decoded (OBJECT CODE). Also compare the operations described in the two examples.

EXAMPLE #2

```
000000 63B510800                    3281        LI    X8000,X'80',MOD=SWAP
```

This assembly statement does not have a branch label. From the microinstruction op-code (MSD of OBJECT CODE) we can tell its format is RI.

The assembly statement op-code is LI, its format is found in Table B-1, on page B-3:

LI regd,imm,flr,MOD=mod,DEST=INHIB

The function of the assembly statement is found in Table B-2 on page B-5:

LI Load Immediate regd←MOD(imm)

This assembly statement loads the destination register (regd) from an immeidate data field defined by imm. The immediate data passes through the Byte Manipulator and can be modified by MOD.

In this case, MOD=SWAP is defined on Page B-8 (Table B-3). Swap bytes means that the bytes on the B-bus will be exchanged by the Byte Manipulator.

In an RI microinstruction the 8-bit data field is placed on the 2 LSDs of the B-bus and the 2 MSDs of the B-bus are set to zeros.

The imm field defines the immediate data as hex '80'. On the B-bus that becomes '0080'. After passing through the Byte Manipulator it becomes '8000'. This 16-bit word is loaded into the destination register which the micro-programmer refers to as X8000. The definition of this register is found in the definitions of the internal registers. The definitions of the internal registers are shown below:

```
                2364 ***** INTERNAL REGISTERS
                2365 *
                2367            REGEQU ABORTAD,0        ABORT RETURN ADDRESS (NOT USED)
00000           2368+@ABORTAD   EQU    0
                2369            REGEQU TEMPYD,1         REGD OPERAND FOR HW INSTRUCTIONS
00001           2370+@TEMPYD    EQU    1
                2371            REGEQU X00FF,2          CONSTANT VALUE X'00FF'
00002           2372+@X00FF      EQU    2
                2373            REGEQU X0004,3          CONSTANT VALUE 4
00003           2374+@X0004     EQU    3
                2375            REGEQU X0002,4          CONSTANT VALUE 2
00004           2376+@X0002     EQU    4
                2377            REGEQU X8000,5          CONSTANT VALUE X'8000'
00005           2378+@X8000     EQU    5
                2379            REGEQU LOCX,6           TEMP FOR LOC
00006           2380+@LOCX      EQU    6
                2381            REGEQU LOC,7            LOCATION COUNTER
00007           2382+@LOC       EQU    7
                2384 *                  (8,9,A,B,C,D,E,F)       RESERVED FOR LOCAL USE
                2385            REGEQU (SR8,SR9,SRA,SRB,SRC,SRD,SRE,SRF),
                                       (8,9,10,11,12,13,14,15)
00008           2386+@SR8       EQU    8
00009           2387+@SR9       EQU    9
0000A           2388+@SRA       EQU    10
0000B           2389+@SRB       EQU    11
0000C           2390+@SRC       EQU    12
0000D           2391+@SRD       EQU    13
0000E           2392+@SRE       EQU    14
0000F           2393+@SRF       EQU    15
```

In statement number 2378 "X8000 EQU 5" defines X8000 to be internal register 5. A comment states that internal register 5 contains a "Constant Value '8000'".

This microinstruction will load '8000' into internal register 5. This is the microinstruction the microprogram uses to initialize internal register 5. The same microinstruction was decoded in Example #2 in Chapter 4. Compare the results of the two examples.

EXAMPLE #3

```
000135 C1B1B2540                3562          LR  ..YD,TEMPYD,ST=JMC,DC=IFCH,MC=MRI1 .....
```

From the MSD of the OBJECT CODE we can tell that this is a CTL micro-instruction.

The assembly statement op-code is LR. Note that in Table B-1 there are two op-codes LR, the first is for an LR, RR format and the second is an LR, CTL format and it is:

LR regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt

The LR function is:

LR Load Register regd ← regs

(the MOD function is not applicable to a CTL microinstruction)

The operation will be to load regd(YD) from regs(TEMPYD). YD we saw in Example #1, it is the General Register in File 1 specified by the R1 field of a software instruction. TEMPYD is found in the definitions of the internal registers, TEMPYD is internal register 1. Therefore, this microinstruction will load the General Register specified by R1 with data from internal register 1.

ST=JMC is defined in Table B-3 on page B-9. This causes CVGL (the FLR) to be loaded into the Condition Code (PSW(12:15)), and zeros are loaded into CVGL.

DC=IFCH is not defined in Table B-3. It is defined in the Definitions in the Listing and is shown below:

```
.............  2480 ***  DC=... DEFINITIONS ..........................................
.............  2481 *  .....................................................................
.....................  .....................................................................
........  00001  2483 PFDC  ...  EQU  1  ..........................  PRIMARY FUNCTION DECODE ..
........  00002  2484 SFDC  ...  EQU  2  ..........................  SECONDARY FUNCTION DECODE .
.......,  00003  2485 IFCH  ...  EQU  3  ..........................  INSTRUCTION FETCH .........
```

In statement number 2485 "IFCH EQU 3" tells us that the DC field of this microinstruction will be set to '3'. Table 4-13 tells us that this is an Instruction Fetch (Interrupt Check). The signifigance of this operation is discussed in section 5-3.

The meaning of MC=MRI1 is also found in the Definitions in the Listing and it is shown below:

```
.........  2468 ***  MC=... DEFINITIONS ..........................................
.........  2469 *  .....................................................................
.....................  .....................................................................
..  00001  2471 MRI1  ....  EQU  1  ..........................  READ INSTN (LOAD MDR & IR)  ....
..  00002  2472 MRI2  ....  EQU  2  ..........................  READ 2ND WORD OF INSTN (LOAD MDR
.  00003  2473 MRO  .....  EQU  3  ..........................  READ OPERAND ..................
.  00004  2474 MROH  ....  EQU  4  .....................................................
.  00005  2475 MW  ......  EQU  5  ..........................  WRITE .........................
.  00006  2476 MWP  .....  EQU  6  ..........................  WRITE, PRIVILEGED .............
.  00007  2477 MWR  .....  EQU  7  .....................................................
```

In statement number 2471 "MRI1 EQU 1" tells us the MC field of this microinstruction will be set to '1'. Table 4-14 tells us that this operation is read the First Half (16 bits) of a software instruction. The comment by the definition in the Listing tells us that the MDR and IR will be loaded from memory.

This microinstruction actually specifies four operations. 1) Register transfer, 2) Condition Code load, 3) Interrupt Check and 4) read from memory.

The same microinstruction was decoded as Example #3 in Chapter 4. Compare the results of the two examples.

EXAMPLE #4

```
00003C 51B800210 ..... 3037 ..... OUT  SR8,ADRS,SYNC .....
```

This is an IO microinstruction. The assembly statement op-code is OUT, its format is:

OUT reg,oflag,iflag,MOD=mod,BC=oc

and its function is:

OUT Output output bus←reg

This microinstruction is sending the specified register (SR8) out on the I/O Mux Bus. SR8 is internal register 8.

The microprogrammer must also specify the output flag (oflag) and the input flag (iflag) used in the I/O operation. The oflag is ADRS. The definitions of the input and output flags are in the Listing. They are shown below.

```
.....  2488 ***  INPUT/OUTPUT FLAG DEFINITIONS .....
.....  2489 *  .....
.....  00001  2491 ADRS ... EQU  1 .....
.....  00002  2492 CMD .... EQU  2 .....
.....  00003  2493 DA ..... EQU  3 .....
.....  00004  2494 DR ..... EQU  4 .....
.....  00005  2495 ACK .... EQU  5 .....
.....  00006  2496 DACK ... EQU  6 .....
.....  00007  2497 SR ..... EQU  7 .....
.....  00001  2498 SYNC ... EQU  1 .....
.....  00002  2499 ESYNC .. EQU  2 .....
.....  00003  2500 FSYNC .. EQU  3 .....
```

In statement number 2491 "ADRS EQU 1" tells us that the 0F field of the microinstruction will be set to '1'. From Table 4-15 we find that this assembly statement specifies the Address flag. All IO microinstructions in the HMP-1116 will specify SYNC as the input flag. The IF field will be set to '1'.

This microinstruction will place internal register 8 on the I/O data bus, and set the Address flag. The microprogram will continue only after the SYNC flag is received.

The same microinstruction was decoded in Chapter 4, Example #4. Compare the results of the two examples.

EXAMPLE #5

```
0000E3 15B027FE0 ..... 3293 ..... BCT  G,XXPRGLD .....
```

This microinstruction is a BR format. The assembly statement format is:

BCT cond,addr

and its function is:

BCT	Branch on Condition True	cond false: no operation cond ture: RAR ← addr

This microinstruction will branch if the branch condition is true, that is if at least one of the flags specified by cond is set. In this example cond is G which means that the branch will be taken only if the G flag is set. The code placed in the MSK field of the microinstruction will be a function of cond, and it is defined in Table B-3.

The addr is a branch label assigned to an assembly statement somewhere in the Listing. If this microinstruction branches, then it will branch to the assembly statement labeled XXPRGLD. The easiest way to find the branch address is to read it directly from the second, third and fourth LSDs of the OBJECT CODE. It is '7FE'.

If the G flag is set, this microinstruction will branch to '7FE'.

The same microinstruction was decoded in Example #5 in Chapter 4. Compare the results of the two examples.

SUMMARY

As you grow more familiar with the Microprogram Listing you will find that you can read most assembly statements without referring to Appendix B or the Definitions. Once you develope this ability you will find the Microprogram Listing provides a concise and complete description of all the inner workings of the HMP-1116.

(Prints, sheet 4)

5.3 Software Execution Cycle. The microsequences required to execute software instructions and service interrupts are linked together by start addresses generated by the Instruction Decode Rom. The Instruction Decode Rom is divided into three functions; 1) Interrupt Check(IFCH), 2) Primary Function Decode(PFDC) and 3) Secondary Function Decode(SFDC). The truth tables for these functions are on pages 5-11 and 5-12. The Software Execution Cycle is begun by a CTL microinstruction with DC=IFCH.

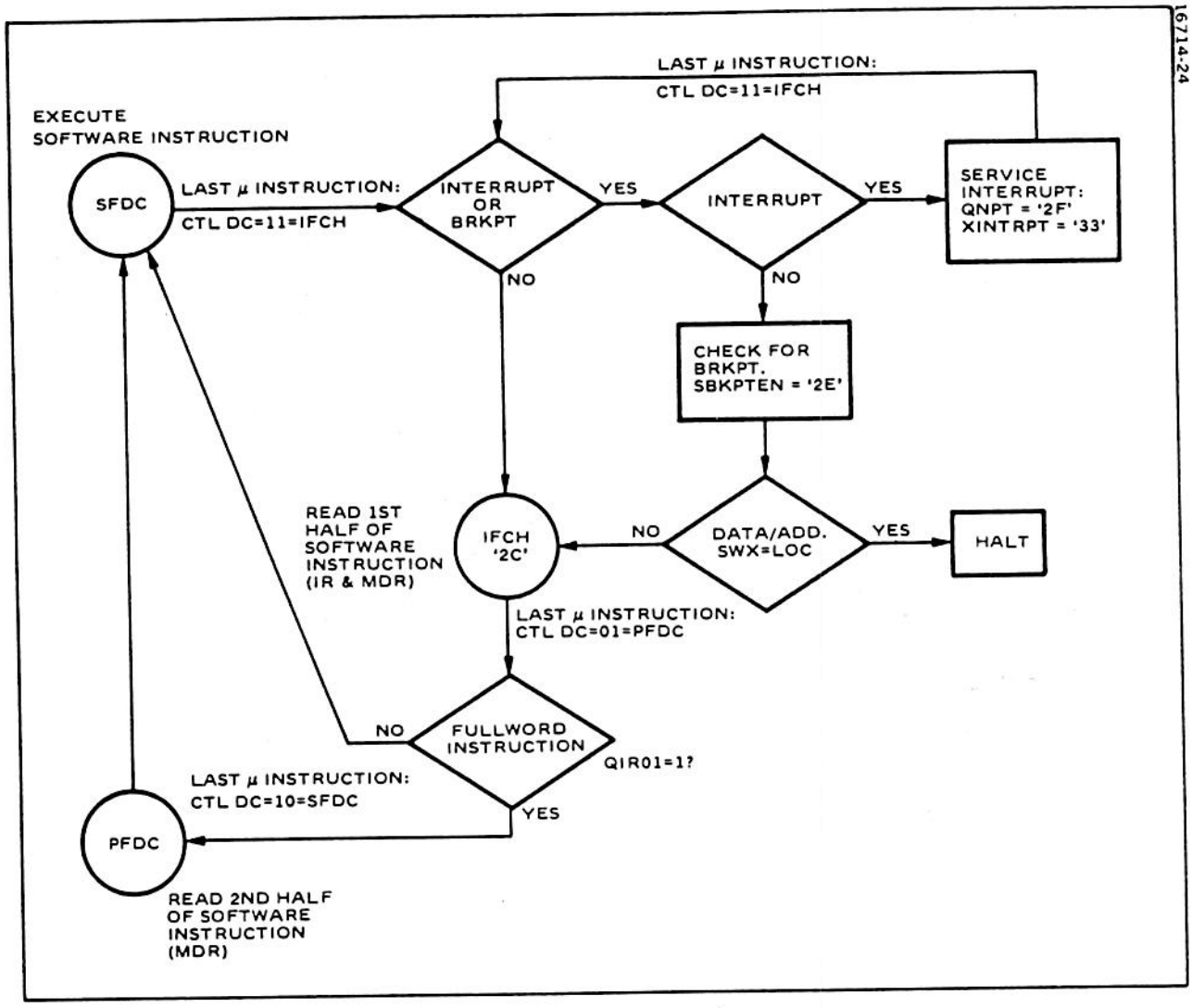
Then we check for breakpoint

If no interrupts are pending when an IFCH is generated, the Instruction Decode Rom generates the address of the Common Instruction Fetch Routine. This routine increments the Location Counter (PSW(16:31)) by two. The last microinstruction in this routine is a CTL which has DC=PFDC.

When a CTL microinstruction has DC=PFDC either a PFDC or an SFDC will occur. If the software instruction being decoded is a halfword instruction an SFDC is generated. If the software instruction being decoded is a fullword instruction a PFDC is generated.

Based on the software op-code the PFDC will generate one of ten microsequence start addresses. These ten microsequences are used for setting up the second operand of a fullword instruction. Each routine fetches the second half of the software instruction from memory, performs the necessary indexing and additional data fetching and adds two to the Location Counter. The last microinstruction in a PFDC routine is a CTL which has DC=SFDC.

Based on the software op-code the SFDC generates the start address of the microsequence which performs the operations unique to the software instruction. The last microinstruction in the microsequence is a CTL that causes the next software instruction to be read from memory and has DC=IFCH.



Software Execution Flow

IFCH DECODE ROM

QNPT	XINTRPT	SBKPTEN	OUTPUT	
1	X	X	02F	ABORT INTERRUPT
0	1	X	033	NORMAL INTERRUPT
0	0	1	02E	BREAKPOINT
0	0	0	02C	COMMON INSTRUCTION FETCH

PFDC DECODE ROM (4 sheet 4)

THESE ARE ALL FULLWORD SOFTWARE INSTRUCTION OP-CODES (QIR01=1)

OP-CODES	Type
40	RXAD
41-43	RI
44-4F	RXHW
50	RXDP
51-53	RXHW
54	RI
55	RXDP
56-57	RI
58	RXDP
59	RXAD
5A-5D	RXDP
5E-5F	RI
60	RXAD
61	RXOP
62-63	RI
64-6D	RXOP
6E-6F	RI
70	RXAD
71-73	RI
74-7D	RXAD
7E-7F	RI
C0-C1	RI
C2	RXAD
C3-CF	RI
D0-D1	RXAD
D2-D4	RXOP
D5	RXAD
D6-D7	RXOP
D8	RXHW
D9	RXAD
DA-DB	RXOP
DC	RXHW
DD-DF	RXOP
E0-F0	RI
F1-F4	RXOP
F5-F7	RI
F8-FB	RXAD
FC-FF	RI

PFDC DECODE ROM OUTPUT

TYPE	$\overline{X}$NDXRZO	OUTPUT	
RI	0	004	RINOX
	1	008	RIX
RXAD	0	00C	RXADNOX
	1	010	RXADX
RXOP	0	014	RXOPNOX
	1	018	RXOPX
RXHW	0	01C	RXHWNOX
	1	020	RXHWX
RXDP	0	024	RXDPNOX
	1	028	RXDPX

$\overline{X}$NDXZRO = 0 WHEN
X2-FIELD OF
SOFTWARE INSTRUCTION
IS ZERO (NO INDEX)

Addresses 057, 058, 059 are the locations for Illegal or Privileged Instruction Interrupt routines. Privileged Instruction Op-Codes are listed twice on the chart.

SFDC DECODE ROM T.T

OP-CODES	QPSW07	OUT-PUT	OP-CODES	QPSW07	OUT-PUT	OP-CODES	QPSW07	OUT-PUT	OP-CODES	QPSW07	OUT-PUT
00	X	059	45	X	149	96	1	059	D7	1	058
01	X	1A9	46	X	158	97	0	282	D8	0	262
02	X	17E	47	X	15C	97	1	059	D8	1	058
03	X	190	48	X	12E	98	0	261	D9	0	255
04	X	152	49	X	14C	98	1	059	D9	1	058
05	X	147	4A	X	13D	99	0	24B	DA	0	247
06	X	156	4B	X	145	99	1	059	DA	1	058
07	X	15A	4C	X	1F1	9A	0	244	DB	0	240
08	X	12C	4D	X	20B	9A	1	059	DB	1	058
09	X	14B	2E	X	13C	9B	0	207	DC	X	1FA
0A	X	13A	4F	X	144	9B	1	059	DD	0	23B
0B	X	142	50	X	409	9C	X	1F9	DD	1	058
0C	X	1F0	51	X	347	9D	0	236	DE	0	203
0D	X	20A	52	X	34D	9D	1	059	DE	1	058
0E	X	139	53	X	343	9E	0	200	DF	0	23A
0F	X	141	54	X	057	9E	1	059	DF	1	058
10	X	349	55	X	412	9F	0	235	E0	X	057
11	X	34F	56-57	X	057	9F	1	059	E1	X	2AA
12	X	345	58	X	404	A0-Af	X	059	E2	0	076
13-14	X	059	59	X	421	B0	X	496	E?	1	057
15	X	40E	5A	X	42F	B1	X	49B	E3	0	340
16-17	X	059	5B	X	43B	B2	X	4B6	E3	1	057
18	X	400	5C	X	447	B3-B4	X	4E7	E4	0	33C
19	X	418	5D	X	463	B5-BF	X	059	E4	1	057
1A	X	42A	5E-5F	X	057	C0-C1	X	19A	E5	X	057
1B	X	436	60	X	057	C2	0	03F	E6	X	600
1C	X	442	61	X	136	C2	1	058	E7	X	60C
1D	X	45E	62-63	X	057	C3	X	15E	E8	X	619
1E-1F	X	059	64-65	X	2E9	C4	X	154	E9	X	62C
20	X	172	66-67	X	30D	C5	X	149	EA	X	1C9
21	X	178	68-6F	X	057	C6	X	158	EB	X	1C3
22	X	188	70	X	642	C7	X	15C	EC	X	1BD
23	X	18C	71-73	X	057	C8	X	12E	ED	X	1B2
24	X	128	74	X	2D9	C9	X	14C	EE	X	1E8
25	X	12A	75	X	2DD	CA	X	13D	EF	X	1D8
26	X	134	76	X	2E1	CB	X	145	F0	X	057
27	X	13F	77	X	2E5	CC	X	1B8	F1	X	4A0
28-2D	X	059	78-7A	X	67B	CD	X	1AD	F2	X	4BF
2E	X	32D	7B	X	655	CE	X	1E1	F3-F4	X	4EE
2F	X	333	7C-7D	X	67B	CF	X	1CF	F5-F7	X	057
30-37	X	059	7E-7F	X	057	D0	X	0D1	F8	X	2C9
38-3A	X	657	80-8F	X	059	D1	X	0F2	F9	X	2CD
3B	X	651	90	X	1B8	D2	X	169	FA	X	2D1
3C-3D	X	657	91	X	1AD	D3	X	162	FB	X	2D5
3E-3F	X	059	92	X	165	D4	X	16E	FC-FF	X	057
40	X	130	93	X	160	D5	0	269			
41	X	1AA	94	X	16C	D5	1	058			
42	X	183	95	0	2A6	D6	0	287			
43	X	195	95	1	059	D6	1	058			
44	X	154	96	0	282	D7	0	287			

7.0 I/O SOFTWARE PROGRAMMING

7.1 Introduction. An I/O software instruction will cause information to be transferred on the I/O Mus Bus. The information transferred can be either a device address, command byte, status byte or data word. The processor identifies the information being transferred by setting a flag.

A device address is an 8-bit number assigned to an I/O device. The I/O Mux Bus is made up of common data lines and control flags going to all devices. Therefore, the processor needs a way of selecting only one of the devices to communicate with. The processor uses a device address to select a device. Only the addressed (selected) device can receive a command from the processor, send its status to the processor or transfer data.

A command byte is an 8-bit word the processor can send to the addressed device. The command controls the device. For example, a command can tell a device to do a DMA transfer.

A status byte is an 8-bit word the addressed device can send to the processor. The status of a device tells the processor about the condition of the device. For example, it can tell the processor if the device is busy doing an operation or if it is having problems completing an operation.

Data transferred will be either 8-bit or 16-bit words. The addressed device controls the Halfword Flag (THW) which when set tells the processor that the device transfers 16 bits of data at a time. If THW is not set, the device transfers only 8 bits of data at a time. A write is from the processor to a device, and a read is from a device to the processor.

The basic I/O operations a programmer can perform are 1) Acknowledge Interrupt, 2) Sense Status, 3) Output Command, 4) Write Data and 5) Read Data. The processor breaks these operations into two or more IO microinstructions. The microinstructions performed by the processor for each type of operation is shown in Table 7-1.

Table 7-1 only shows the Read and Write Halfword instructions. There are other types of read and write instructions but only Read Halfword and Write Halfword can be used with 16-bit oriented devices. All other read and write instructions only transfer bytes of data. When using the RX format of a byte transfer instruction, an even memory address refers to the MSB byte of a memory location, and an odd memory address refers to the LSB byte of a memory location.

Table 7-1. I/O Instruction Sequences

SOFTWARE INSTRUCTION	MICROINSTRUCTION OPERATIONS	
	FLAG	OPERATION
Acknowledge Interrupt(AIR)	TACK	Interrupting device places its address on TD08-15 and sets TSYN.
Sense Status(SSR)	TADRS	Processor places device address on TD08-15 and waits for TSYN.
	TSR	Device places its status on TD08-15 and sets TSYN.
Output Command(OCR)	TADRS	Processor places device address on TD08-15 and waits for TSYN.
	TCMD	Processor places command on TD08-15 and waits for TSYN.
Write Halfword(WHR)	TADRS	Processor places device address on TD08-15 and waits for TSYN.
		IF DEVICE SETS THW:
	TDA	Processor places data on TD00-15 and waits for TSYN.
		IF DEVICE DOES NOT SET THW:
	TDA	Processor places 8 MSBs of halfword on TD08-15 and waits for TSYN.
	TDA	Processor places 8 LSBs of halfword on TD08-15 and waits for TSYN.
Read Halfword(RHR)	TADRS	Processor places device address on TD08-15 and waits for TSYN.
		IF DEVICE SETS THW:
	TDR	Device places data on TD00-15 and sets TSYN.
		IF DEVICE DOES NOT SET THW:
	TDR	Device places first 8 bits of data on TD08-15 and sets TSYN.
	TDR	Device places second 8 bits of data on TD08-15 and sets TSYN.

DMAB, PCC are only 16bit I/O
others are byte devices

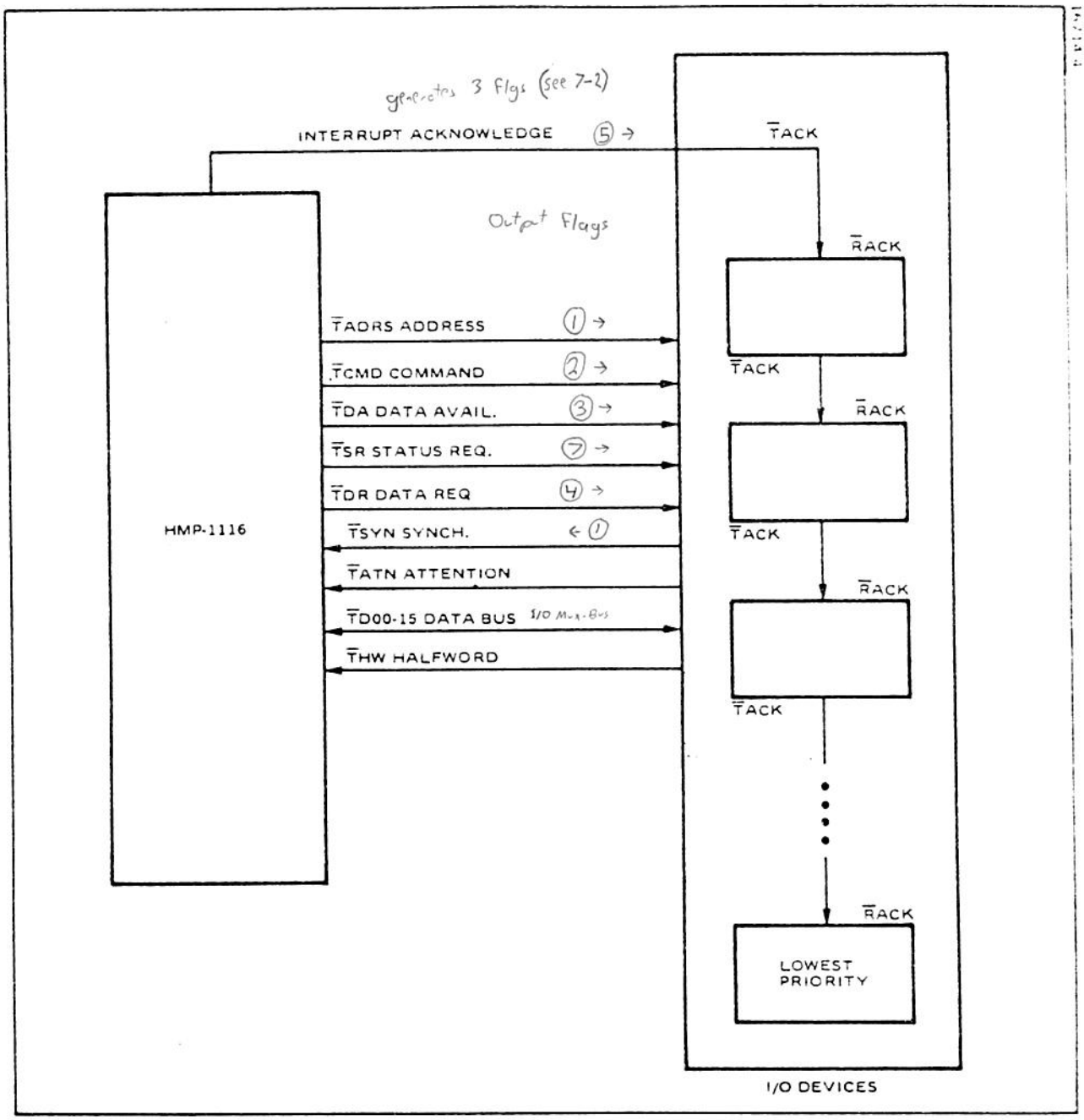
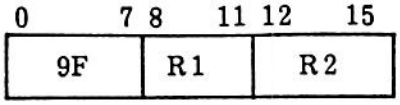


Figure 1-2. HMP-1116 I/O Mux Bus Interface

7.2 I/O Software Instruction Formats

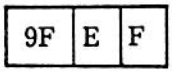
ACKNOWLEDGE INTERRUPT (AIR)



(RR)

This instruction acknowledges an I/O interrupt and places the interrupting device's address in the General Register specified by R1. Then that address is used to address the device and its status is requested. The status of the device is placed in the General Register specified by R2

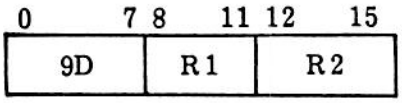
Example:



Reg. E(8:15)←Interrupting device's address
Reg. F(8:15)←Interrupting device's status

The address of the interrupting device is placed in bits 8 through 15 of General Register E, and its status is placed in bits 8 through 15 of General Register F.

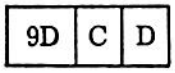
SENSE STATUS (SSR)



(RR)

Bits 8 through 15 of the General Register specified by R1 are used to address a device. The status of the addressed device is requested and placed in bits 8 through 15 of the General Register specified by R2.

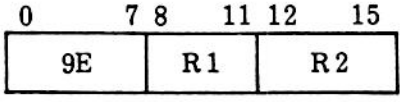
Example:



Address device ← Reg. C(8:15)
Reg. D(8:15) ← Addressed device's status

The status byte of the device specified by General Register C is placed in General Register D.

OUTPUT COMMAND (OCR)



(RR)

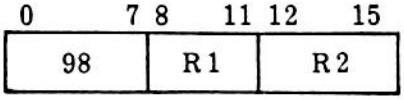
The General Register specified by R2 contains a command for the device specified by the address in the General Register specified by R1.

Example:

Address ← Reg. A(8:15)
Command ← Reg. B(8:15)

The device specified by General Register A bits 8 through 15 is sent the command in bits 8 through 15 of General Register B.

WRITE HALFWORD (WHR)



(RR)

The General Register specified by R1 contains a device address. The General Register specified by R2 contains 16 bits of data for that device.

Example:

98	8	9

Address ← Reg. 8(8:15)

Data ← Reg. 9(0:15) } ——— (If THW is set)

Data ← Reg. 9(0:7)
Data ← Reg. 9(8:15) } ——— (If THW is not set)

The device specified by General Register 8 is sent the 16 bits of data in General Register 9. The data is sent as a 16-bit halfword if THW is set, or it is sent as two 8-bit bytes if THW is not set.

READ HALFWORD (RHR)

0 — 7	8 — 11	12 — 15
99	R1	R2

(RR)

The General Register specified by R1 contains a device address. The General Register specified by R2 will be loaded with 16 bits of data received from the device.

Example:

99	6	7

Address ← Reg. 6(8:15)

Reg. 7(0:15) ← Data } ——— (If THW is set)

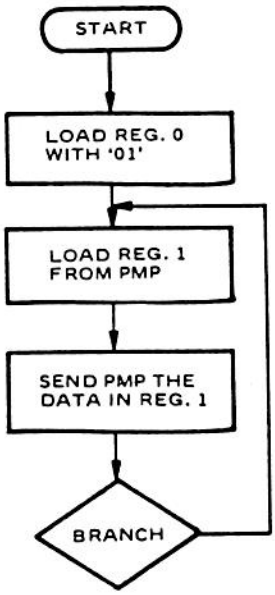
Reg. 7(0:7) ← Data
Reg. 7(8:15) ← Data } ——— (If THW is not set)

The processor receives 16 bits of data from the device specified by General Register 6. The data is loaded into General Register 7. The data is received as a 16-bit halfword if THW is set, or it is received as two 8-bit bytes if THW is not set.

7.3 Sample Programs. The following programs are written using the Processor Maintenance Panel (PMP) as an I/O device. The PMP is a byte device and it is assigned device address '01'. The Thumbwheel switches are data read from the PMP, and the Displays are data written to the PMP. When writing bytes of data to the PMP, the first byte sent will be placed in the two LSDs of Display 2 and the second byte written will be placed in the next two LSDs of Display 2. The MSD of Display 2 will remain a '0'. When reading data from the PMP, the first byte read will be the two LSDs of the Thumbwheel switches, the second will be the next two LSDs of the Thumbwheel switches.

I/O PROGRAM #1

Continuously display the contents of the Thumbwheel switches in Display 2.



16714-25

Put PMP device address in General Register 0 (LIS)

Load Thumbwheel switches into General Register 1 (RHR)

Write the contents of General Register 1 to Display 2 (WHR)

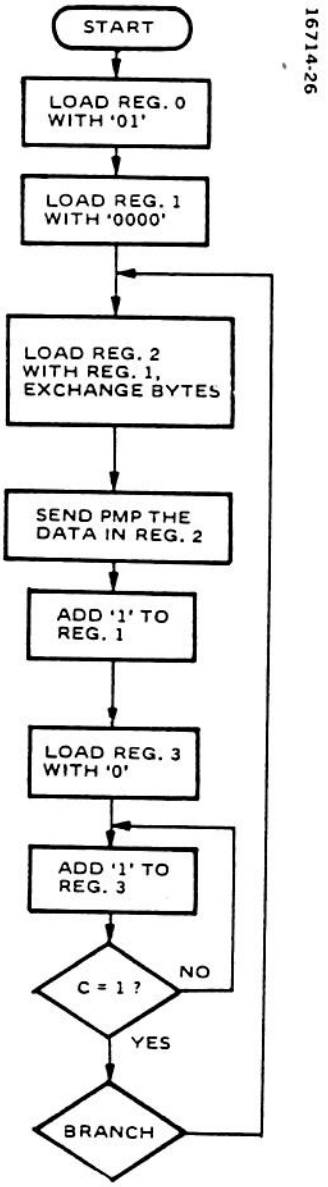
Branch unconditionally to repeat Read and Write (BFBS)

MEMORY LOCATION	MEMORY CONTENTS	INSTRUCTION MNEMONIC	COMMENTS
1000	2401	LIS	(Reg. 0) ← '01'
1002	9901	RHR	Address ← (Reg. 0), (Reg. 1) ← Data
1004	9801	WHR	Address ← (Reg. 0), Data ← (Reg. 1)
1006	2202	BFBS	Branch to '1002'

I/O PROGRAM #2

Make Display 2 count from '0000' to 'FFFF', continuously. (Must be visible)

I break this program into three main functions, 1) Initialization of Registers, 2) Display and Increment Count and 3) Arbitrary Delay to slow count enough to make it visible.



1) Initialization

Reg. 0 contains device address (LIS)

Initialize count to zero (LIS)

2) Display and Increment Count

Since the panel displays the first byte sent (the MSBs) in the two LSDs of Display 2, it is necessary to exchange the bytes of the count before display. (EXBR)

Write count to Display 2 (WHR)

Increment Count (AIS)

3) Delay

Initialize delay values (LIS)

Add '1' to delay value (AIS)

Continue adding until delay value carries out of 16-bit register (BFBS)

Branch unconditionally to update display

MEMORY LOCATION	MEMORY CONTENTS	INSTRUCTION MNEMONIC	COMMENTS
			1) Initialization
1000	2401	LIS	(Reg. 0) ← '01'
1002	2410	LIS	(Reg. 1) ← '0000'
			2) Display and Increment Count
1004	9421	EXBR	Reg. 2(0:7) ← Reg. 1(8:15) Reg. 2(8:15) ← Reg. 1(0:7)
1006	9802	WHR	Address ← (Reg. 0), Data ← (Reg. 2)
1008	2611	AIS	(Reg. 1) ← (Reg. 1) + '1'
			3) Delay
100A	2430	LIS	(Reg. 3) ← '0'
100C	2631	AIS	(Reg. 3) ← (Reg. 3) + '1'
100E	2281	BFBS	If (Reg. 3) < '10000' then branch to '100C'
1010	2206	BFBS	Branch to '1004'

7.4 External I/O Interrupt Service

When the computer is interrupted via the I/O mux bus a software program may be used to service the interrupt. This program is called the external interrupt service routine. The computer jumps to the service routine by changing PSWs. When an I/O interrupt is detected the computer saves the current PSW in memory locations '0040' and '0042' and loads the PSW from memory locations '0044' and '0046'. Prior to the interrupt the programmer must load memory locations '0044' and '0046' with a PSW that will point to the interrupt service routine.

When interrupt occurs, the current PSW is saved in memory.	['0040'] ← PSW(0:15) ['0042'] ← PSW(16:31)
Then the PSW is loaded from memory with a value that will point at the interrupt service routine.	PSW(0:15) ← ['0044'] PSW(16:31) ←['0046']

The current PSW must have PSW bit 1 set to enable the I/O interrupt to be serviced. The new PSW must not have bit 1 set.

When the service routine has completed servicing the interrupt, the normal procedure is to return to the interrupted program by executing an LPSW instruction using memory location '0040'.

7.5 I/O Mux Bus Hardware and Timing

The I/O Mux Bus and its control are shown in the Detailed Functional Block Diagram in Chapter 3. The four functions controlling the I/O Mux Bus are the I/O Mux Bus and Flag Timing Control, Output Flag Control, Input Flag Control and the Data Transceivers (XCVRS).

When the I/O Mux Bus and Flag Timing Control decodes an Input microinstruction (Op-code '4') it sets XIN (Input Data Enable), when it decodes an Output microinstruction(op-code '5') it sets QODE(Output Data Enable). XIN causes the XCVRS to place TD00-15 on the B-bus. QODE causes the XCVRS to place the A-bus on TD00-15. The I/O Mux Bus and Flag Timing Control also sends $\overline{X}$ERDSTP to the Multiple Clock Cycle Control when it decodes an Input or Output microinstruction.

When the Multiple Clock Cycle Control receives $\overline{X}$ERDSTP, it stops the microprogram on the IO microinstruction and sets QSMTH(Single Cycle Control) active. When the I/O Mux Bus and Flag Timing Control sees QSMTH it sends XOFE (Output Flag Enable) to the Output Flag Control. The Output Flag Control sets one of the output flags ($\overline{T}$CMD, $\overline{T}$SR, $\overline{T}$DR, $\overline{T}$DA $\overline{T}$ADRS or $\overline{T}$ACK) based on QRD22-26 (the OF field).

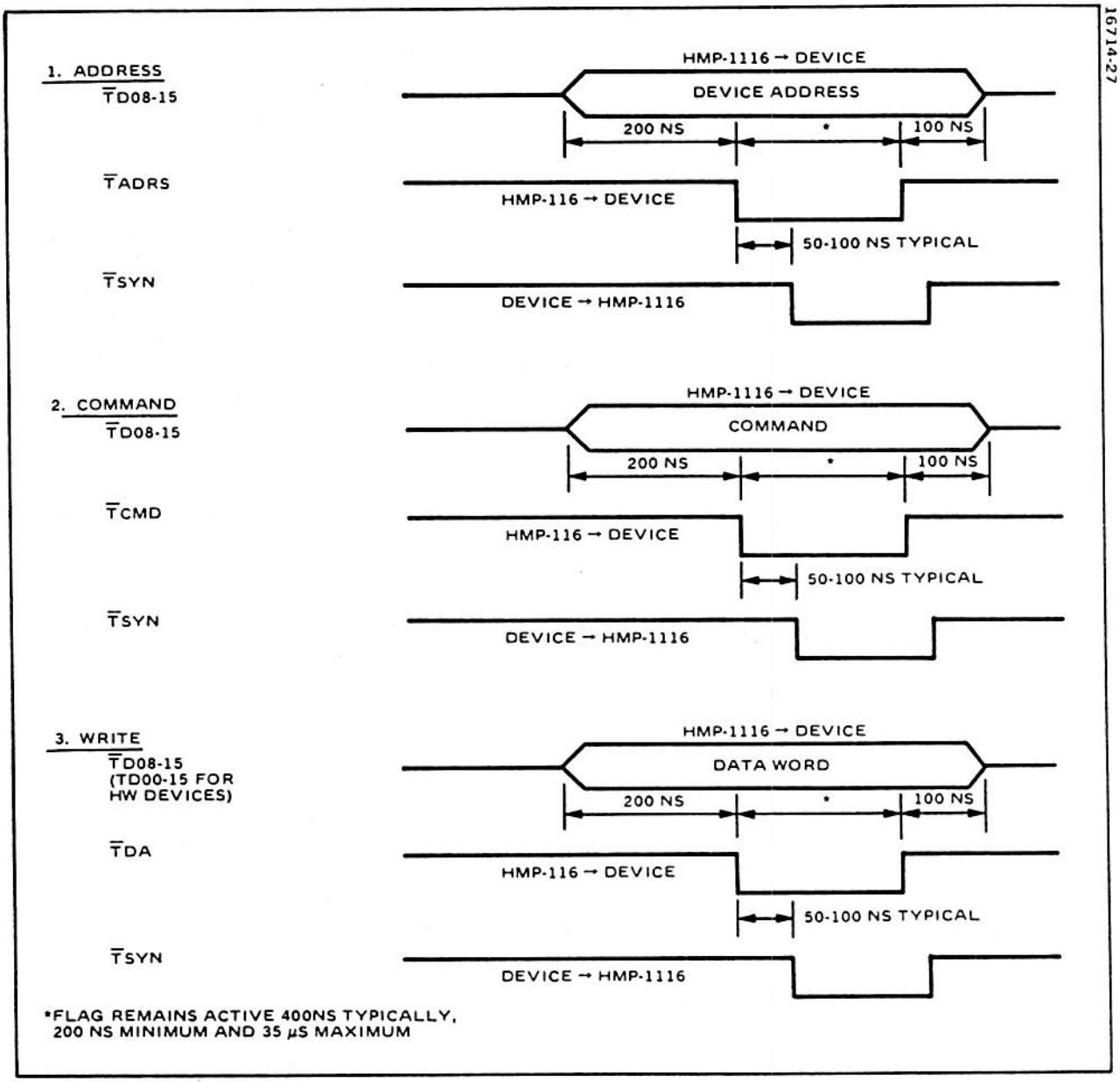
When the Input Flag Control receives TSYN (Sync Control) from a device it sends XCOMP (Sync Received) to the I/O Mux Bus and Flag Timing Control. The I/O Mus Bus and Flag Timing Control holds XERDSTP active until it receives XCOMP. If TSYN is not received and 35 microseconds elapse from when the output flag was set, the I/O Mux Bus and Flag Timing Control automatically terminates the I/O operation and allows the microprogram to continue.

Page 7-9 shows the timing of the flags and data for the three types of output microinstructions, Address, Command and Data Available.

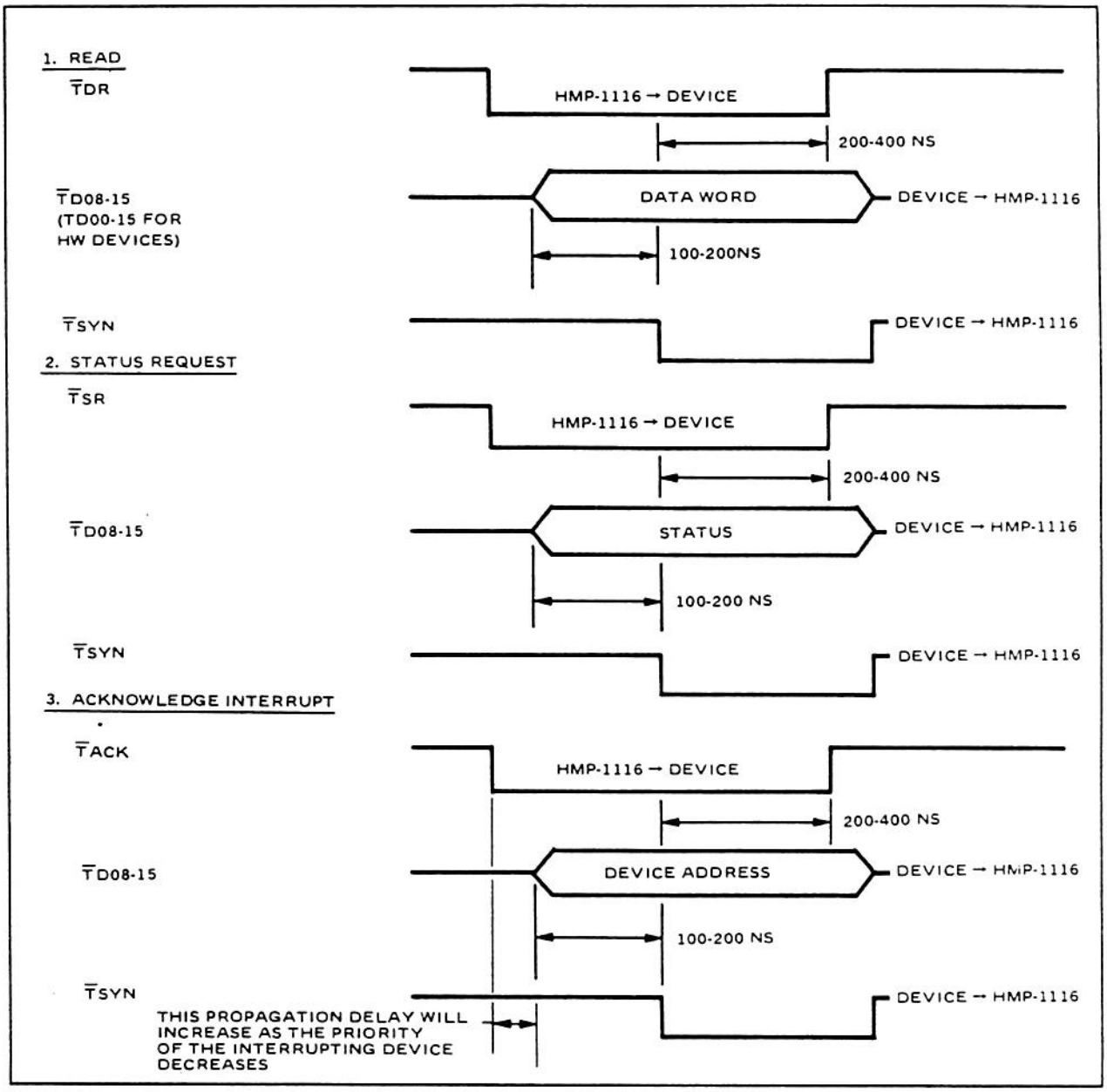
Page 7-10 shows the timing of the flags and data for the three types of input microinstructions, Data Request, Status Request and Acknowledge Interrupt.

Page 7-11 shows the detailed timing of an Output microinstruction.

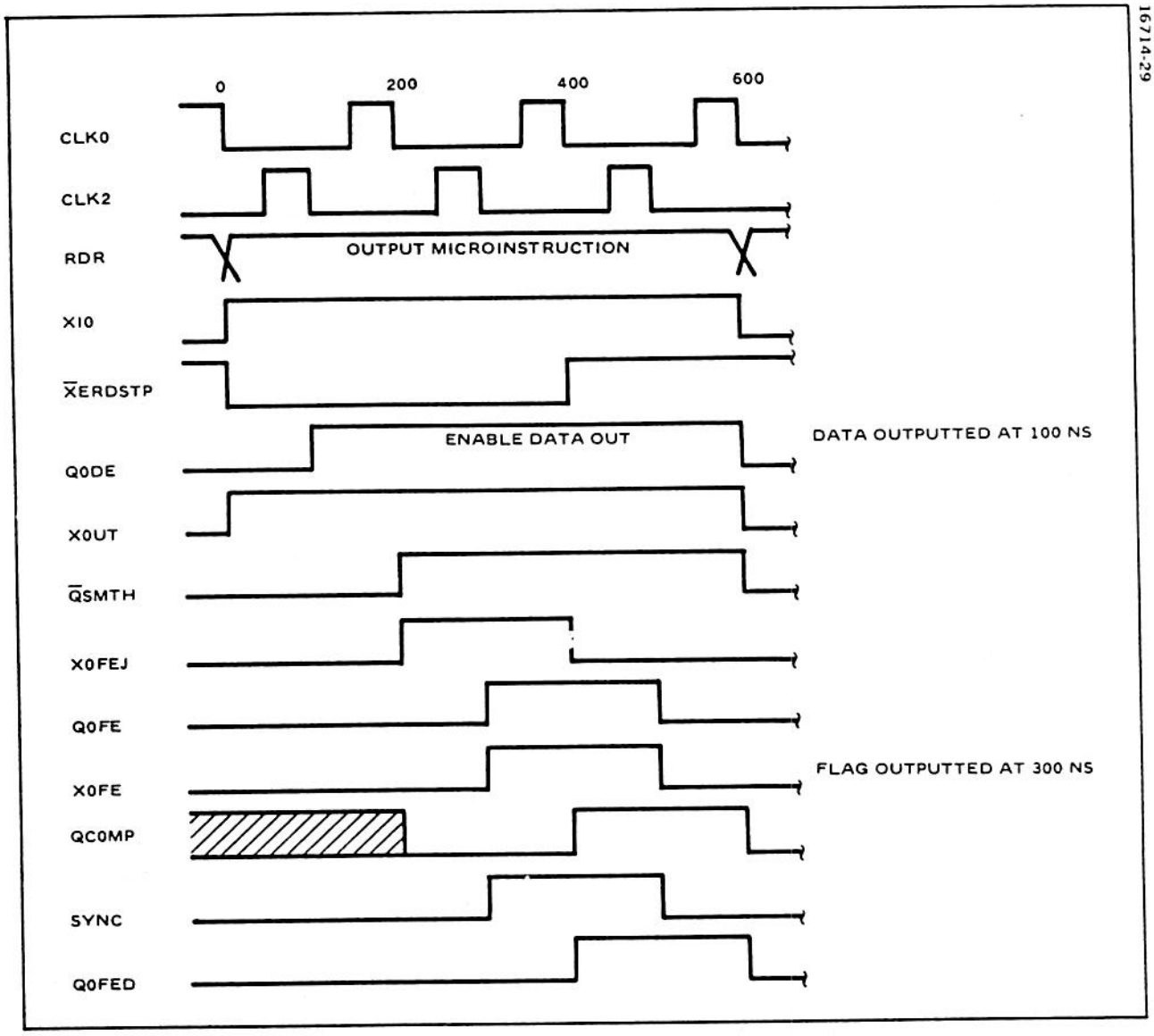
Page 7-12 shows the detailed timing of an Input microinstruction.



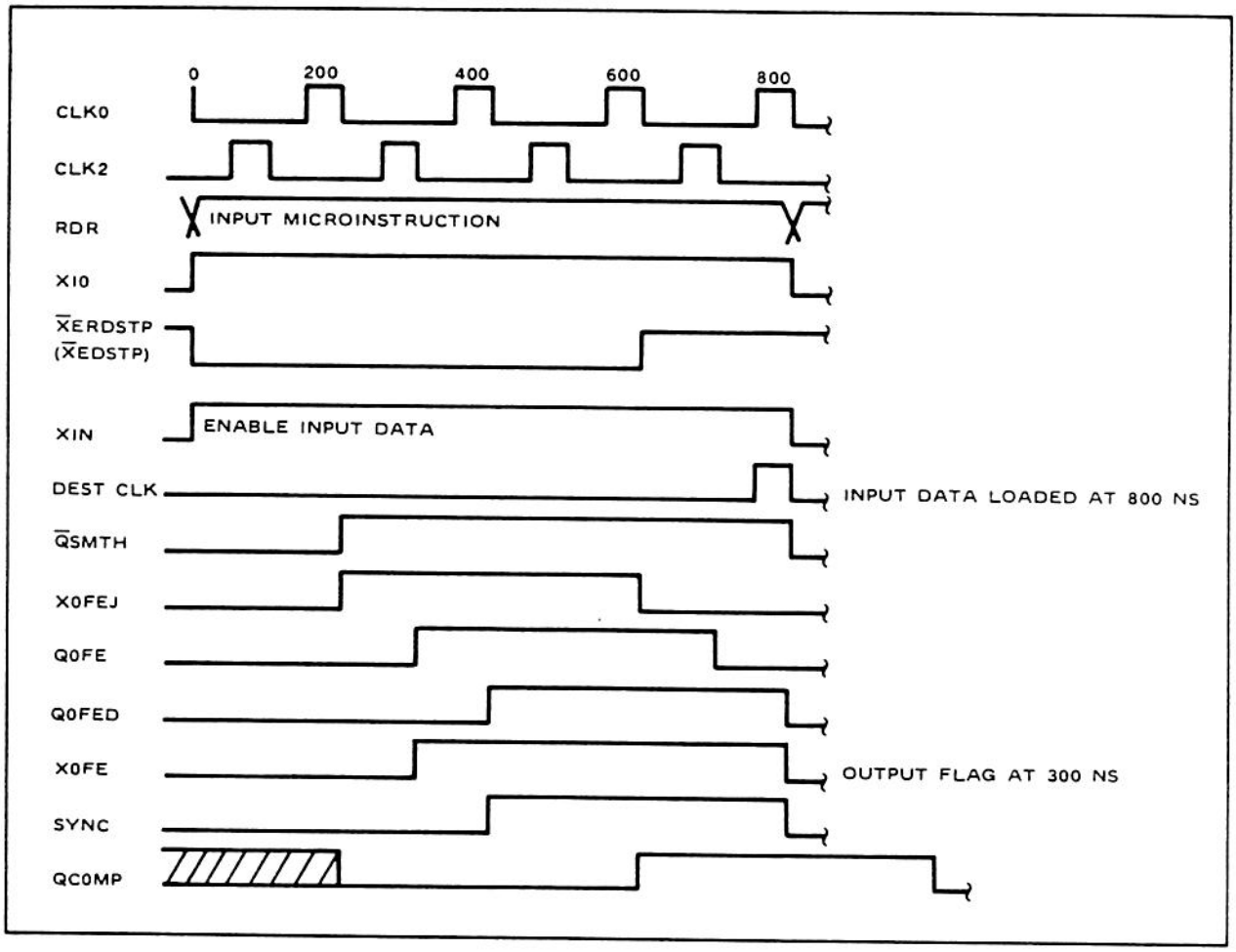
HMP-1116 IOM Bus Outputs



HMP-1116 IOM Bus Inputs



Output Microinstruction Timing



Input Microinstruction Timing

8.0 INTERRUPTS

8.1 Introduction. Interrupts are caused by either a fault condition within the HMP-1116 or a request from an external device. The computer services interrupt with either software or firmware routines.

When an interrupt occurs that is serviced by software, the microprogram exchanges the PSW using the appropriate reserved memory locations. The new PSW read from memory will point to the software service routine. The microprogram saves the old PSW in memory so that after the interrupt has been serviced the interrupt service routine can return to the interrupted program.

Firmware serviced interrupts are handled completely by the microprogram. After the interrupt has been serviced, the microprogram returns to the interrupted software program. Four interrupts are serviced by the microprogram; 1) PMP "Execute", 2) PMP "Single Mode", 3) Primary Power Failure and 4) Remote Program Load.

Section 8-2 outlines fault type interrupts internal to the HMP-1116, and how they are enabled, detected and serviced.

Section 8-3 describes I/O Mux Bus interrupt service.

Section 8-4 contains the Functional Flowcharts for the HMP-1116. These Flowcharts show how the microprogram responds to interrupts.

HMP-1116

INTERRUPT

TYPES

*1. Machine Malfunction

2. Fixed Point Divide Fault

3. Floating Point Divide Fault

4. Privilege Mode Violation

5. Illegal Instruction

Firmware interrupts

*6. I/O Interrupt

*7. Console Attention

*8. Single Mode

*9. Primary Power Fail

*Or'ed to generate Xintrpt to Instruction Decode ROM

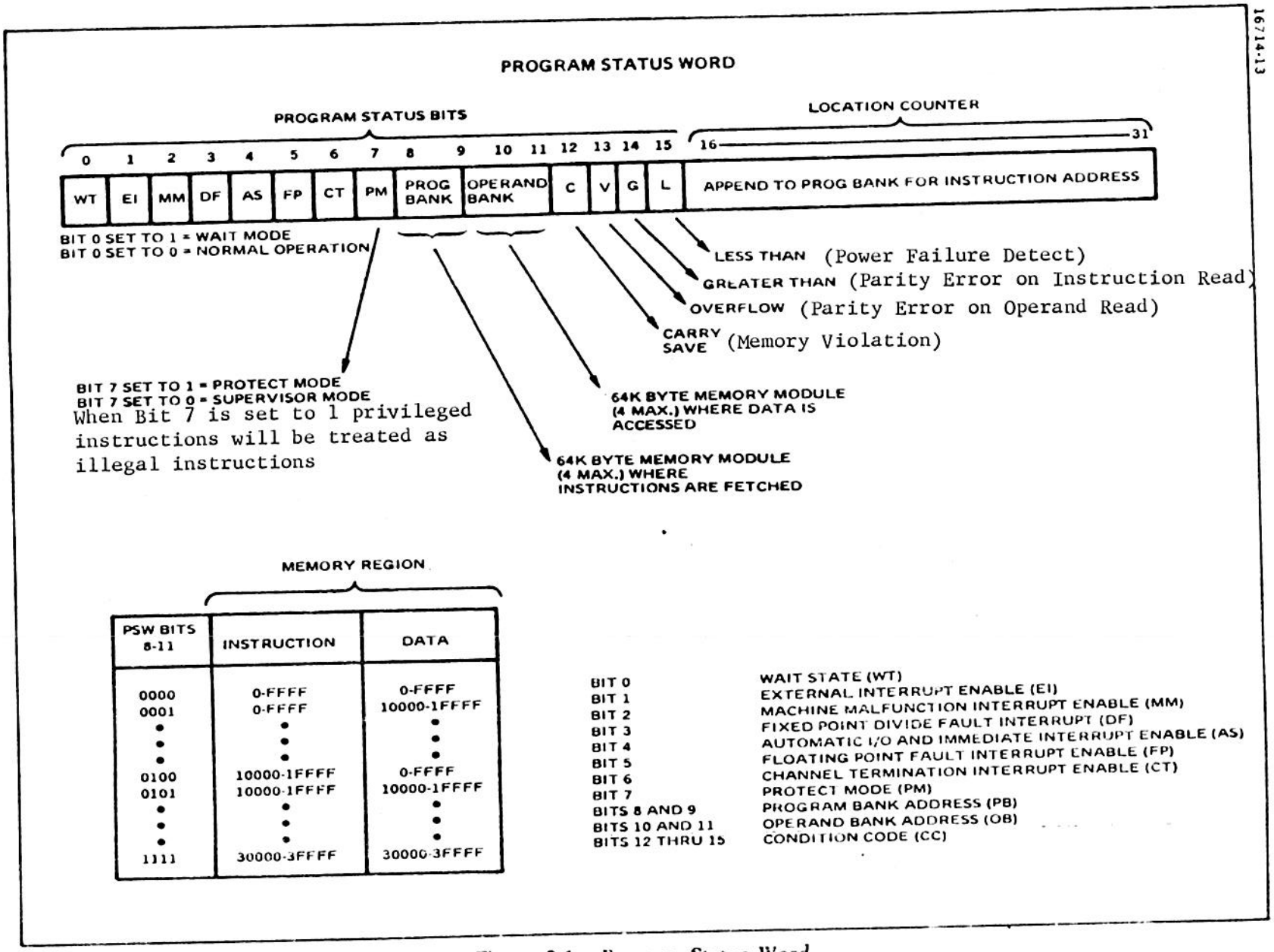


Figure 2-1. Program Status Word

*10. Remote Program Load

*Or'ed to generate Xintrpt to Instruction Decode ROM

8.2 Internal Interrupts

I. Machine Malfunction

A. Caused by:

1. Protected Memory Violation (C flag set)

2. Parity Error on Instruction Read (G flag set)

3. Parity Error on Operand Read (V flag set)

4. Early Power Failure (L flag set)

B. Enabled by PSW bit 2

C. Old PSW at '38', New PSW at '3C'

D. Detected between software instruction (IFCH)

II. Fixed-Point Divide Fault

A. Caused by divide Overflow

B. Enabled by PSW bit 3

C. Old PSW at '48', New PSW at '4C'

D. Detected by Divide Instruction

III. Floating-Point Arithmetic Fault

A. Caused by Floating Point Exponent Overflow or Underflow

B. Enabled by PSW bit 5

C. Old PSW at '28', New PSW at '2C'

D. Detected by Floating-Point Insruction

IV. Protect Mode Violation

A. Caused by Executing a Priveleged Instruction

B. Enabled by PSW bit 7

C. Old PSW at '30', New PSW at '34'

D. Detected when Software Op-code Decoded (SFDC)

V. Illegal Instuction

A. Caused by Executing an Undefined Instruction

B. Always Enabled

C. Old PSW at '30', New PSW at '34'

D. Detected when Software Op-code Decoded (SFDC)

8.3 I/O Mux Bus Interrupt Service. Any I/O device, including the Processor Maintenance Panel, the Memory Protect Card (MPC, Chapter 9), the Programmed Controlled Clocks (PCC, Chapter 9), or any other device connected to the I/O Mux Buss is considered an EXTERNAL device. The term EXTERNAL, in this situation imples EXTERNAL to the PROCESSOR itself.

External devices interrupt the Processor with $\overline{\text{T}}$ATN (attention flag) and thereby request some interactive servicing of their needs. External devices can have data for the Processor or Memory, require data from the Processor or Memory, or have some fault to report in a status message.

There are three methods available to the Processor for servicing external, I/O MUX Bus interrupt sources.

Two methods for servicing External, I/O interrupts are the IMMEDIATE INTERRUPT SERVICE and the CHANNEL COMMAND BLOCK method. These two methods are termed "Automatic" I/O servicing because the firmware automatically responds to $\overline{\text{T}}$ATN with the proper interrupt Acknowledge Signal ($\overline{\text{T}}$ACK). The Automatic I/O servicing methods are described in Chapter 8.

The third method for servicing External, I/O interrupts is the EXTERNAL I/O INTERRUPT SERVICE. This technique is described in Section 7-4 and is contrasted to the other two methods in Chapter 8, Figure 1.

The External I/O Interrupt Service is not considered "automatic". This is because the software programmer must determine the step-by-step procedure for servicing the interrupting device. The software must provide the Interrupt Acknowledge and provide the other steps in the routine.

I. PSW bit 1 must be set to enable I/O interrupts. A device will set the "Attention" line on the I/O Mux Bus when it wants to interrupt the HMP-1116. The computer detects this interrupt between software instructions (IFCH).

II. If PSW bit 4 is zero:

A. Old PSW at '40', New PSW at '44'

III. If PSW bit 4 is one:

A. Microprogram Acknowledges Interrupt

B. The Microprogram uses the interrupting device's address to calculate where in memory the device's Interrupt Pointer (IP) is located. The location of the IP is calculated as 'DO'+(2*Device Address). For example, device '01' has an Interrupt Pointer in memory location 'DO'+(2*'01') = 'D2'.

C. If bit 15 of IP is zero:

1. Old PSW at IP, New PSR at IP+4

2. IP+6 is New LOC

D. If bit 15 of IP is one:

1. IP points to Channel Command Word in Channel Command Block

2. The Channel Command Block controls a firmware routine which services the interrupt.

3. When a Channel Command Block finishes its assigned function it can place its IP in a list called the Channel Termination Queue. Entries in this Queue will result in an interrupt to the computer.

IV. Channel Termination Interrupt

A. Caused by an entry being made in the Channel Termination Queue

B. Enabled by PSW bit 6

C. Old PSW at '82', New PSW at '86'

D. Detected at end of Automatic I/O service or when PSW is changed

V. Channel Termination Queue Overflow Interrupt

A. Caused by Automatic I/O service trying to make an entry into the Channel Termination Queue when it is full

B. Always Enabled

C. Old PSW at '8C', New PSW at '90'

D. Detected when an entry is to be made into the full queue

The automatic input/output channel executes channel programs that control the activities of peripheral devices. The execution of channel programs takes place between the execution of user instructions resulting in a program delay rather than a program interrupt with an exchange of Program Status Words. The I/O channel may generate an interrupt because of abnormal conditions or because of the occurrence of an event for which the software had requested an Interrupt. Bits 1 and 4 of the Current Program Status Word control the operation of the I/O channel. Both of these bits must be set to permit channel operations. Channel operations also depend on the interrupt pointer table, the channel control block with its associated channel command word, and the channel termination queue; See Figure 1.

General Operation. With bits 1 and 4 of the current PSW set, the following occurs when the processor detects an interrupt: the processor automatically acknowledges the interrupt and obtains the device address. It uses the device address times two to index into the Interrupt pointer table. The interrupt pointer table starts at location x '00DO' and contains halfword entries for each of the 256 possible peripheral device addresses. The contents of the table locations point to either a service routine or a channel command as determined by the value of bit 15. If bit 15 of the entry is zero, the processor takes an immediate interrupt (Software Service). If bit 15 is one, the processor activates the I/O channel. The I/O channel uses the entry minus one to locate the channel command word. It decodes the channel command word and performs the required service using the entries in the channel command block as necessary. See Figure 2.

It should be noted that the automatic I/O channel works in bank zero regardless of the setting of the PSW bank indicators. Therefore, channel control blocks and interrupt handlers must be in bank zero.

If the channel operation for this device is not yet complete, the I/O channel returns control to the processor. The processor now checks for pending interrupt signals and, if none are present, continues program execution. If any are present, it services them before returning to program execution.

If the channel determines that the operation for this device is complete, it terminates the channel program by storing the device address and final status in the channel control block, and for data transfers, changes the channel command word to a no operation.

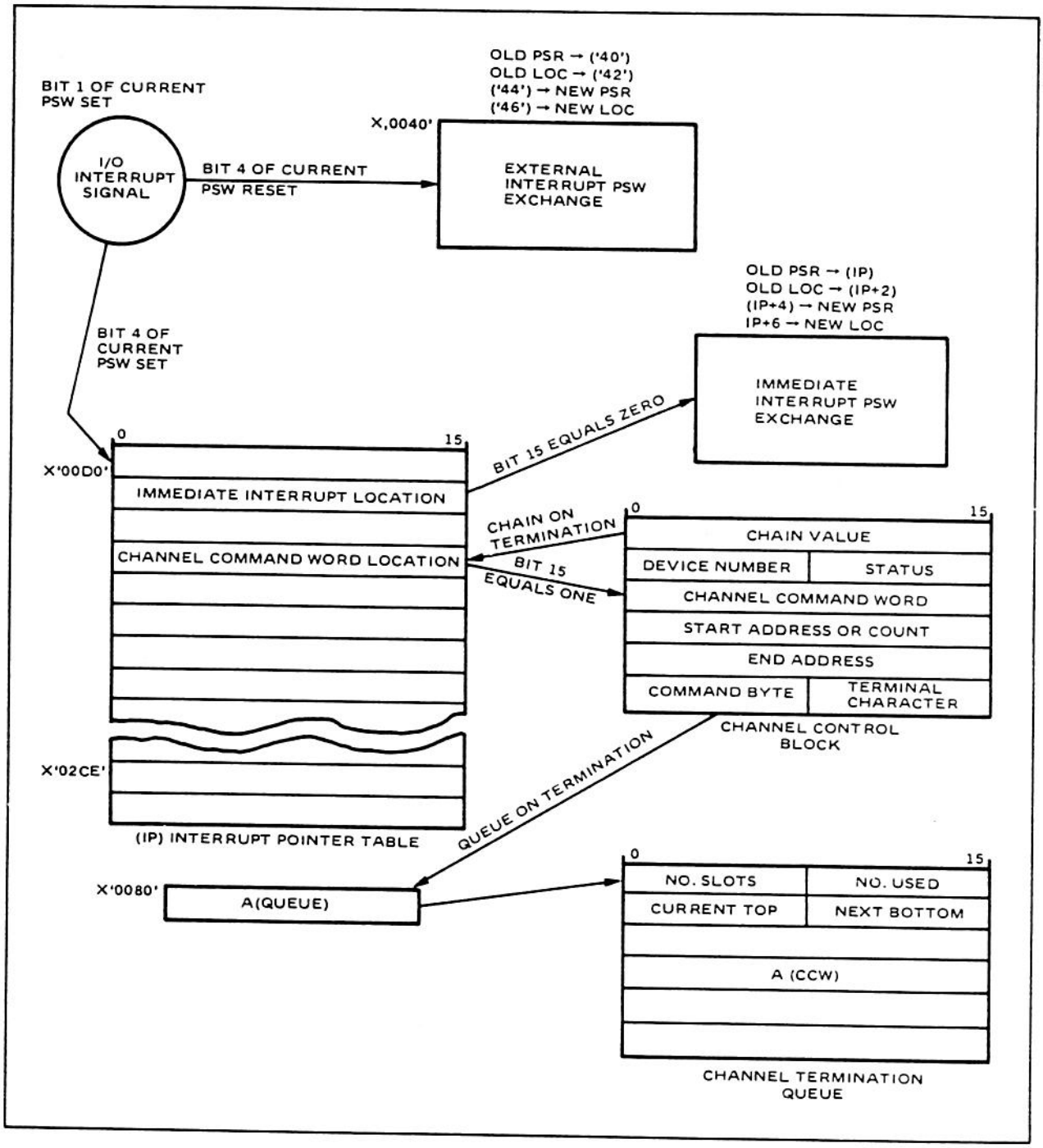

16714-31
BIT 1 OF CURRENT PSW SET
OLD PSR → ('40')
OLD LOC → ('42')
('44') → NEW PSR
('46') → NEW LOC
X,0040'
I/O INTERRUPT SIGNAL
BIT 4 OF CURRENT PSW RESET
EXTERNAL INTERRUPT PSW EXCHANGE
OLD PSR → (IP)
OLD LOC → (IP+2)
(IP+4) → NEW PSR
IP+6 → NEW LOC
BIT 4 OF CURRENT PSW SET
IMMEDIATE INTERRUPT PSW EXCHANGE
BIT 15 EQUALS ZERO
0
15
X'00D0'
IMMEDIATE INTERRUPT LOCATION
CHAIN ON TERMINATION
CHANNEL COMMAND WORD LOCATION
BIT 15 EQUALS ONE
CHAIN VALUE
DEVICE NUMBER
STATUS
CHANNEL COMMAND WORD
START ADDRESS OR COUNT
END ADDRESS
COMMAND BYTE
TERMINAL CHARACTER
CHANNEL CONTROL BLOCK
X'02CE'
(IP) INTERRUPT POINTER TABLE
QUEUE ON TERMINATION
X'0080'
A(QUEUE)
NO. SLOTS
NO. USED
CURRENT TOP
NEXT BOTTOM
A (CCW)
CHANNEL TERMINATION QUEUE

Figure 1.

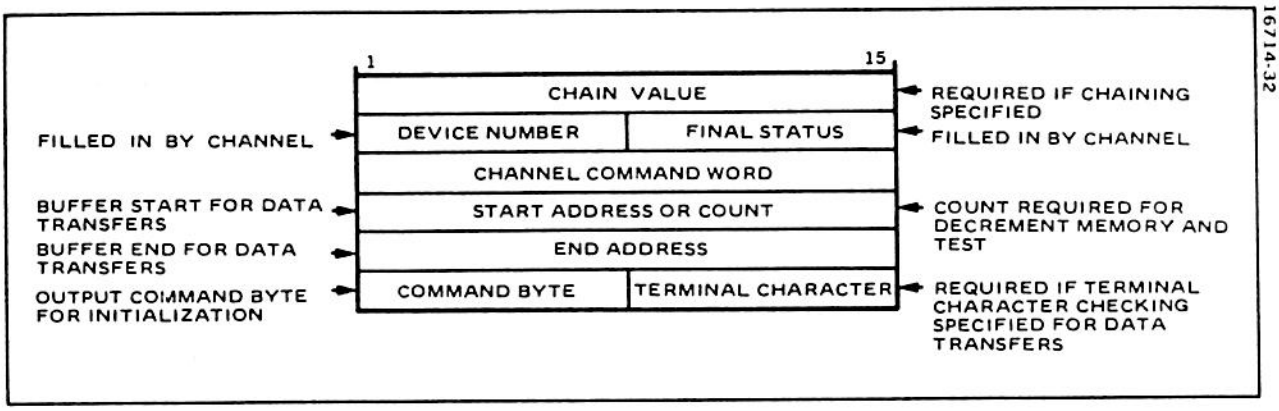


Figure 2. Channel Command Block

The I/O channel can now take any or all of the following actions:

1. Make an entry in the termination queue.
2. Chain to another channel command word.
3. Generate an immediate interrupt.

The action taken depends on the bit configuration of the channel command word. See Figures 3 and 4.

Channel Command Word. There are three phases involved in channel operations:

1. Initialization
2. I/O operation
3. Termination

All three phases are controlled by the bit configuration of the channel command word. A single command word can be encoded to perform all three types of operation.

Initialization. If bit 0 (INIT) is set when the channel decoded the command word, it resets bit 0 and checks bit 8 (output command). If bit 8 is set, the channel issues the output command located at the start of the channel control block plus ten and returns control to the processor. Channel operations with the device resume when an interrupt signal from the device occurs. To start channel operations, a simulate interrupt instruction is executed which will cause the channel to enter the initialization phase. The software may initialize the device by output command instructions after which channel operations would begin when an interrupt signal from the device occurs.

I/O Operations. There are five types of I/O operations the automatic I/O channel can perform. They are:

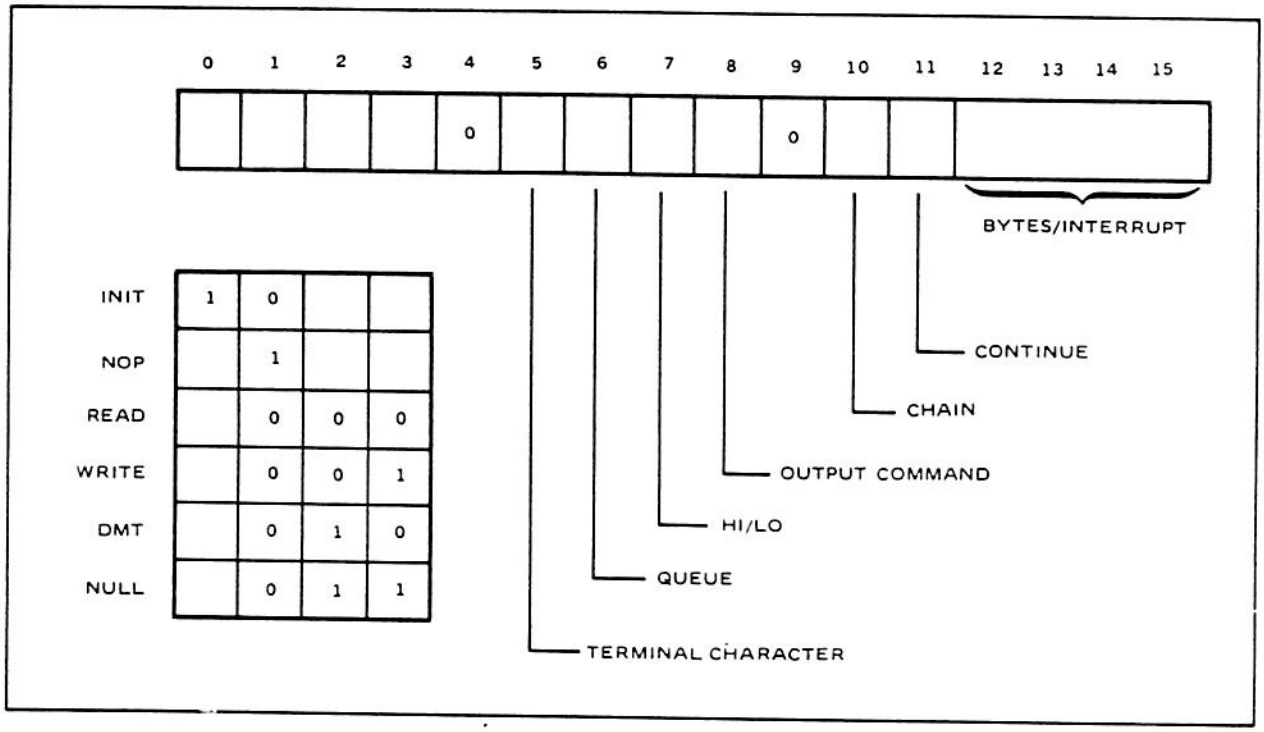


Figure 3. Bit Configuration for Channel Command Word

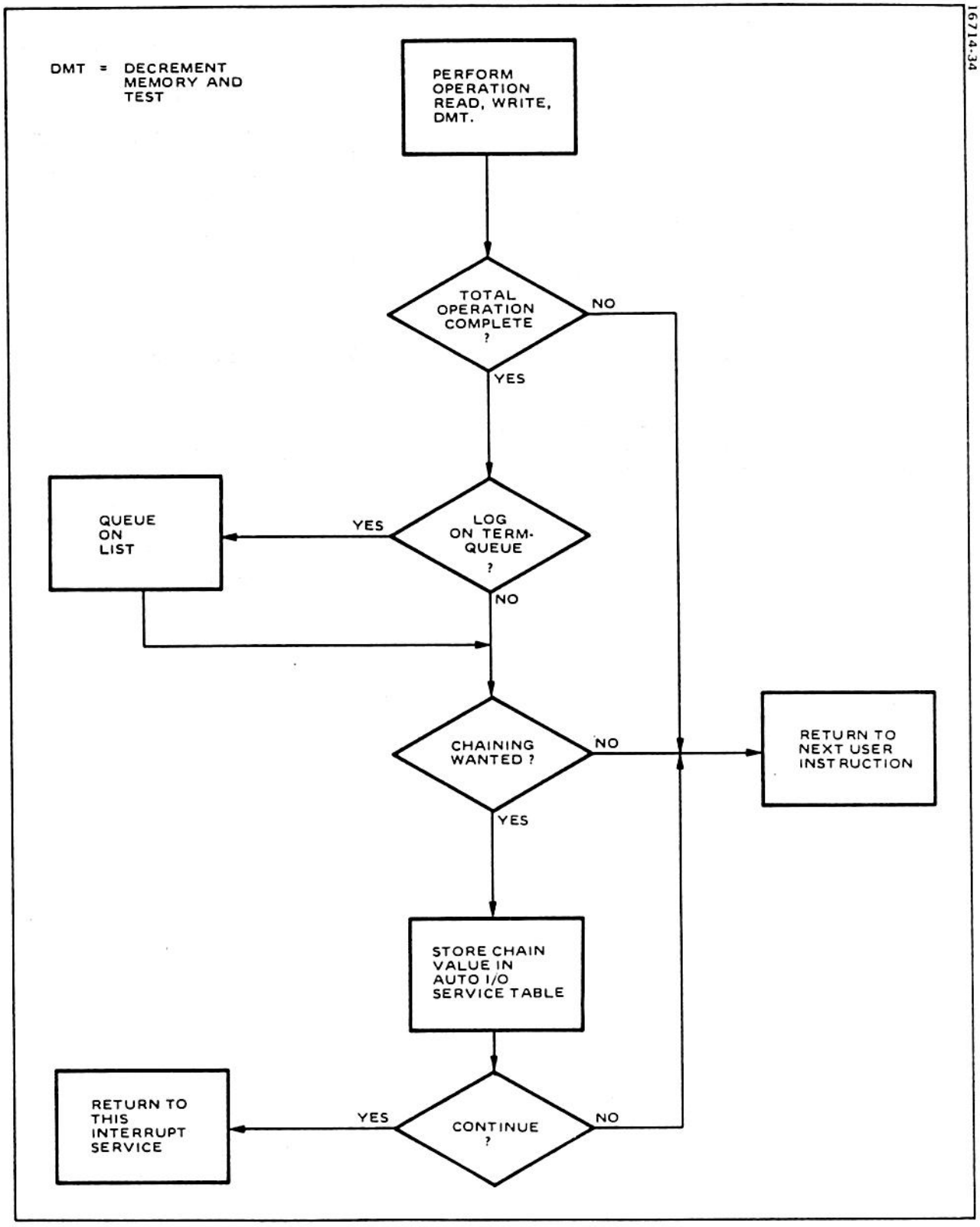


Figure 4. Automatic I/O

1. READ

2. WRITE

3. DECREMENT MEMORY AND TEST

4. NO OPERATION

5. NULL

For read and write operations, bits 12-15 must contain the number of bytes to be transferred on each interrupt signal. All zeros in these bit positions indicate that sixteen bytes are to be transferred on each interrupt signal. The next two halfwords following the channel command word must specify the beginning address of the I/O buffer and the ending address of the I/O buffer. After the specified number of bytes has been transferred, the starting address is incremented the appropriate amount and compared to the ending address. If it is greater, the channel enters the termination phase.

If it is less, the channel returns control to the processor for program execution.

Bit five controls the optional terminal character data transfer. When this bit is set, the transfer proceeds as described above with the exception that the last byte transferred on each interrupt signal is compared with the terminal character byte located at channel control block plus eleven. If these two bytes match, the channel enters the termination phase.

Before starting a data transfer, the Automatic I/O Channel checks the device status. Any non-zero status condition will stop the transfer and cause the channel to enter the termination phase. Before entering the termination phase, the Initial (INIT) bit and No Operation bit are set in the Channel Command Word, the Queue bit is set to force an entry in the Termination Queue, and the Chain bit and Continue bit are reset to prevent chaining.

The Decrement Memory and Test Operation causes the value contained in the halfword immediately following the Channel Command Word to be decremented by one for each interrupt signal. The new value is compared to zero. If greater than zero, the channel returns control to the Processor for program execution. If equal to zero, the channel enters the termination phase without changing the Channel Command Word to a "no operation". Subsequent interrupt signals from the device will cause the count field to increase negatively. The No Operation code in the Channel Command Word indicates that the channel is to ignore any interrupt signal from the associated device. The channel itself sets this code in the command word on completion of data transfers. The software can use this code to ignore unsolicited interrupt signals. The Automatic I/O Service Table should contain pointers to "no operation" control words for all non-existent devices.

The Null Operation differs from the No Operation in that, while no I/O function is performed, the channel enters the termination phase without setting the No Operation code.

Termination. The Automatic I/O Channel enters the termination phase upon completion of a data transfer, when the count field of a Decrement Memory and Test Operation has reached zero, or when the Null Operation is decoded. All of the operations in the termination phase are optional. If none are specified, the channel returns control to the Processor. The two termination functions are Queue and Chain. Bit 6 of the Channel Command Word controls queuing. If this bit is set, the channel, on entering the termination phase, stores the address of the Channel Command Word in the Channel Termination Queue. The condition of Bit 7 of the Channel Word controls positioning with the queue. If Bit 7 is set, the entry is made at the bottom of the queue. If Bit 7 is reset, the entry is made at the top of the queue.

The channel termination queue is located at the address specified by the termination queue pointer. The termination queue pointer is located in low core at X'80-81' with the channel I/O termination parameters. When the automatic I/O channel enters the termination phase with queuing specified, an automatic I/O channel termination interrupt is generated causing a PSW swap if bit 6 of the PSW is set. This notifies the software of the completion of a channel I/O operation.

If the processor attempts to enter an I/O channel termination pointer in the termination queue and the queue is already full, a channel termination queue overflow interrupt is generated.

The I/O channel termination pointer is saved and a PSW swap occurs allowing software to clear the queue before any channel I/O terminations are lost.

Bit 10 of the Channel Command Word controls chaining. In this operation, the channel stores the first halfword of the Channel Control Block in the appropriate location in the Automatic I/O Service Table for this device. This chain value may be either the address of another Channel Command Word or the address of PSW exchange location for the Immediate Interrupt. Subsequent interrupt signals will be handled as indicaed by this value. If the Chain bit and the Continue bit (Bit 11) are both set, the channel checks the new value placed in the Service Pointer Table and takes appropriate action before returning control to the Processor. In this way, depending on the new value stored in the Service Pointer Table, the channel can either generate an Immediate Interrupt or start another channel program.

Channel I/O Programming Example. This example is for the card reader whose physical address is X'04'. The progam is set up to:

1. Issue an output command to start the device.

2. Read 80 columns of data into memory, 2 bytes per interrupt signal.

3. On completion of the transfer, the automatic I/O channel enters the termination phase by chaining to an immediate interrupt and causing the interrupt to occur.

The channel command block is shown in Figure 5. The chain value points to an immediate interrupt location. The status byte and device number are set to zero.

The command word is set to initialize, output command, chain, and continue and transfer one byte per interrupt signal. The next two halfwords point to the beginning and ending buffer addresses. The output command is set to issue the Feed command to the car reader. A simulate interrupt is issued specifying device address X'04' to get the operation started. The channel then issues the output command, resets the initialize bit and gives control to the processor for the execution of normal instructions. As each interrupt signal is received from the device, the channel inputs two bytes until 160 bytes (80 colums) have been read in. Between each interrupt signal, control is returned to the processor.

After the last byte has been transferred, the no operation bit in the channel command word is set and stores the chain value in location X'D8', interrupt pointer table entry for device X'04'. An interrupt is now generated which informs the software that one card has been read. The software can now do the following to enable the I/O channel to read another card.

1. Put the address of the channel control block in the interrupt pointer table.

2. Reset the no operation bit, and set the initialize bit of the channel command word.

3. Reset the beginning buffer address (this may be determined by a Get Buffer routine).

4. Check device status for successful I/O.

5. Inform card reader command processor that the card reader data is ready for conversion and sending.

6. Set the device status and the device number to zero.

If, during the data transfer the channel had received a bad status from the card reader, it would have terminated the operation by setting the initialize and no operation bits in the channel command word, suppressed chaining, and forced on entry at the top of the channel termination queue.

8.4 Functional Flowcharts. The Functional Flowcharts summarize the main control functions of the HMP-1116 microprogram. They begin on sheet 1 with the Power Up Routine. This routine begins at microprogram address '000' and it is executed at power up and following every Master Clear.

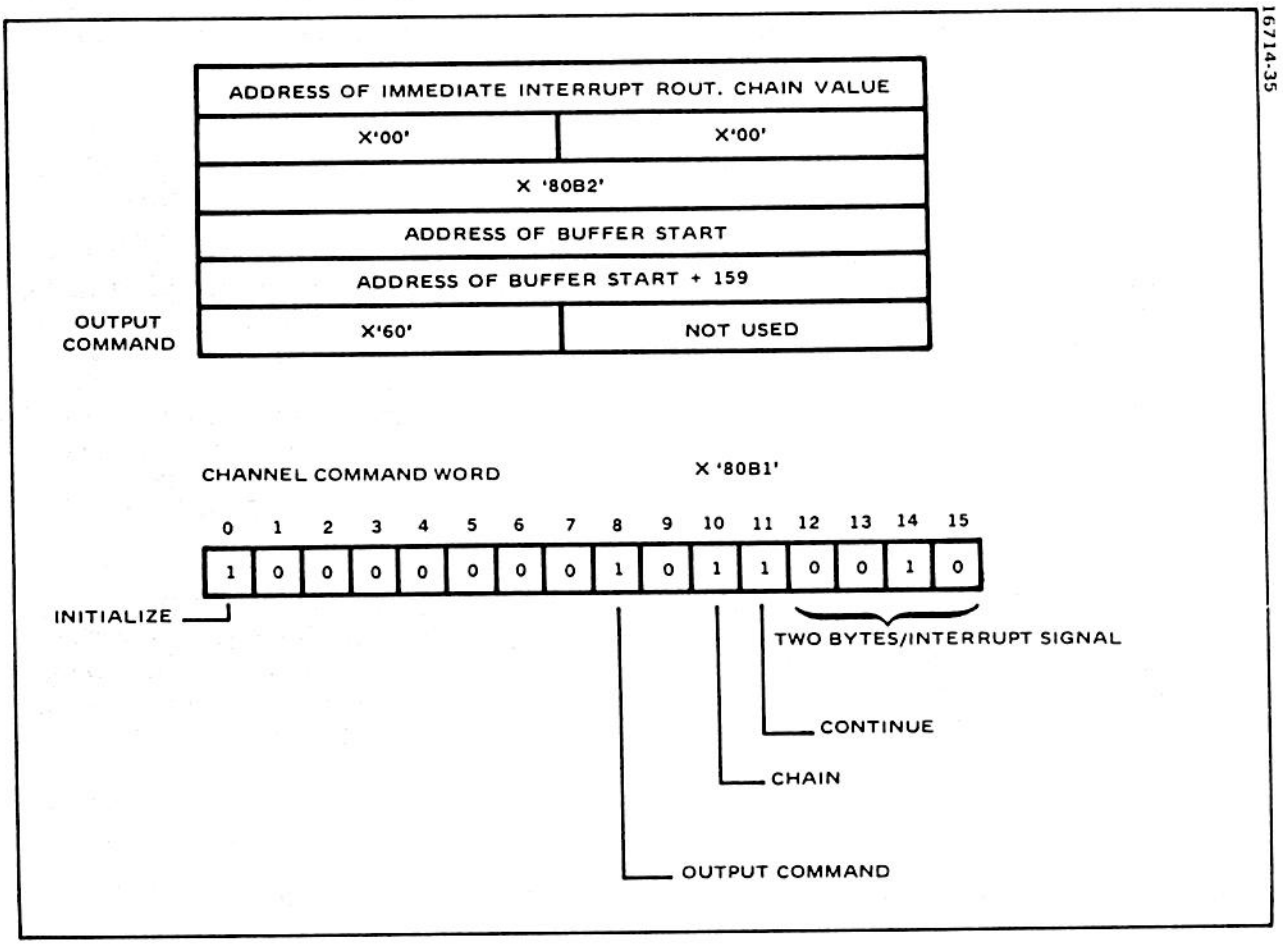


Figure 5. Channel Control Block for the Card Reader

Sheet 2 begins with the Wait Loop. The microprogram waits here after PSW bit 0 has been set. Any one of the following interrupts will cause the microprogaam to leave the Wait Loop:

1) Machine Malfunction (MALF)

2) Primary Power Failure (PPF) (any power fault)

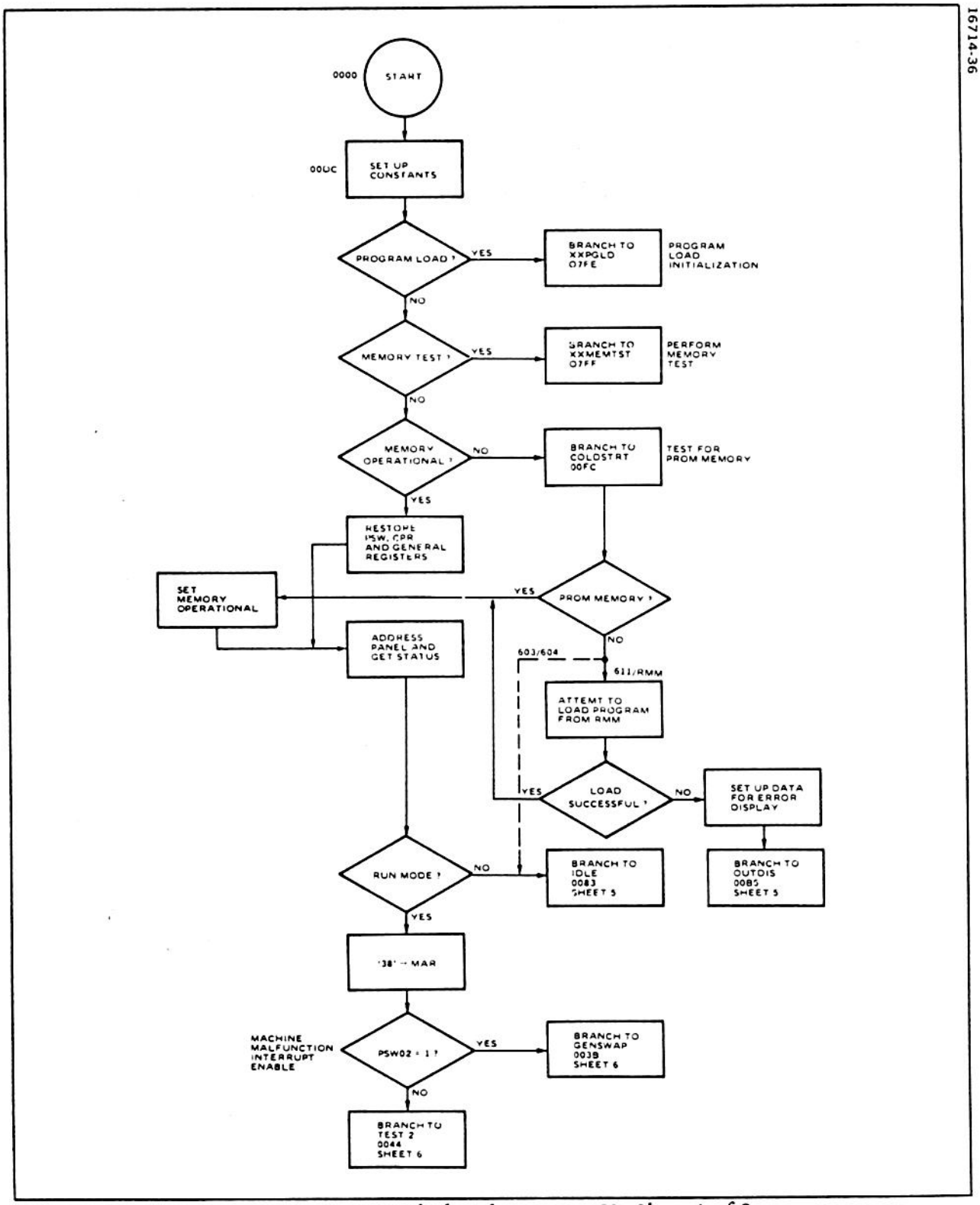
3) I/O Mux Bus Interrupt (TATN) (the Attention flag)

4) PMP Execute (CATN)

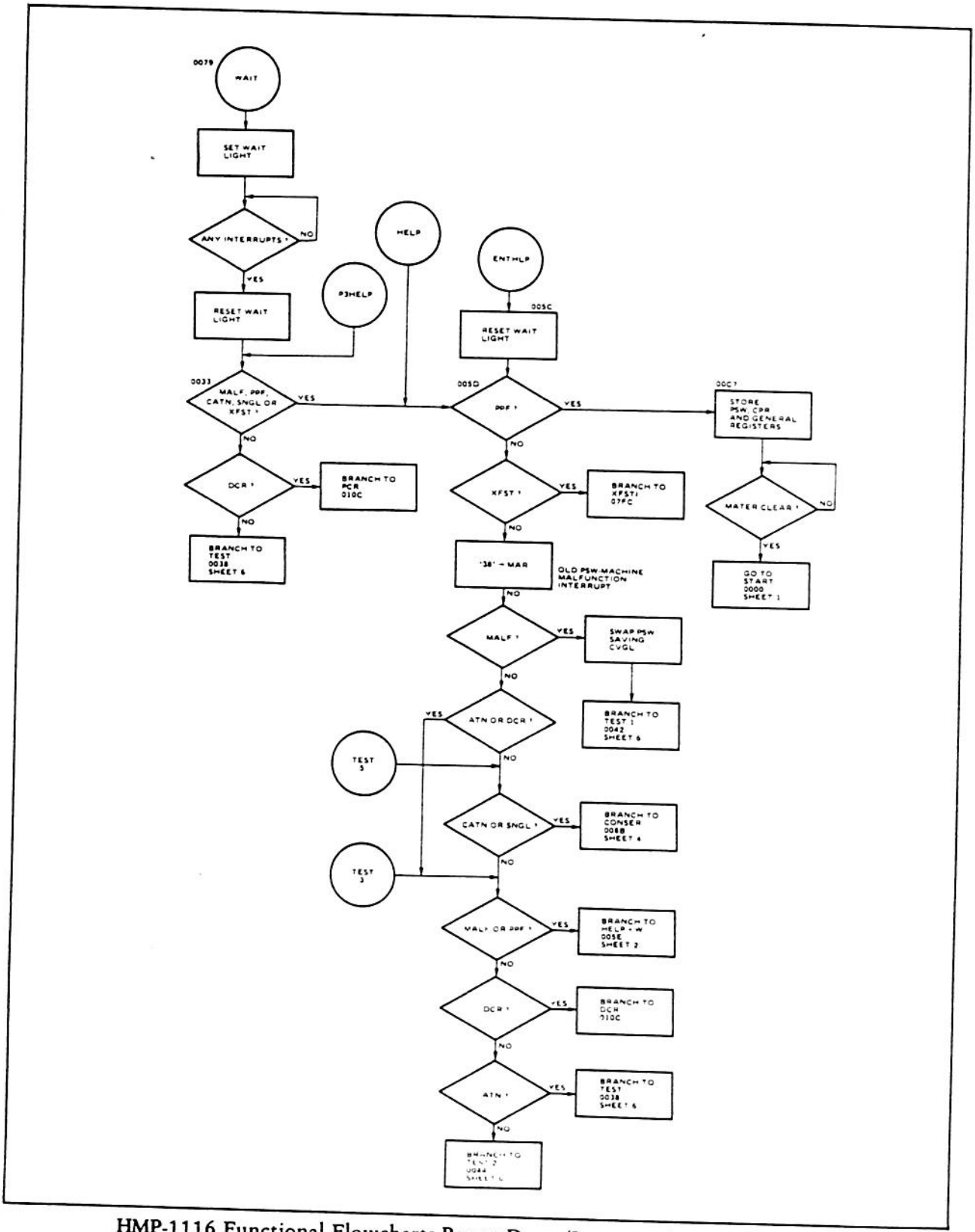
5) PMP Single Mode (SNGL)

6) Remote Program Load (XFST)

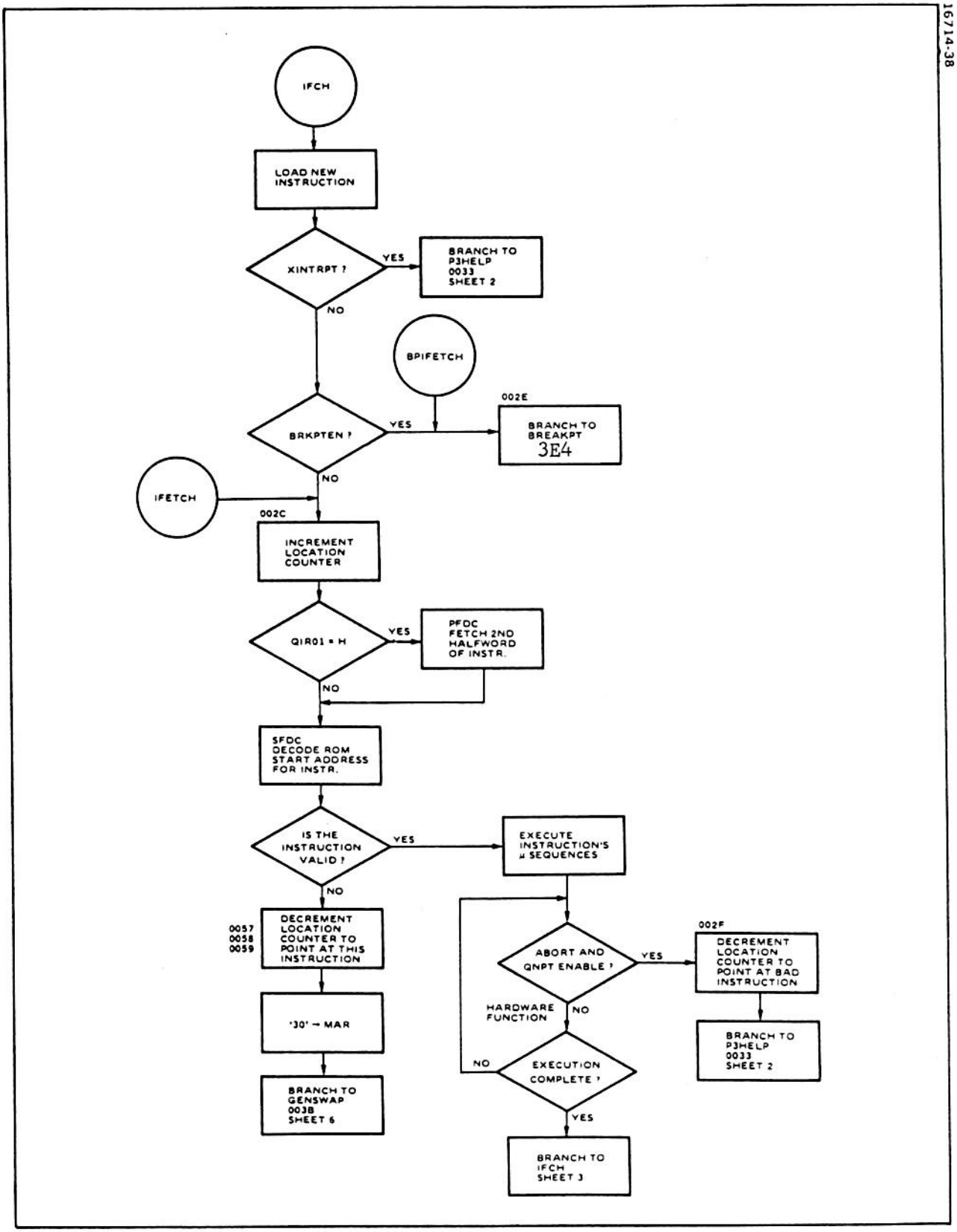
The microprogram determines which interrupt has occurred and takes the appropriate actions.



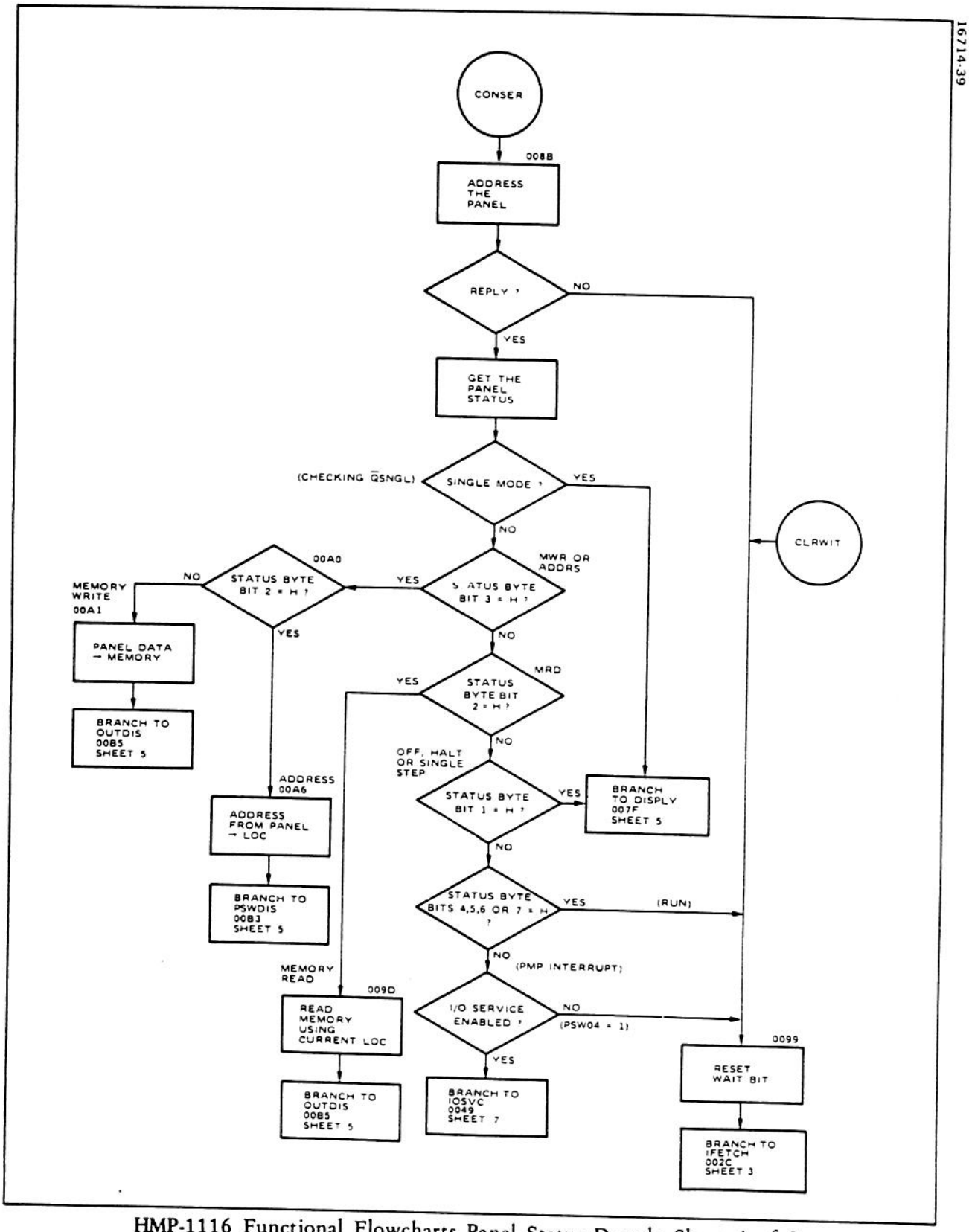
HMP-1116 Functional Flowchart Power Up Sheet 1 of 8



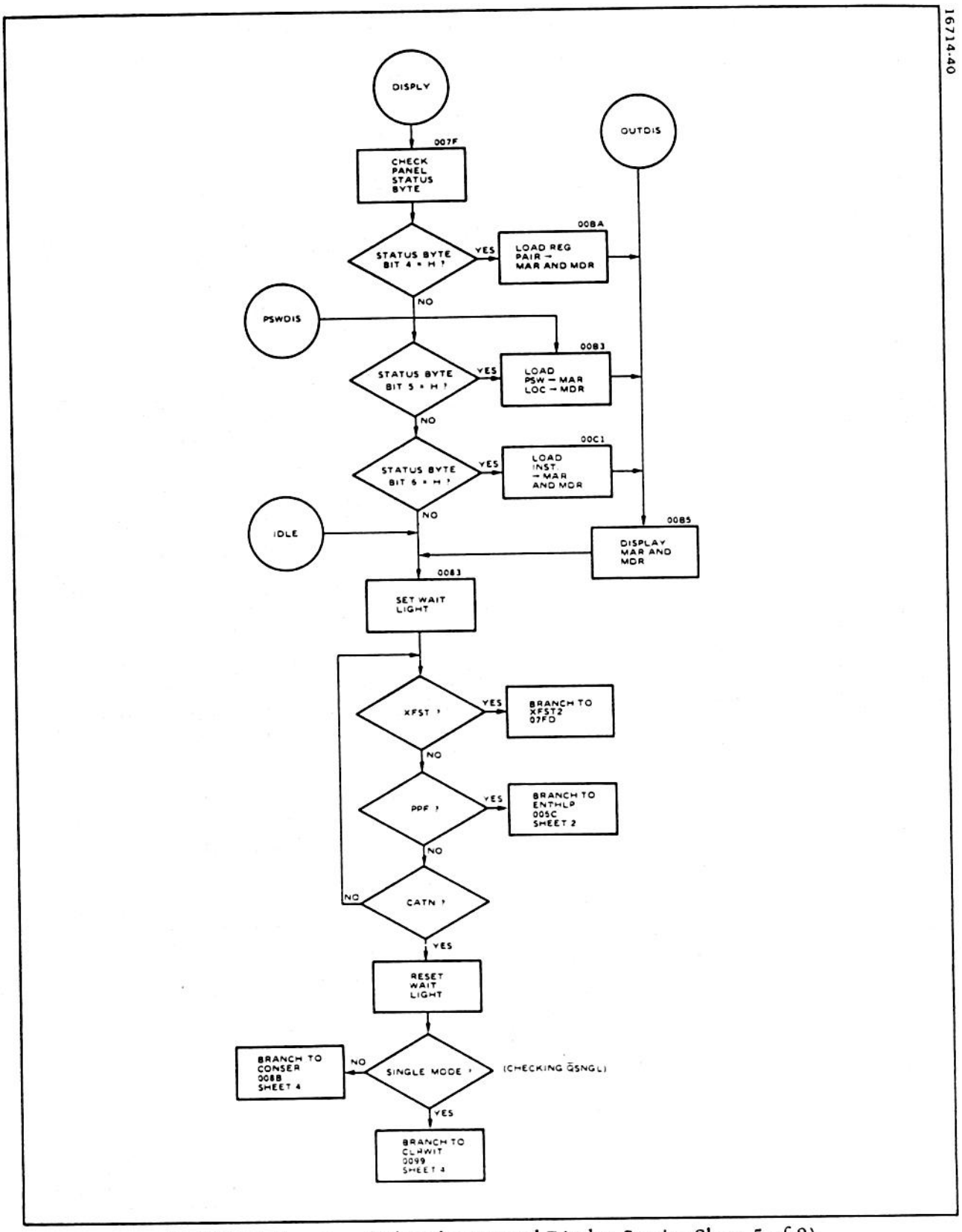
HMP-1116 Functional Flowcharts Power Down/Interrupt Service Sheet 2 of 8)



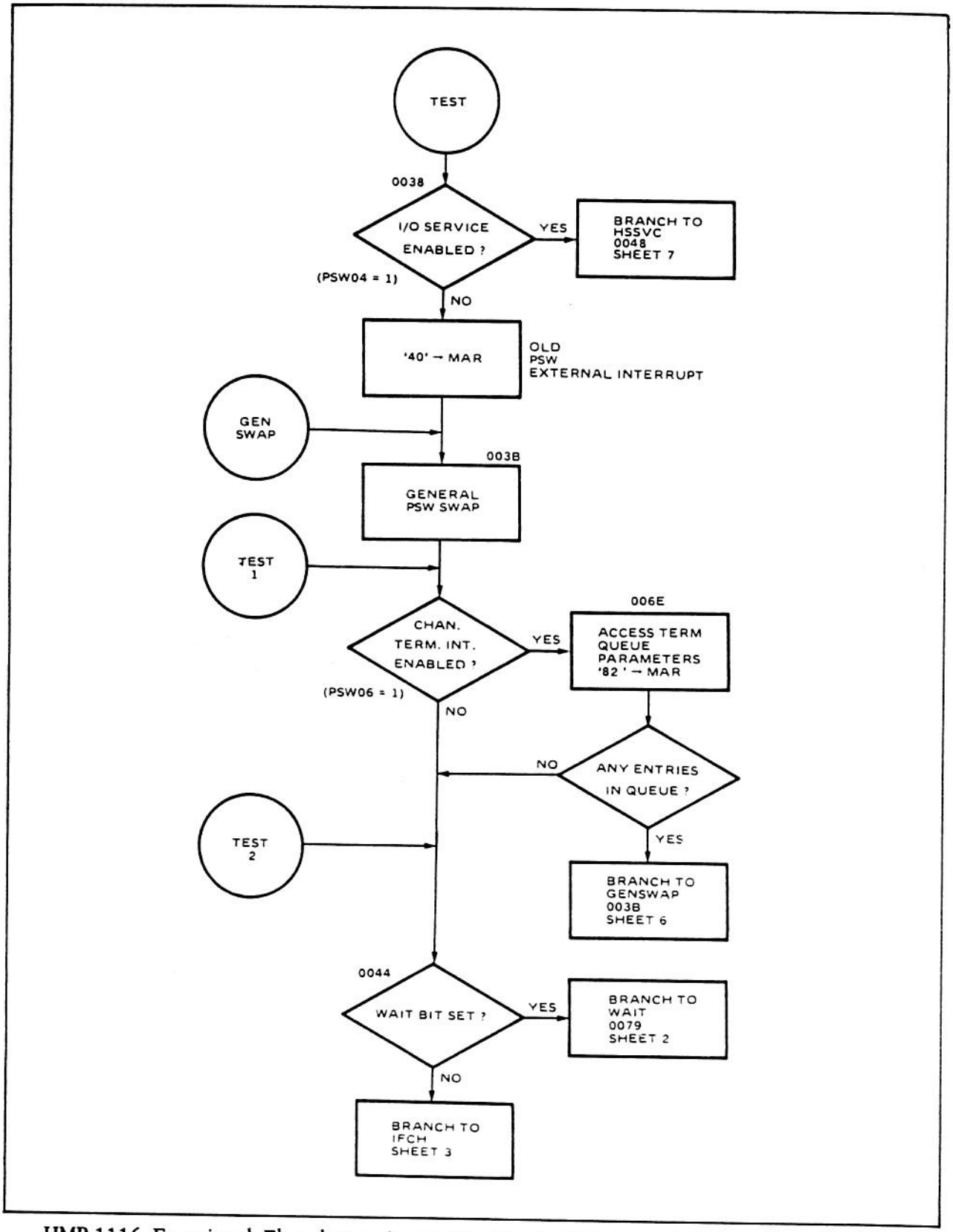
HMP-1116 Firmware Flowcharts Instruction Fetch and Execution Sheet 3 of 8



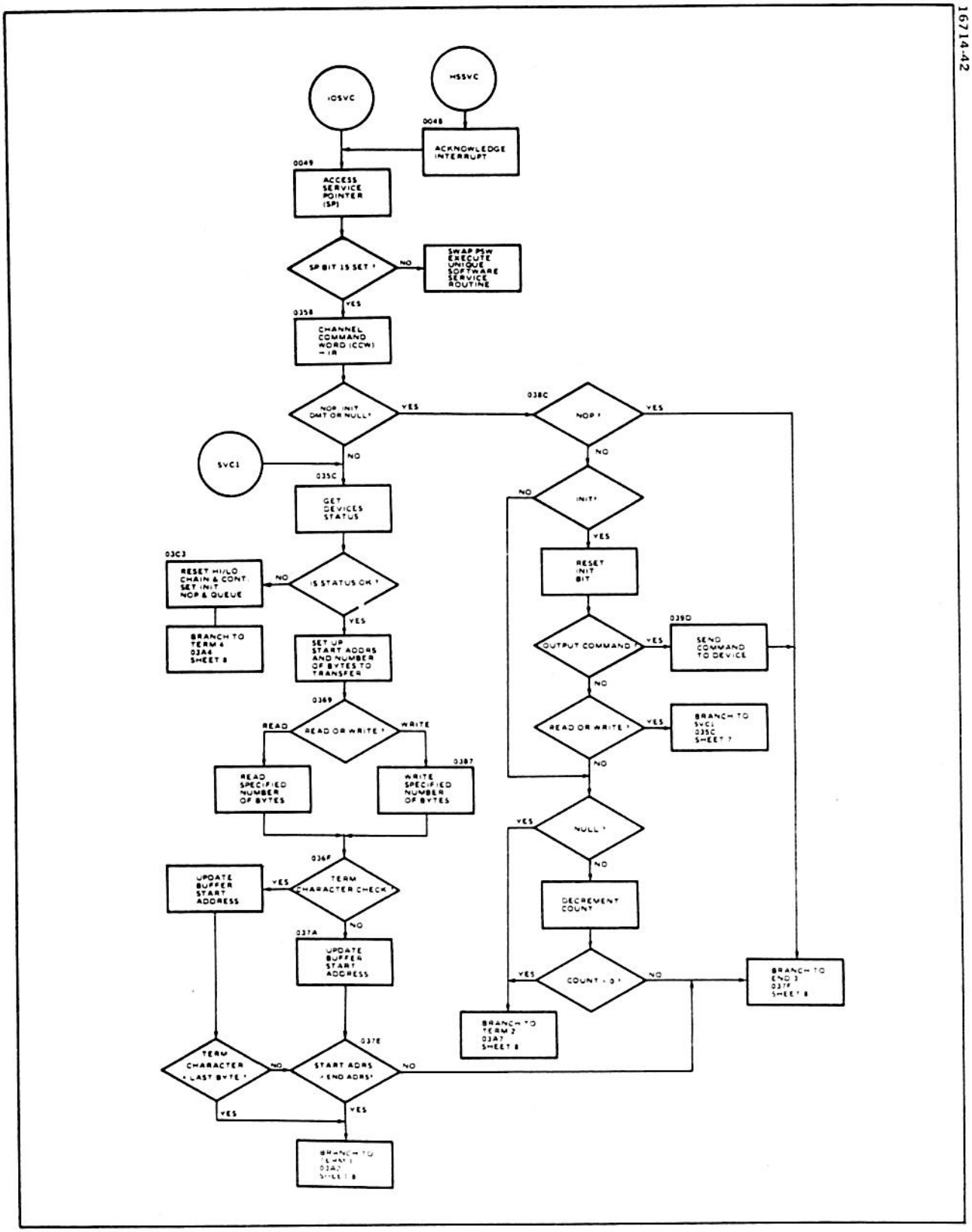
HMP-1116 Functional Flowcharts Panel Status Decode Sheet 4 of 8



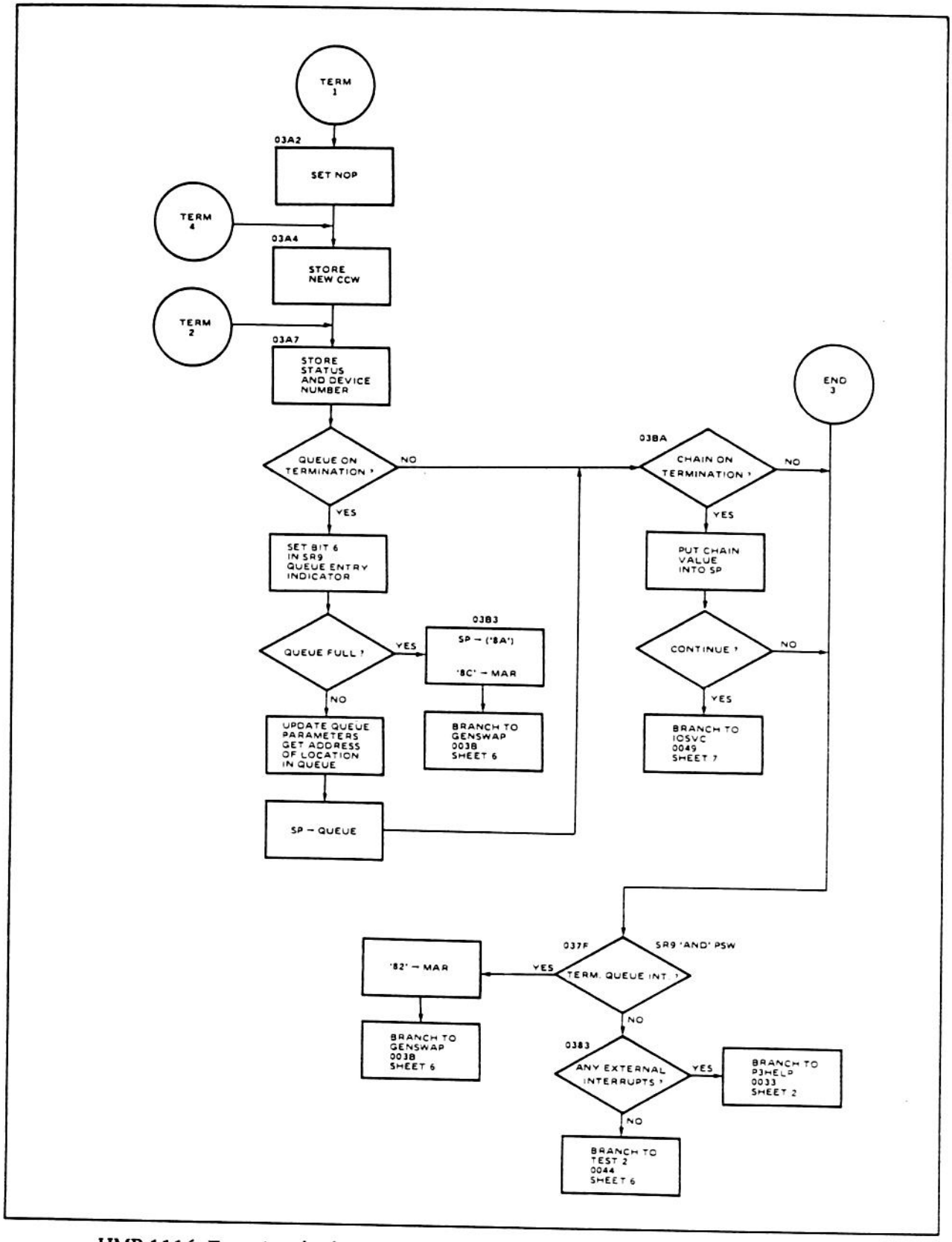
HMP-1116 Functional Flowcharts Panel Display Service Sheet 5 of 8)



HMP-1116 Functional Flowcharts I/O Service Check PSW Swap Routine Sheet 6 of 8



HMP-1116 Functional Flowcharts Automatic I/O Service Sheet 7 of 8



HMP-1116 Functional Flowcharts Automatic I/O Termination Sheet 8 of 8

9.0 I/O OPTIONS

9.1 Processor Maintenance Panel (PMP). Figure 9-1 is a simplified block diagram of the PMP. The PMP interfaces with the I/O Mux Bus. It is assigned device address '01', and is a byte device.

Data read from the PMP is the data designated by the Thumbwheel Switches (the DATA/ADDRESS Switches).

Data written to the PMP is loaded into DISPLAY 1 and DISPLAY 2.

The command to the PMP places it in the normal or incremental mode. The mode affects how data is read from and written to the PMP.

The status of the PMP is a function of the RUN and SINGLE Pushbuttons, and the position of the FUNCTION Switch. This status is used by the HMP-1116 to determine what action the operator at the PMP has specified. The HMP-1116 firmware samples the PMP status whenever the EXECUTE Pushbutton is pressed.

The PMP command and status formats are shown in Figure 9-2.

Unlike all other I/O devices, the PMP is not connected to the I/O Attention line. The PMP has its own interrupts; 1) XCATN - goes active when the EXECUTE Pushbutton is pressed and 2) XSNGL - is active when the PMP is set up for program execution in single step.

The operator has the capability of simulating an Attention type interrupt from the PMP by the following switch actions:

1) FUNCTION Switch to OFF/M WR

2) RUN on, SINGLE off

3) Press EXECUTE

In response to these actions, the firmware routine which services PMP interrupts will branch to the I/O Mux Bus interrupt service routine.

The firmware service for the PMP is in the Functional Flowcharts on pages 8-17 and 8-18. The detailed functional schematic of the PMP interface card is on page 26 of the Functional Schematics (Chapter 6).

The PMP reads and writes bytes in the order shown in Figure 9-3.

In the Normal mode, a read or write following addressing the PMP will begin with the First Byte.

In the Incremental mode addressing the PMP has no effect on what data is read or written next; each data transfer will be of consecutive bytes.

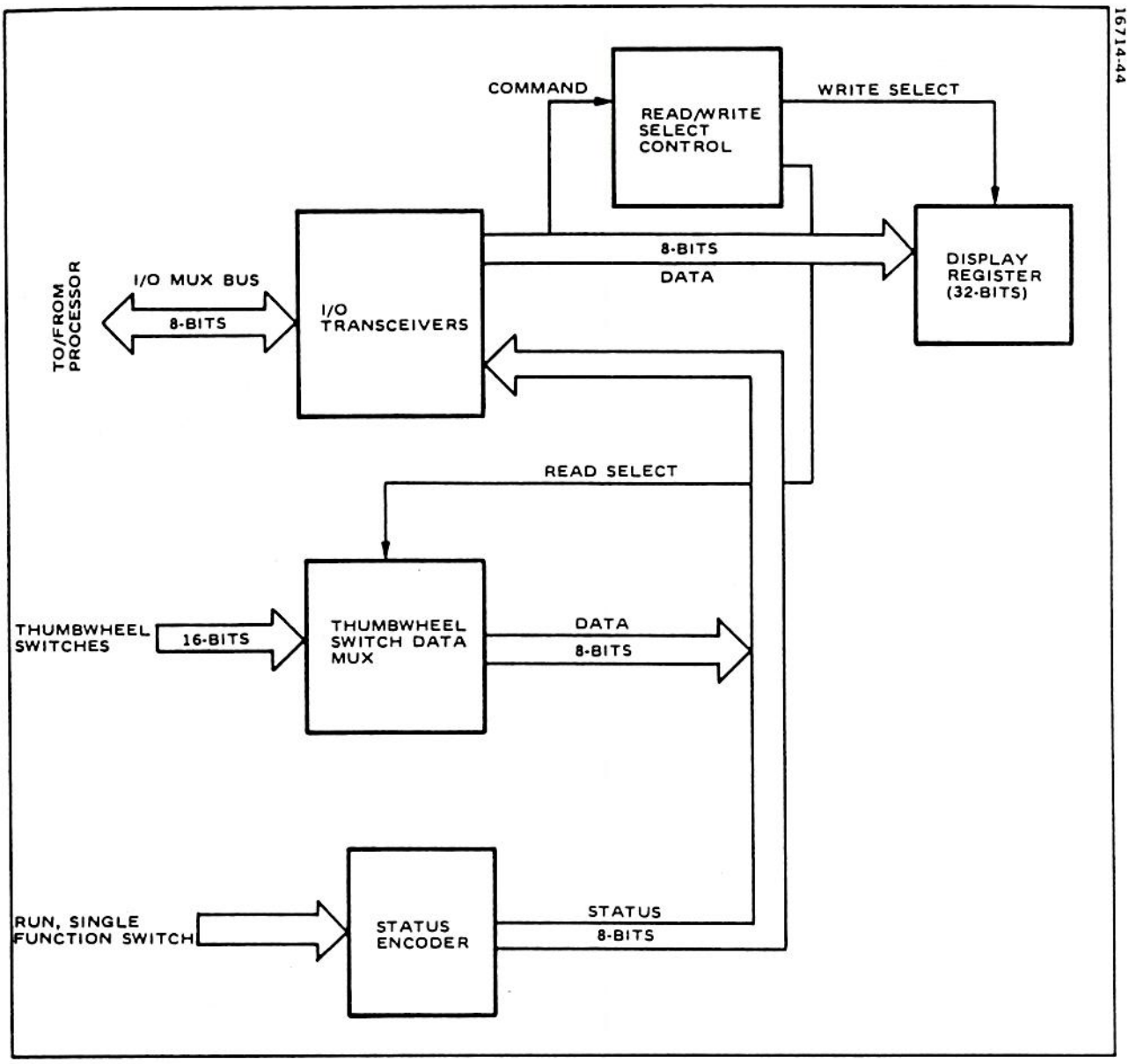


Figure 9-1. Processor Maintenance Panel Block Diagram

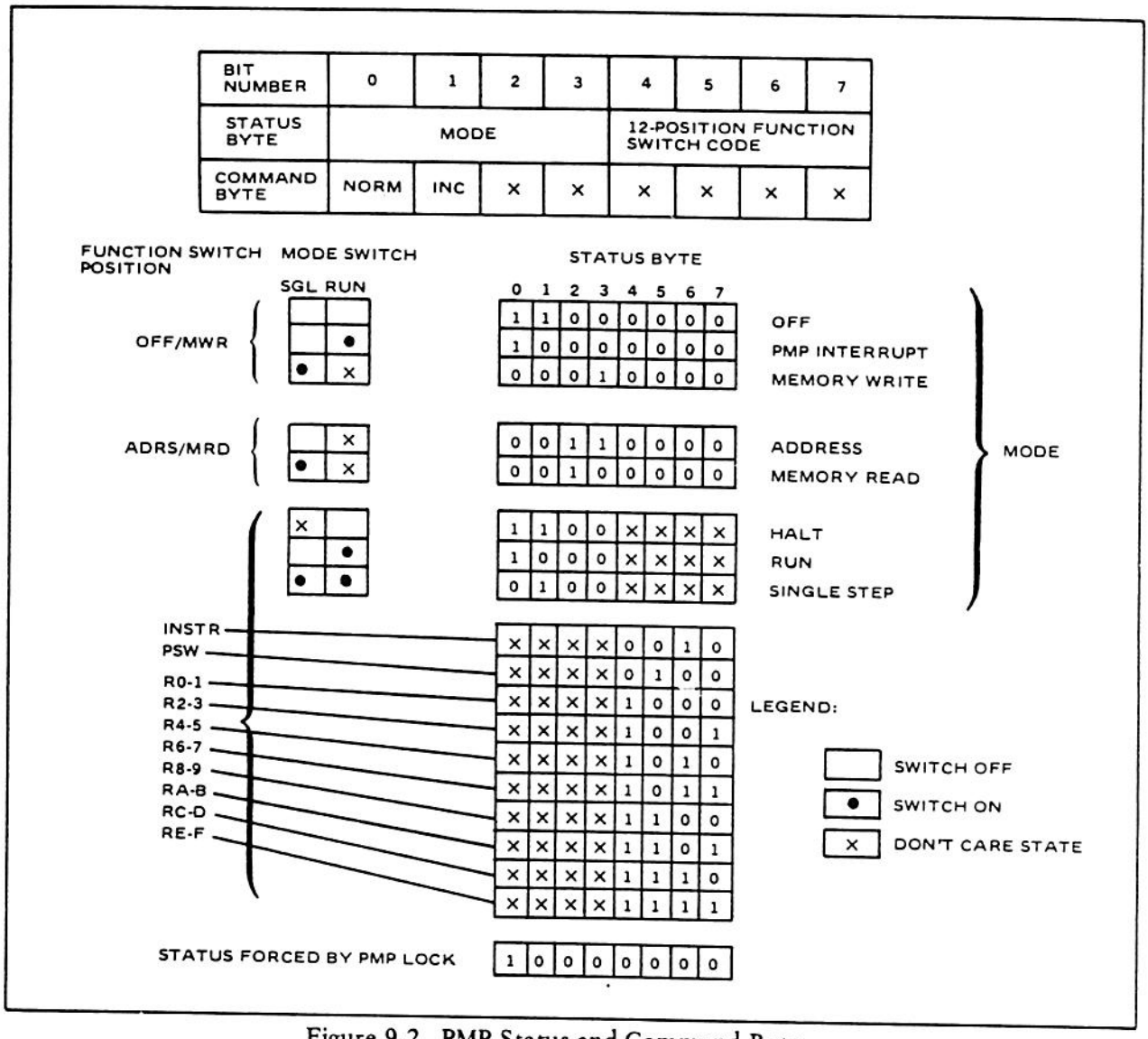


Figure 9-2. PMP Status and Command Byte

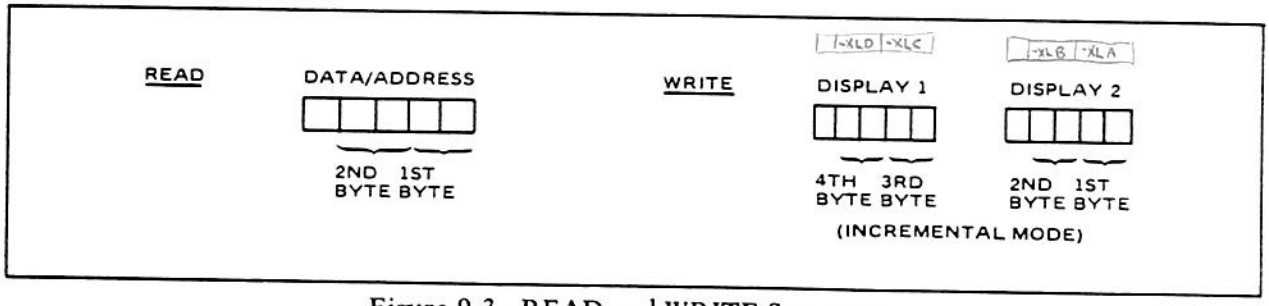


Figure 9-3. READ and WRITE Sequence

After Power Up or Master Clear the PMP is in the Normal mode.

ROTARY FUNCTION SWITCH OUTPUTS

POSITION	$\overline{S}$FC4	$\overline{S}$FC5	$\overline{S}$FC6	$\overline{S}$FC7
RE-F	0	0	0	0
RC-D	0	0	0	1
RA-B	0	0	1	0
R8-9	0	0	1	1
R6-7	0	1	0	0
R4-5	0	1	0	1
R2-3	0	1	1	0
R0-1	0	1	1	1
PSW	1	0	1	1
INSTR	1	1	0	1
ADRS/M RD	1	1	1	0
OFF/M WR	1	1	1	1

9.2 Program Controlled Clocks (PCC). Figure 9-4 is a simplified block diagram of the PCC. The PCC consists of two clocks; 1) the Real Time Counter (RTC) and 2) the Elapsed Time Counter (ETC). The functions of the RTC and ETC are summarized below:

I. Real Time Counter(RTC)

1. 32-bit up counter
2. Starting count can be programmed
3. Current count can be read
4. Counts with 25 microsecond resolution
5. Can generate interrupt on Overflow
6. Max time to Overflow 29.8 hours

II. Elapsed Time Counter(ETC)

1. 16-bit down counter

2. Starting count is programmed

3. Counts with 25 microsecond resolution

4. Can generate an interrupt on Timeout

5. Max time to Timeout 1.6 seconds

The command to the PCC tells it if the RTC is to be read or programmed, or if the ETC is to be programmed. The command also controls whether the ETC or the RTC can generate an interrupt when they have finished counting.

The status byte of the PCC tells the programmer whether an ETC Timeout or an RTC Overflow caused an interrupt.

The format of the command and status bytes are shown on page 9-7.

The sequence of software instructions required to Load and Arm the RTC, Read the RTC and Load and Arm the ETC are shown on page 9-8.

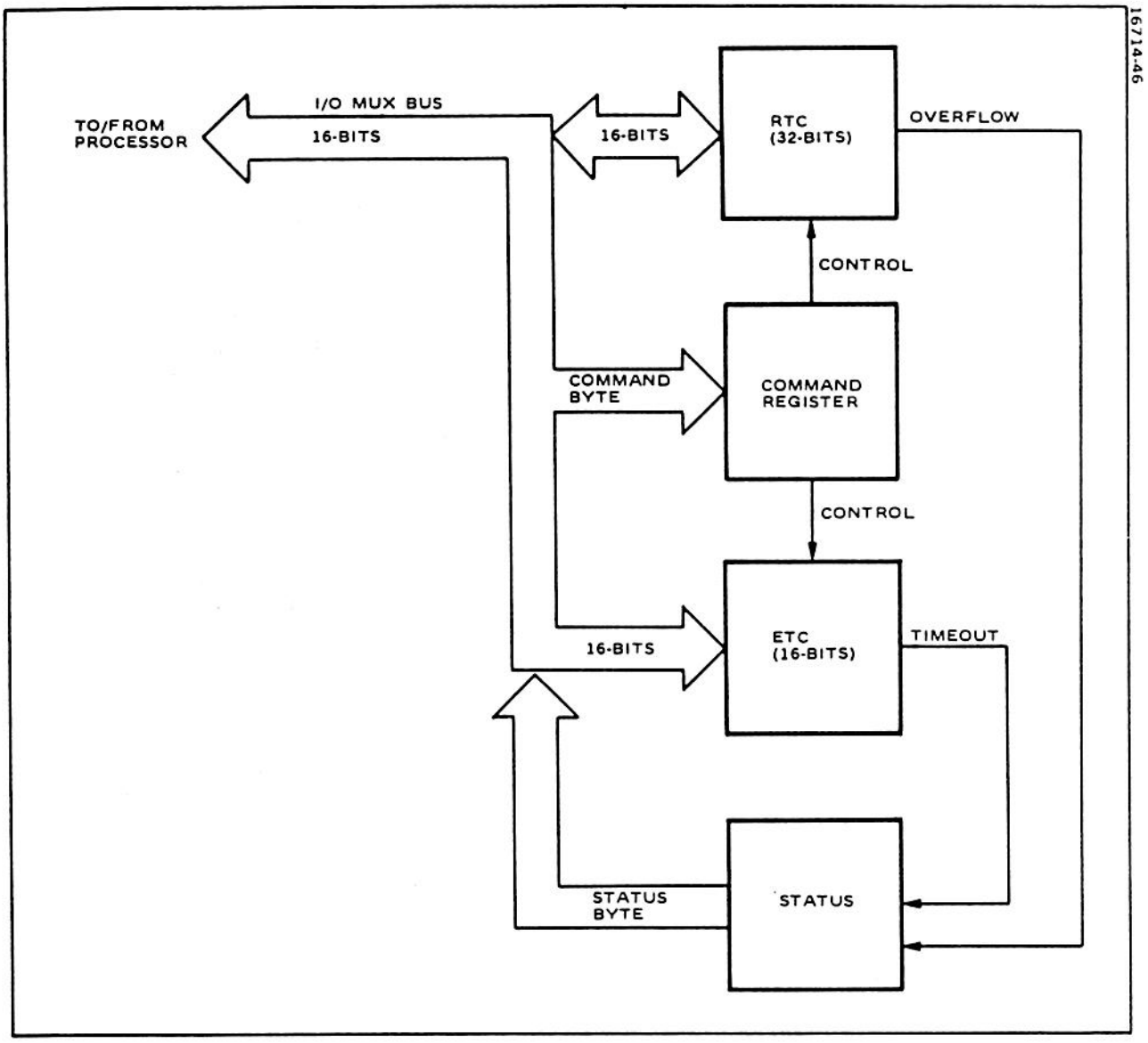


Figure 9-4. Program Controlled Clocks

16714-47

PCC WORD FORMATS

PCC DEVICE ADDRESS = '40'

PCC COMMAND FORMAT:

BITS 8	9	10	11	12	13	14	15
ARM RTC INT	DISARM RTC INT	LOAD RTC	READ RTC	ARM ETC INT	DISARM ETC INT	LOAD ETC	NOT USED

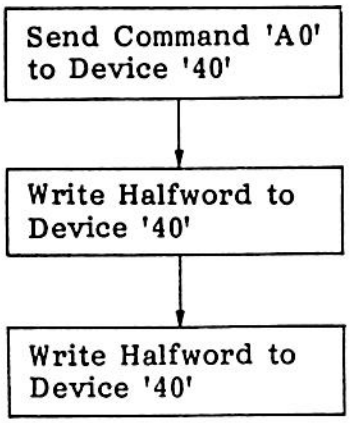
BIT	INTERPRETATION
8	ARM RTC INTERRUPTS
9	DISARM RTC INTERRUPTS
10	LOAD RTC (32 BITS)
11	READ RTC (32 BITS)
12	ARM ETC INTERRUPT
13	DISARM ETC INTERRUPT
14	LOAD ETC (16 BITS)
15	NOT USED

PCC STATUS BYTE:

BITS 8	9	10	11	12	13	14	15
X	X	X	X	X	X	ETC TIMEOUT	RTC OVERFLOW

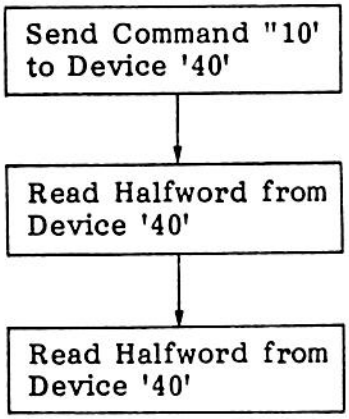
BIT	INTERPRETATION
8 THRU 13	NOT USED
14	ETC HAS COUNTED DOWN TO ZERO
15	RTC HAS COUNTED PAST 'FFFF FFFF'

1) LOAD AND ARM THE RTC



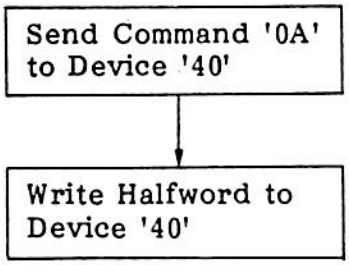
OCR	Command tells the PCC that RTC is to be loaded and enables RTC interrupts
WHR	16-MSBs for the RTC
WHR	16-LSBs for the RTC

2) READ THE RTC

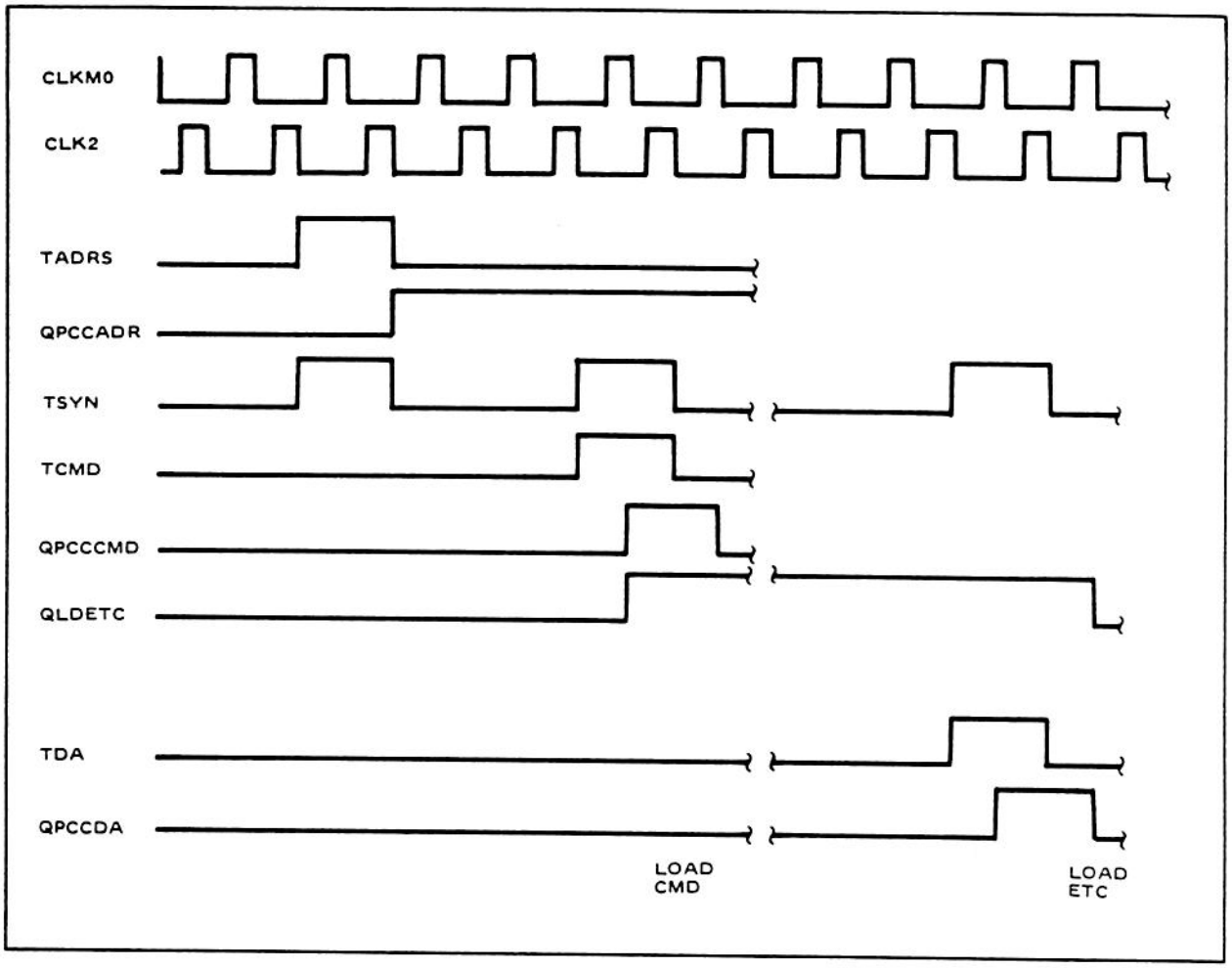


OCR	Command tells PCC that RTC is to be read
RHR	16-MSBs of RTC
RHR	16-LSBs of RTC

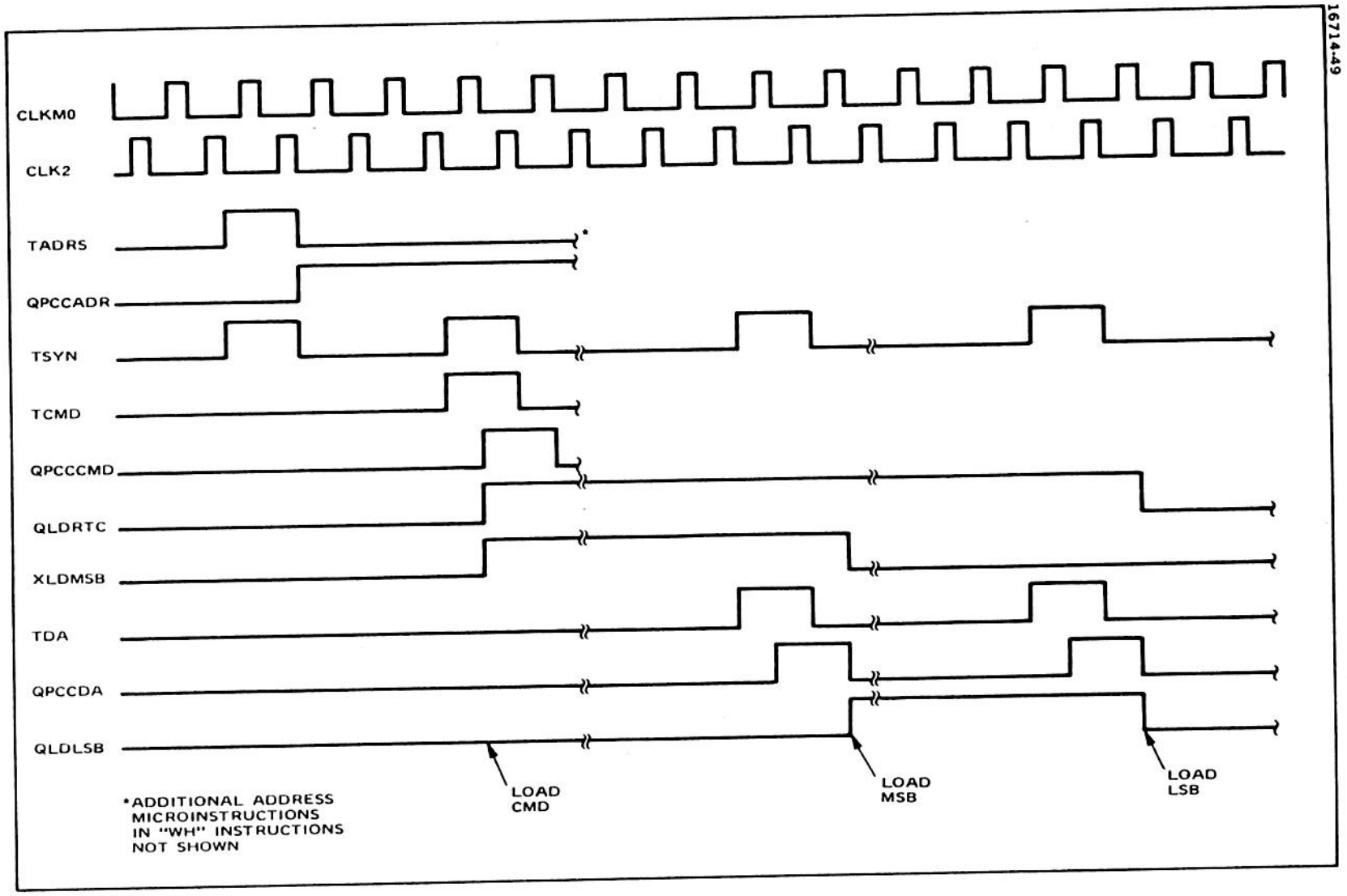
3) LOAD AND ARM THE ETC



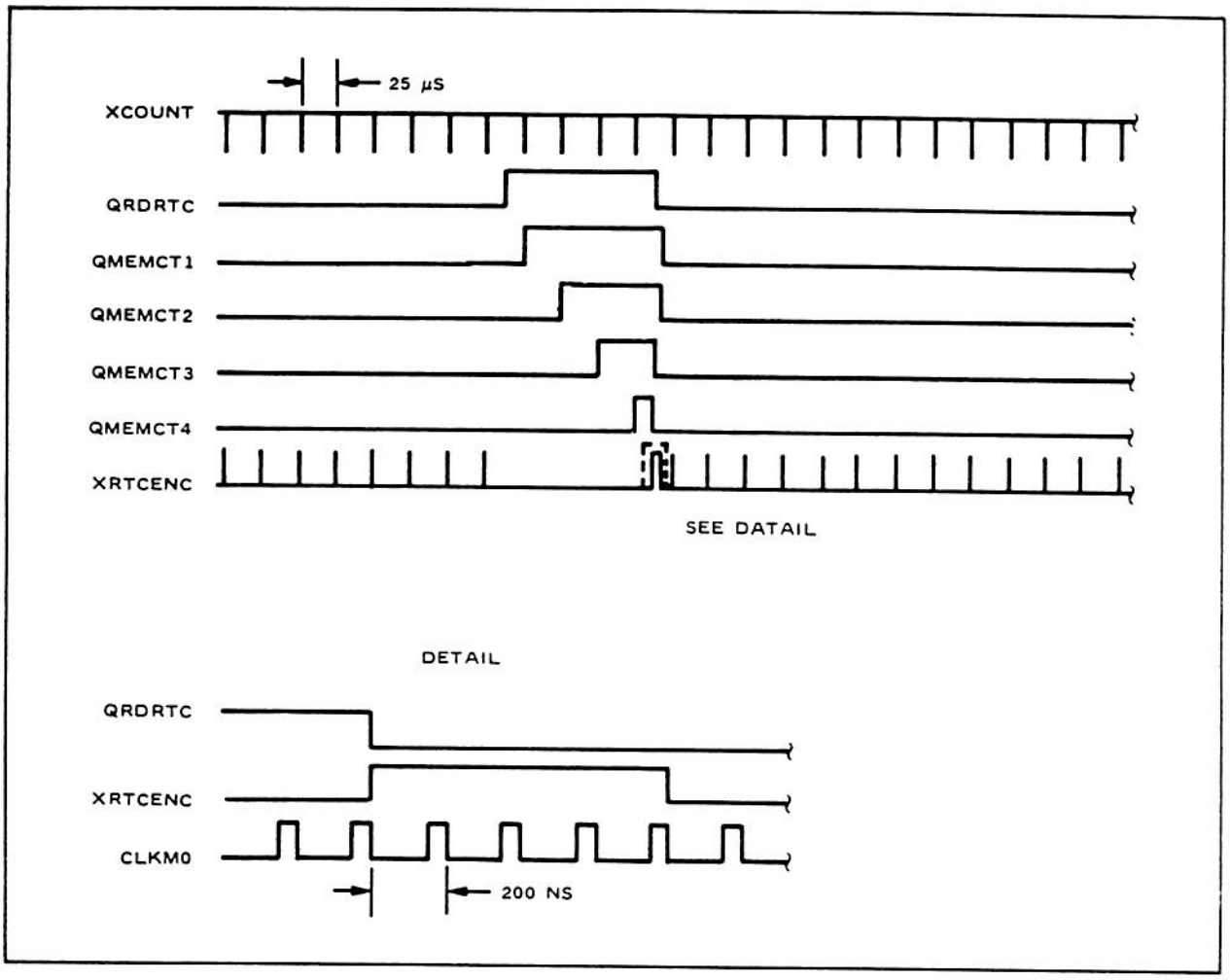
OCR	Command tells the PCC that ETC is to be loaded and enables ETC interrupts
WHR	16 bits for ETC



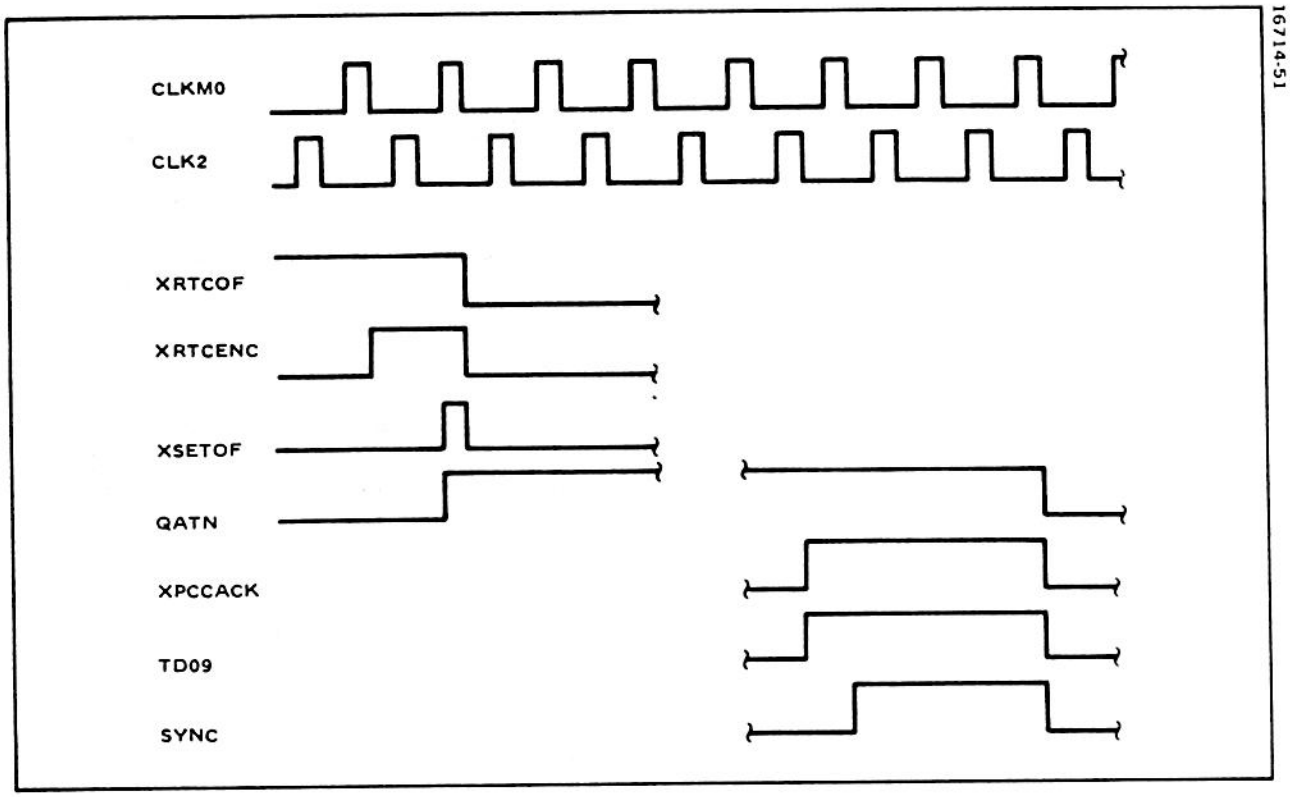
Loading the ETC



Loading the RTC



RTC Count Saver



RTC Overflow Interrupt and Acknowledge

9.3 Memory Protect Controller (MPC). Figure 9-5 is a simplified block diagram of the MPC. The function of the MPC is to provide the capability of protecting blocks of the HMP-1116 RAM Memory from being written into by the Processor or DMA.

The MPC divides memory into 1,024 blocks of 128 halfwords. The Memory Protect Map contains one bit for every 128 halfwords in memory. If a bit is set, its corresponding 128 halfwords in memory are protected.

The word formats of the MPC are shown on page 9-15.

The sequence for programming the MPC is:

Send Command '16' To Device 'F9'	OCR Disable Protect, Load
Send 128 Bytes To Device 'F9'	Can use 128 WDs, 64 WHs or 1 WB instructions, This programs the Memory Protect Map
Send Command '22' To Device 'F9'	OCR Enable Protect

128 BYTES WRITTEN TO MPC

byte 0	0	1	2	3	4	5	6	7
byte 1	8	9	A	B	C	D	E	F
.								
.								
.								
byte 126	3F0	3F1	3F2	3F3	3F4	3F5	3F6	3F7
byte 127	3F8	3F9	3FA	3FB	3FC	3FD	3FE	3FF

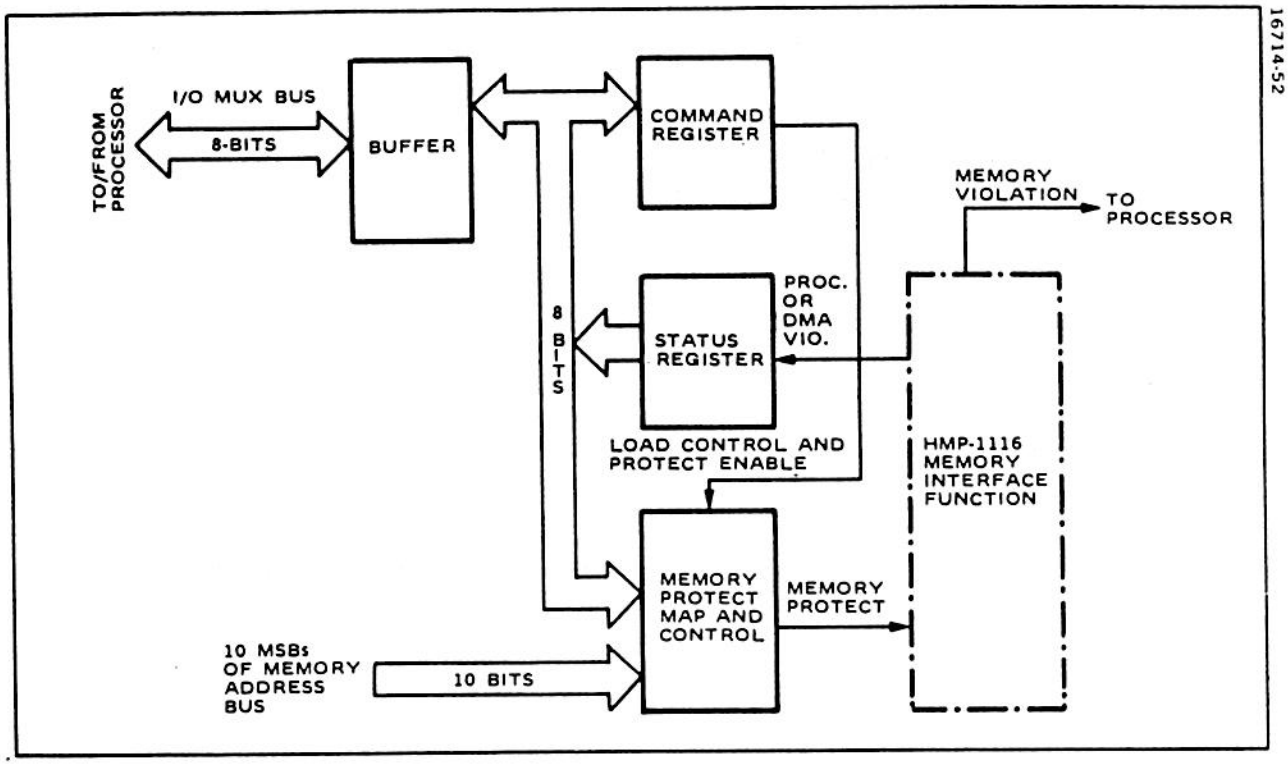


Figure 9-5. Memory Protect Controller Block Diagram

BIT SET	PROTECTS MEMORY LOCATIONS
0	00000 - 000FE
1	00100 - 001FE
2	00200 - 002FE
.	.
.	.
.	.
D	00D00 - 00DFE
.	.
.	.
.	.
3F4	3F400 - 3F4FE
.	.
.	.
.	.
3FE	3FE00 - 3FEFE
3FF	3FF00 - 3FFFE

Once the MPC is loaded and enabled it will monitor the 10 MSBs of the Memory Address bus. These 10 bits are used to select the appropriate protect bit from the Memory Map. If the protect bit for the addressed memory location is set, the Memory Protect control signals are sent to the HMP-1116 Memory Interface. If the Memory Interface sees these protect signals and a write is being performed it 1) disables the write operation, 2) sends the Memory Violation signal to the Processor (which causes a Machine Malfunction Interrupt) and 3) sends Processor or DMA Violation signals back to the Status Register in the MPC.

MPC WORD FORMATS

MPC device address = F9

MPC command format:

BITS	8	9	10	11	12	13	14	15
	X	X	EN	DIS	X	LOAD	DMA CHECK	X

Bit	
8,9,12,15	Not Used
10	Enable Memory Protection
11	Disable Memory Protection
13	Load Memory Protect Map
14	When Reset (0) Disables all DMA Writes Test Function only.

MPC status byte:

BITS	8	9	10	11	12	13	14	15
	X	X	X	X	X	X	DMA	PROC

Bit	Interpretation
8 thru 13	Not used
14	DMA Memory Violation
15	Processor Memory Violation

9.4 Read Mostly Memory (RMM). Figure 9-6 is a simplified block diagram of the RMM. The function of the RMM is to provide a 1,024 byte nonvolatile read/write memory. The RMM will retain data for a minimum of 10 years without power.

One application of the RMM is to contain a software bootstrap loader. The microprogram reads the loader program from the RMM and places it into RAM memory. The loader in the RMM has the advantages of being permanently stored in the computer (like firmware) and being software reprogrammable.

RMM WORD FORMATS

RMM device address = '41' or '42'

RMM command format:

8	9	10	11	12	13	14	15
OP-CODE		NOT USED				1st MSB	2nd MSB

8	9	
0	0	Read Block
0	1	Read Byte
1	0	Write Block
1	1	Invalid Command

Bits 14 and 15 are the two MSBs of the address for a Byte Read. Bits 14 and 15 are not used for Read Block or Write Block

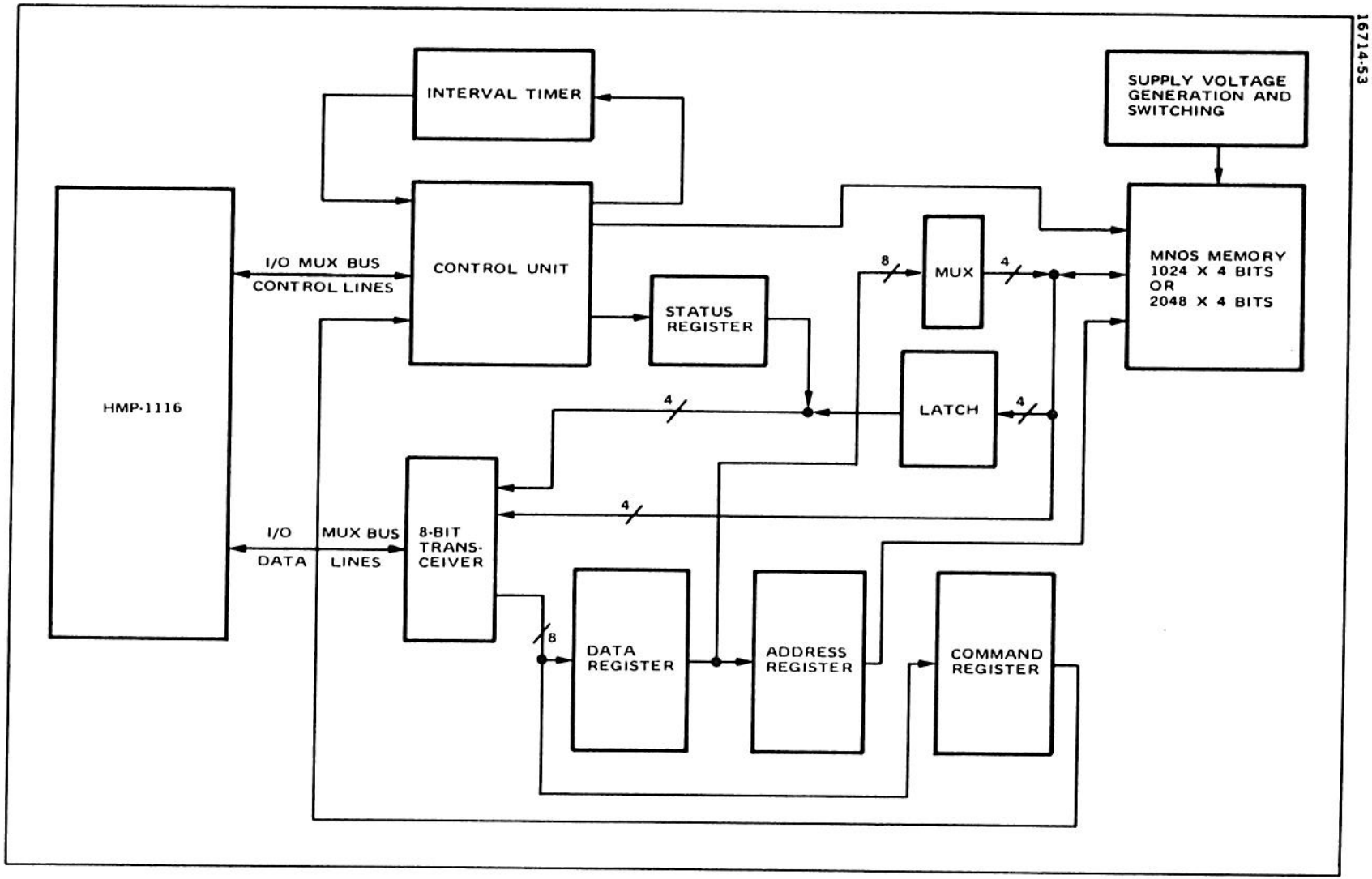


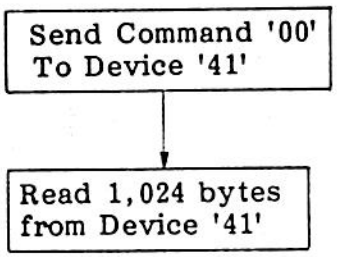
Figure 9-6. Read Mostly Memory Block Diagram

RMM status byte:

8	9	10	11	12	13	14	15
X	X	X	X	BUSY	X	X	X

X = Don't care; BUSY = If set (1) indicates RMM is busy.

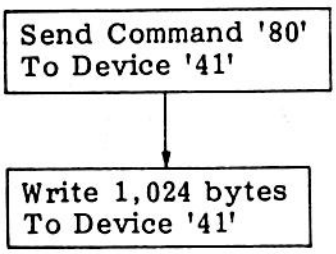
READ BLOCK OPERATION



OCR — Send read block command

RB — Read data from RMM

WRITE BLOCK OPERATION



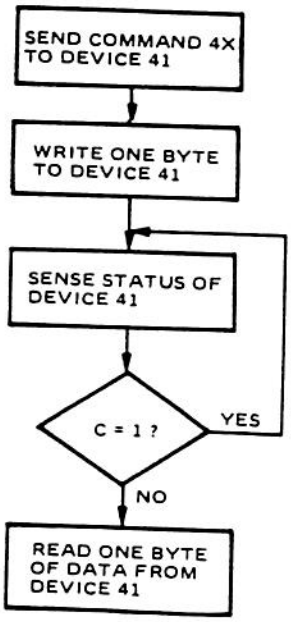
OCR — Send write block command

WB — Write Data to RMM

NOTE: Before Reading, Writing or Commanding the RMM its status should be sensed to make sure it is not Busy. If the RMM is Busy and an attempt is made to do one of these operations the results will probably be invalid.

WARNING: The EAROM used in the RMM has a limited number of write cycles it can perform in its lifetime (100,000)

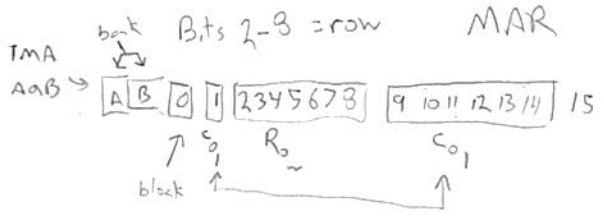
The Read Byte operation requires the programmer to first send a 10 bit address to select the byte to be read. The two MSBs of the 10 bit address are bits 14 and 15 of the Command Byte. The following sequence must be performed to read one byte:



16714-54

OCR	Send Read Byte COMMAND. The 2 LSBs of X will be the MSBs of the address of the byte to be read; X will be 0 , 1 , 2 or 3.
WDR	Byte sent is the 8 LSBs of the byte address
*SSR	Sense the status of the RMM and ...
*BTBS	If the RMM is busy, continue to sense its status. If the RMM is not busy a byte of data is available to be read.
*RDR	Read data byte from RMM

*The last three steps of this program can be repeated in order to read consecutive bytes from the RAM. The RMM will automatically update the byte address after every data transfer.



10. MEMORY

10.1 Introduction. The HMP-1116 MOSRAM Memory is used to store software programs and data. The memory consists of up to 4 cards; each card containing 32K 17-bit words (16-bit halfword plus parity). The memory capacity can be from 32K to 128K halfwords, expandable in 32K increments.

The RAM Memory chips are 16,384 X 1-bit RAMS. The 16,384 bits are arranged in a 128 X 128 bit matrix. A 14-bit address is required to select one bit. Bits 2-8 of the address are called the Row address, Bits 1, 9-14 are called the Column address. The RAM memory bits must be refreshed every 2 ms. Refresh is performed on one Row of bits at a time (128 bits). A Refresh operation only requires a 7-bit Row address.

A group of 17 RAMS chips is used to provide 16K halfwords plus parity. A memory card containing 32K halfwords contains two groups of 17 RAM memory chips.

Figure 10-1 is a block diagram of the HMP-1116 memory and its control. The blocks labeled MIF are functions on the Memory Interface card. The blocks labeled RAM are contained on a RAM memory card (1641763). The Memory Block Diagram only shows the MIF card and one RAM card. To show the complete 128K configuration all functions labeled RAM would be repeated 4 times. All RAM cards are identical; the 32K group of addresses a RAM card responds to is a function of which slot it is in.

The MIF card controls the three functions which access the RAM memory. These functions are listed below from highest to lowest priority:

1) Refresh

2) DMA (Direct Memory Access)

3) Processor Memory Access

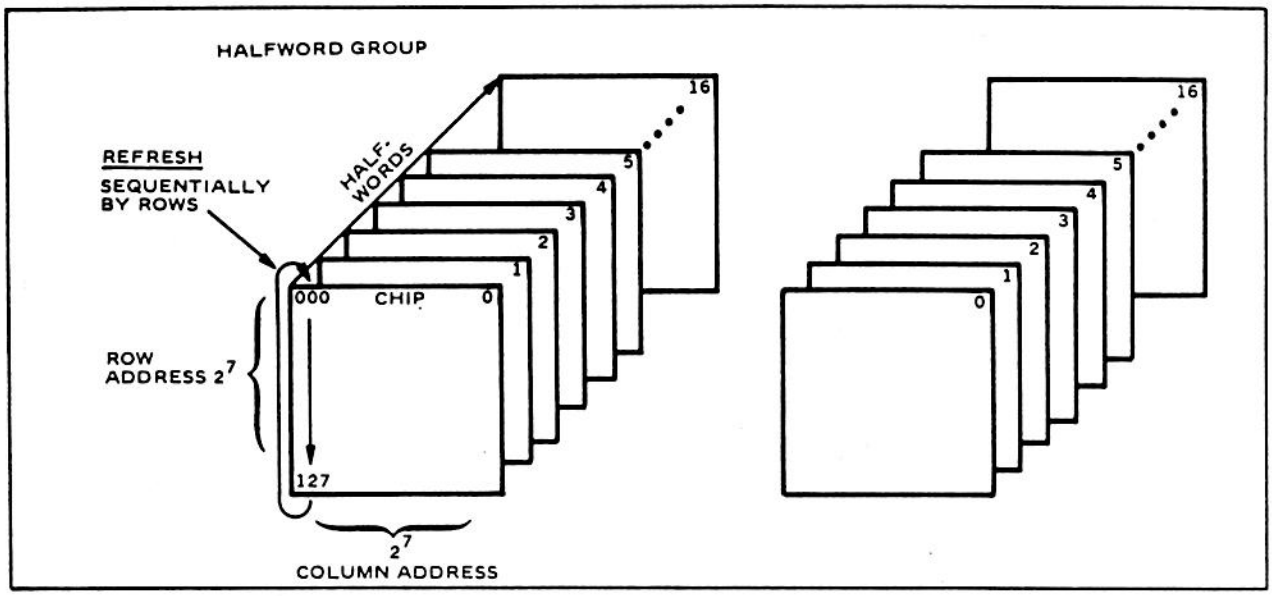
Refresh, DMA, Proc? Phase 0 Control
1.4 μs
Phase 1 Address
Phase 2 Read/Write

10.2 RAM Memory. The interface of the RAM memory card consists of the Memory Enable control signal, Read/Write control signals, the Memory Address Bus and the Memory Data Bus.

The Memory Data Bus ($\overline{\text{TMD}}$00-16) is a 17-bit bidirectional bus. The Memory Data Buffer allows data to flow only into or out of memory for any one operation. The Memory Write Control controls which way data is allowed to flow; Write Enable (QWRTENB) allows data into Memory and Read Enable (QRDENB) allows data to flow out of Memory.

DMA Devices and the Processor can read out of or write into Memory. When a Refresh operation is performed, this control signal selects read.

A 17-bit memory address is required to select a memory location for reading or writing. When a DMA Device or the Processor access memory they



RAM Memory Card (2 HW Groups)

place a 17-bit address on the Memory Address Bus ($\overline{T}$MAA, $\overline{T}$MAB, $\overline{T}$MA00-14). The address specifies one of 128K memory locations. The 3 MSBs ($\overline{T}$MAA, $\overline{T}$MAB, $\overline{T}$MA00) are decoded by the Address Decode Logic. These 3 bits specify a 16K block of memory (17 memory chips). The Address Decode logic on the RAM card containing the selected 16K block of memory data generates the Card Selected signal. This signal allows the memory chips on the selected card to be enabled. The 12 LSBs of the memory address ($\overline{T}$MA02-14) pass through the Memory Address Mux and become $\overline{X}$ADR02-14. $\overline{X}$ADRO2-14 along with $\overline{T}$MA01 is the 14 bit address sent to the memory chips to select one of 16K words. The row address is enabled to the RAM first. One hundred fifty nanoseconds later, the column address is enabled by QCASENB.

Memory Enable (QMENB) comes from the Memory Enable Logic. QMENB will be generated for any memory operation. QMENB will enable the Memory on a RAM card only if the card is selected by the MSBs of the address or if the Memory is being refreshed (XREF).

10.3 Refresh. The RAM memory is refreshed approximately every 2 ms. One hundred twenty-eight cycles are required to completely refresh memory.

The Refresh Interval Counter generates Refresh Time (XJREFTM) every 14.8 microseconds. XJREFTM indicates that it is time to refresh another 1/128 of memory.

The Refresh Request Logic receives XJREFTM and then generates Refresh Request (QREFREQ) to the Memory Enable Logic. This causes the Memory Enable Logic to generate QMENB. The Refresh Request Logic also generates Refresh Memory (XREF). XREF overrides the Address Decode Logic on all RAM cards in order to enable all memory chips to be refreshed. XREF also is sent to the Memory Address Mux and selects QREFADB, 0-5 as the memory address. QREFADB, 0-5 specifies which Row is to be refreshed.

QREFADB, 0-5 comes from the Refresh Address Counter. After the refresh operation is completed the Refresh Request Logic generates Increment Address (XREFCNT) which adds one to QREFACB, 0-5 so that it points at the next Row in memory to be refreshed.

10.4 DMA. When the DMA Control Logic receives a DMA Request ($\overline{T}$DMAREQ) from a DMA Device it responds by sending a DMA Acknowledge ($\overline{T}$EN) back to the device and sets DMA Request (QDMAREQ) and DMA Memory Cycle (QMEMDMA). QDMAREQ disables Processor memory access and QMEMDMA causes the Memory Enable Logic to generate QMENB.

When a DMA Device receives $\overline{T}$EN it takes control of the Memory Address and Memory Data Busses, and it sets DMA Write Request (TDMAWRT) if it wishes to write to memory. The Memory Write Control sets QWRITE, QWRTENB and QRDENB to the appropriate states based on $\overline{T}$DMAWRT.

When the DMA memory cycle is completed, the DMA Control Logic sends DMA Strobe ($\overline{T}$DMAST) to the DMA Device.

10.5 Processor Memory Access. The MC field (QRD20-22) in a CTL microinstruction generates a Processor memory request. This request is detected by the Processor Request Decoder, which decodes whether this request is read ($\overline{X}$MR) or a write ($\overline{X}$MW). Both requests are sent to the Memory Enable Logic and the write request is sent to the Memory Write Control. The Memory Enable Logic generates QMENB and the Memory Write Control sets QWRITE, QWRTENB and QRDENB to the appropriate states.

The Memory Enable Logic sends Memory Request (XMEMREQ) and Processor Memory Request (XMEMPRO) to the Processor/Memory Interface Control. The Processor/Memory Interface Control generates 5 signals for the Processor:

1) XMEMPRO causes Memory Access Granted (QMAG) to be set.

2) XMEMREQ and QWRITE are used to generate Parity Control (XPMD16) to the Processor's Parity Generator/Checker. XPMD16 low means generate parity.

3) XMEMPRO and QWRITE are used to generate XEPMDK. XEPMDK low causes the contents of the MDR to be placed on the Memory Data Bus (write).

4) QDMAREQ when set generates $\overline{X}$EPMAC1 which disables the MAR and MDR from being placed on the Memory Address and Memory Data Busses, respectively.

5) QDMAREQ when reset causes XEPMAJ which enables the contents of the MAR to be placed on the Memory Address Bus.

10.6 Memory Protect Option. The Memory Write Control receives $\overline{X}$PROCWP and XDMAWP from the MPC (Memory Protect Controller) Option. $\overline{X}$DMAWP occurs when a DMA Device tries to write into protected memory. Either of these conditions result in the write operation being disabled and Memory Violation (XMEMVIO) being sent to the Processor. XMEMVIO causes a Machine Malfunction Interrupt.

10.7 Memory Timing. A complete memory cycle takes 600 ns (3 HMP-1116 clock cycles). The memory logic keeps track of where in the 600 ns cycle it is by breaking the 600 ns into three phases; Phase 0 (first 200 ns), Phase 1 (second 200 ns) and Phase 2 (last 200 ns). Figure 10-2 shows the timing of the Memory Phase Counter. The hardware associated with the Memory Phase Counter is shown in the Functional Schematics (Chapter 6) on page 36.

Figure 10-3 shows the timing of Refresh, DMA and Processor memory cycles. It is assumed that the Refresh, DMA and Processor requests all occur simultaneously in the beginnning of the second clock cycle. Therefore, this diagram also shows how the contention logic works.

Figure 10-4 shows in detail the Read and Write cycles for Processor and DMA.

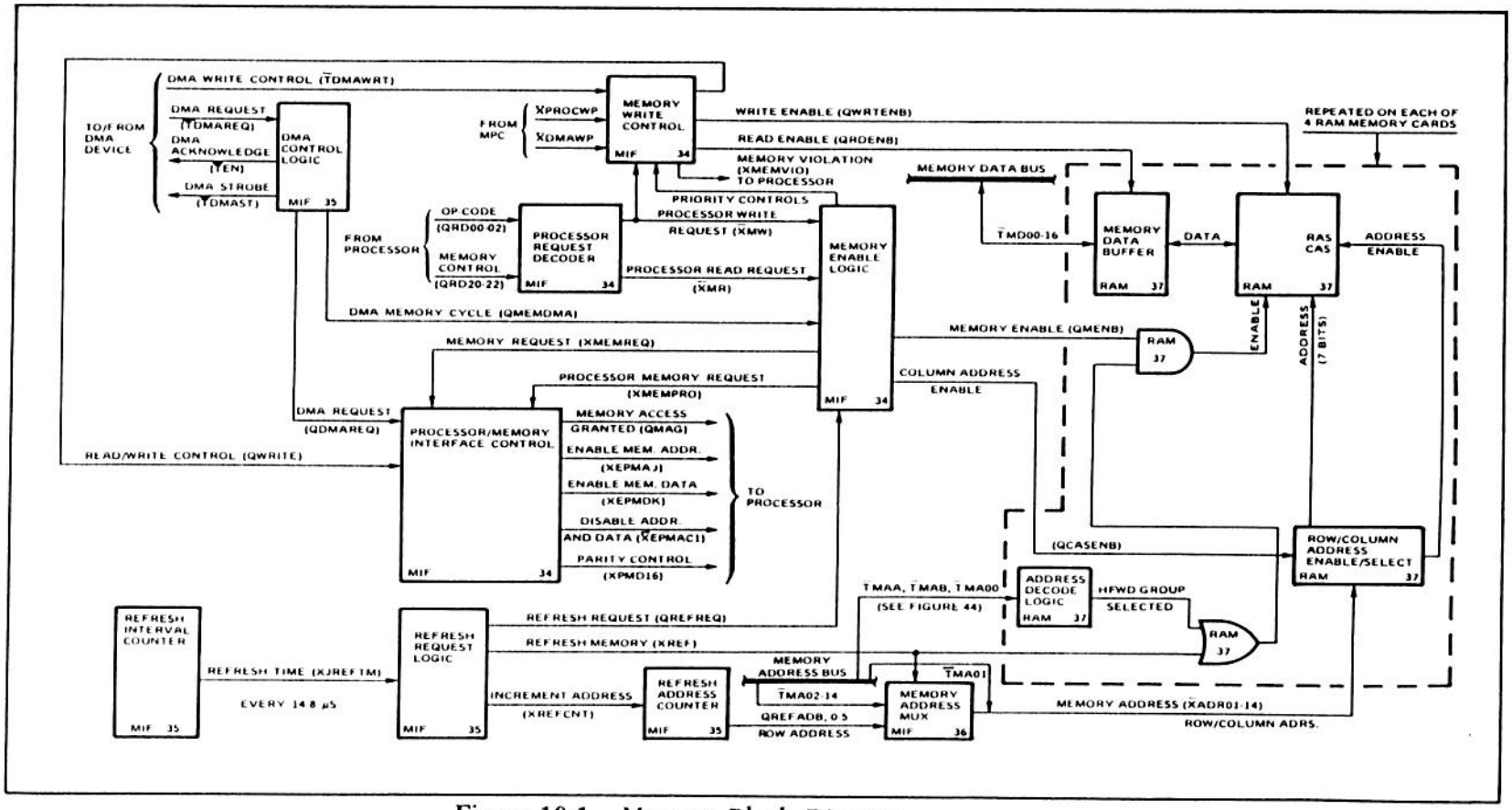


Figure 10-1. Memory Block Diagram

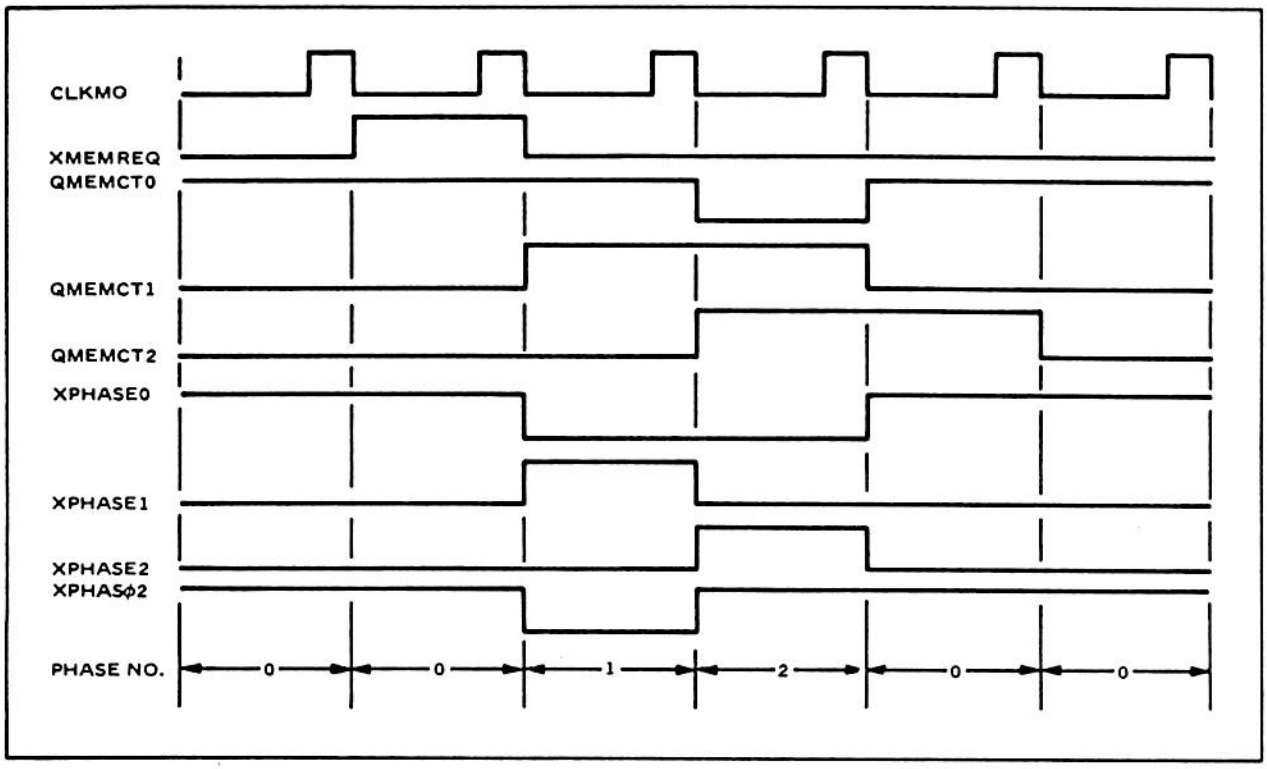


Figure 10-2. Memory Phase Counter Timing

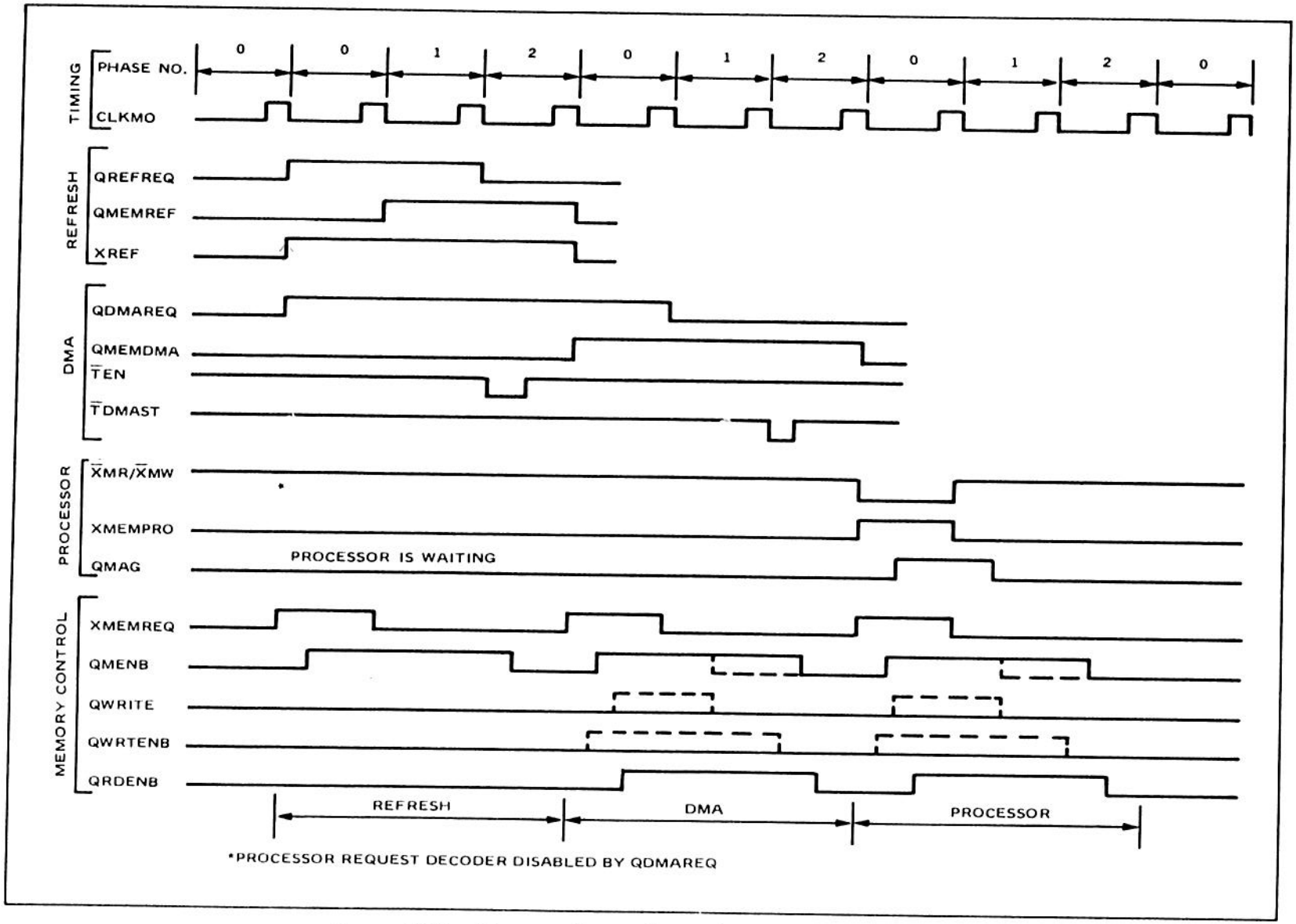


Figure 10-3. Memory Contention and Control Timing

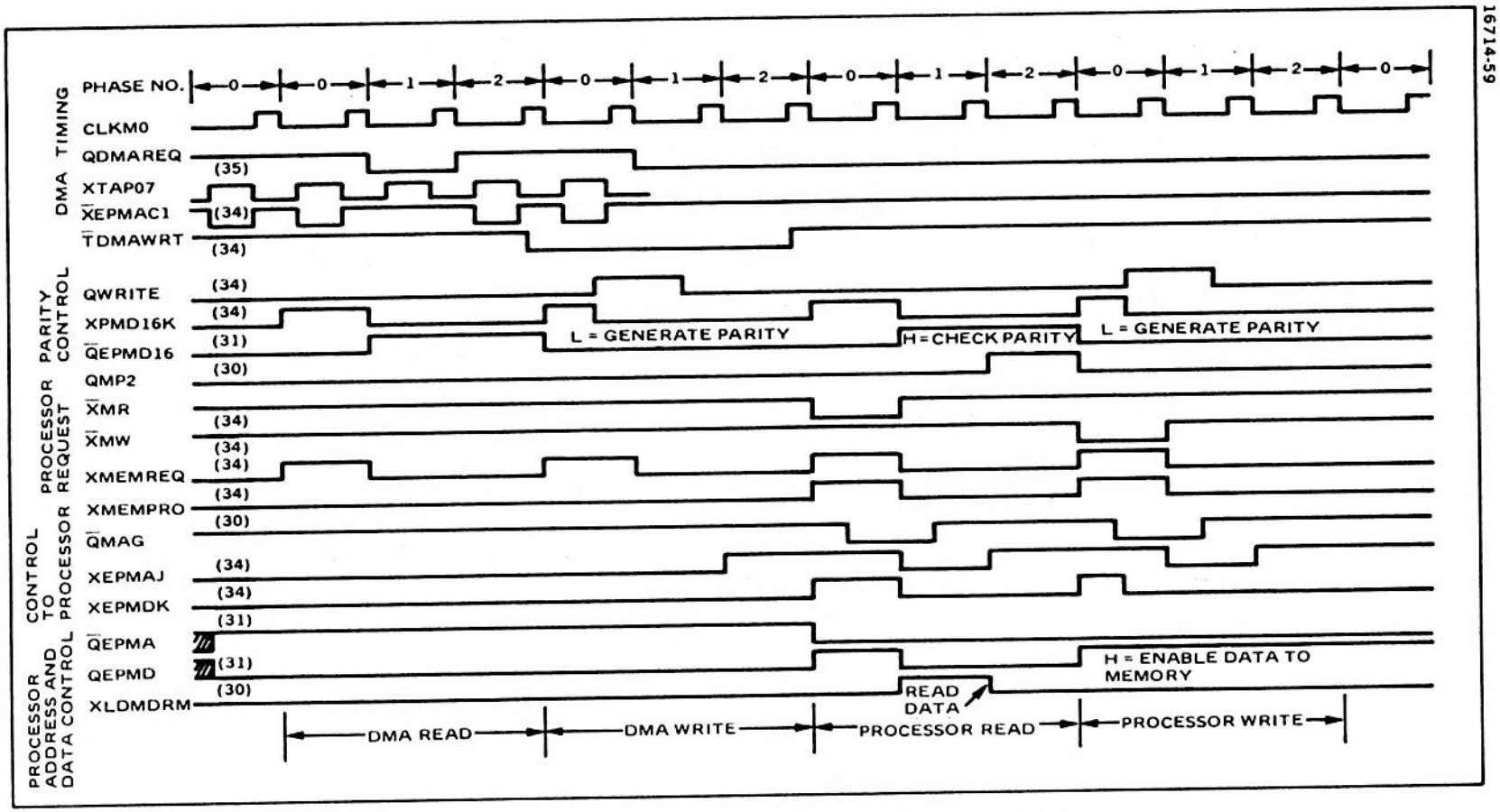


Figure 10-4. Processor/Memory Interface Timing

11.0 POWER FAULT DETECT

11.1 Introduction. The HMP-1116 has four DC power supplies. Three of the power supplies are requred to power the MOSRAM memory, they are +5V AUX, -5V and +12V. They will be active when the computer is switched to STBY, ON or LOCK. These power supplies are backed up by a battery for up to 30 minutes in the case of a prime power (115VAC) failure. The fourth power supply (+5V) provides the power used by the processor. It is turned on only when the computer is switched to ON or LOCK.

The Power Fault Detect function is responsible for:

1. Monitoring the four DC power supplies voltage levels
2. Controlling the STBY and ON lamps on the PMP
3. Controlling Master Clear
4. Turning on the +5V power supply
5. Generating the Clock Enable for the processor
6. Generating power fail interrupts for the processor

Figure 11-1 is a block diagram of the Power Fault Detect function.

11.2 Compare Logic. The Compare Logic monitors the voltage outputs of the DC power supplies. Memory Supply Out of Tolerance ($\overline{X}$MSOT) will be active if either +5V AUX, +12V or -5V are out of tolerance. DC Out of Tolerance ($\overline{X}$DCOT) will be active if any one of the four DC power supplies is out of tolerance. $\overline{X}$FVAOT active indicates that the +5V AUX power supply is out of tolerance.

11.3 Lamp Logic. The Lamp Logic receives $\overline{X}$DCOT and $\overline{X}$MSOT along with $\overline{X}$PPOT which when active indicates that the AC power is out of tolerance. It also receives $\overline{S}$OFF, $\overline{S}$STBY and $\overline{S}$ONLK. $\overline{S}$OFF active indicates that the Key Switch on the PMP is in the OFF position, $\overline{S}$STBY indicates STBY (standby) position and $\overline{S}$ONLK indicates either the ON or LOCK positions.

When $\overline{S}$OFF is active, $\overline{L}$PON and $\overline{L}$PSTBY are inactive disabling the ON and STBY lamps on the PMP. Also XPPEN is inactive. XPPEN active enables the +5V power supply and the processor.

When the key is switched to STBY the memory power supplies are turned on. $\overline{S}$STBY active and $\overline{X}$MSOT inactive causes $\overline{L}$STBY to go active turning on the STBY lamp. $\overline{X}$MSOT will remain inactive in a power failure as long as the battery continues to provide +5V AUX, -5V and +12V.

When the key is switched to ON, $\overline{\text{ONLK}}$ goes active. If $\overline{\text{XPPOT}}$ is inactive indicating AC power is in tolerance XPPEN goes active. $\overline{\text{XDCOT}}$ will become inactive after the +5V power supply is turned on, and this allows $\overline{\text{LPON}}$ to go active turning on the ON lamp.

11-4 Delay Driver Latch. The Delay Driver Latch is initially reset by $\overline{\text{XFVAOT}}$ making QPPEN and XFBA inactive. Once +5V AUX is in tolerance XPPEN will control QPPEN and XFBA. When XPPEN goes active QPPEN goes active and is sent to the Enable Logic. The Enable Logic generates APPEN which turns on the +5V power supply.

XFBA goes active 100ms after XPPEN. XPPEN and XFBA active along with $\overline{\text{XDCOT}}$ inactive disables the APFAIL (any power failure) signal. APFAIL inactive allows a power up sequence to occur.

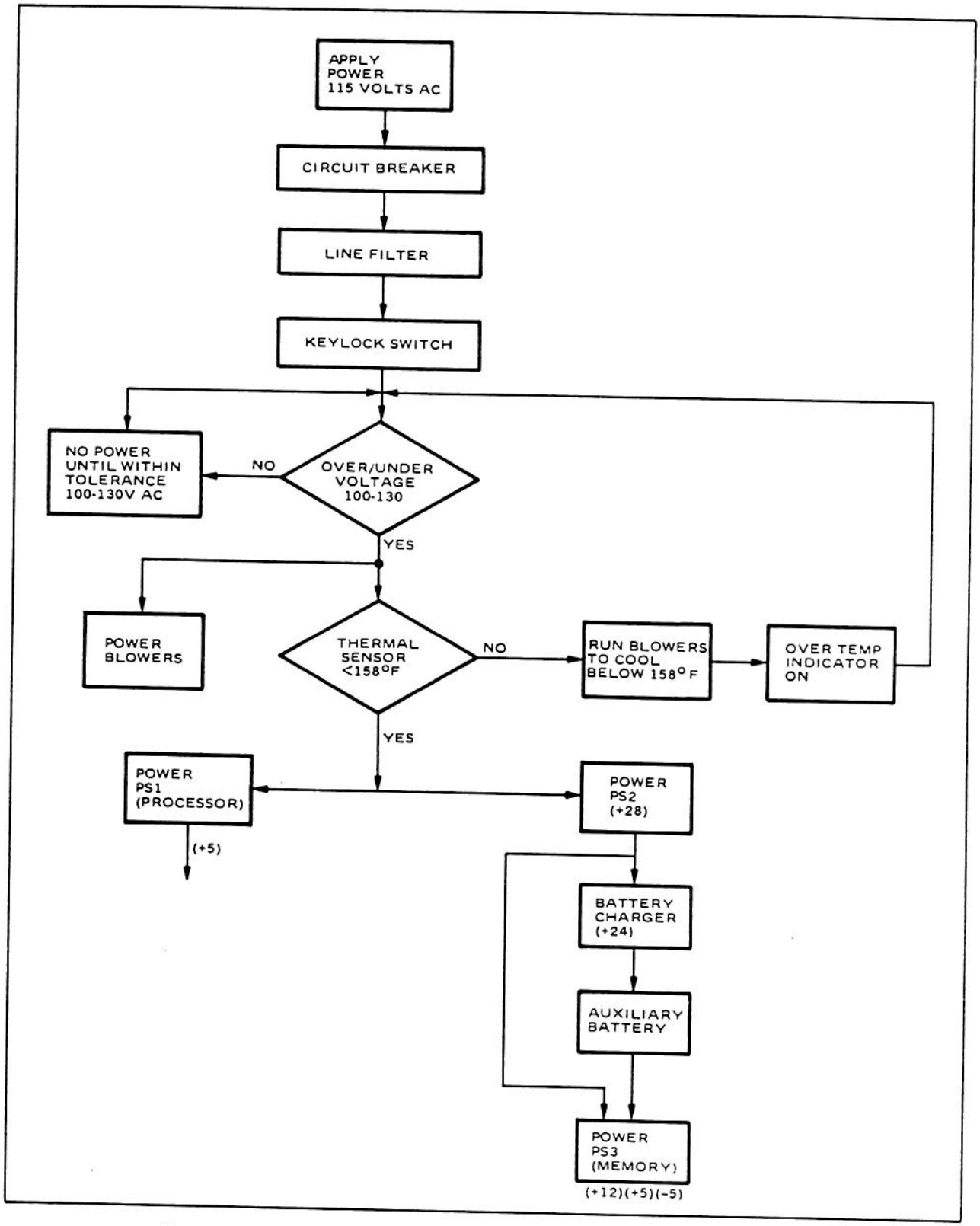
11-5. Power Up/Down Sequence Control Logic. The Power Up/Down Sequence Control Logic controls XTCLKEN which enables the processor clocks. The processor clocks are disabled at the end of a Down Sequence and enabled at the end of an Up Sequence. A Master Clear (SMCR) or a power failure (APFAIL) causes a Down Sequence. The logic remains in the down state until all power is in tolerance and the Master Clear pushbutton is released. SMCR will not cause a Down Sequence if SLOCK is active (Key Switch in LOCK position).

When APFAIL and SMCR are both inactive a Power Up Sequence begins. Master Clear ($\overline{\text{XMCR}}$) is removed from the processor and approximately 100ms later XTCLKEN goes active.

A Power Down Sequence begins by activating QEPF. QEPF can cause a Machine Malfunction interrupt, and it tells the software program that a Down Sequence is in progress. After 400 microseconds $\overline{\text{XPPF}}$ is generated. When the microprogram receives the $\overline{\text{XPPF}}$ interrupt, it halts software execution and saves the PSW and General Registers in RAM memory. 200 microseconds later XTCLKEN is removed and $\overline{\text{XMCR}}$ is generated.

Figure 11-2 shows the timing of the Lamp Logic, Delay Driver Latch, Fail Logic and Enable Logic.

Figure 11-3 shows the timing of the Power Up/Down Sequence Control Logic.



Power Decisions Flow Diagram HMP-1116 Power Up Sequence

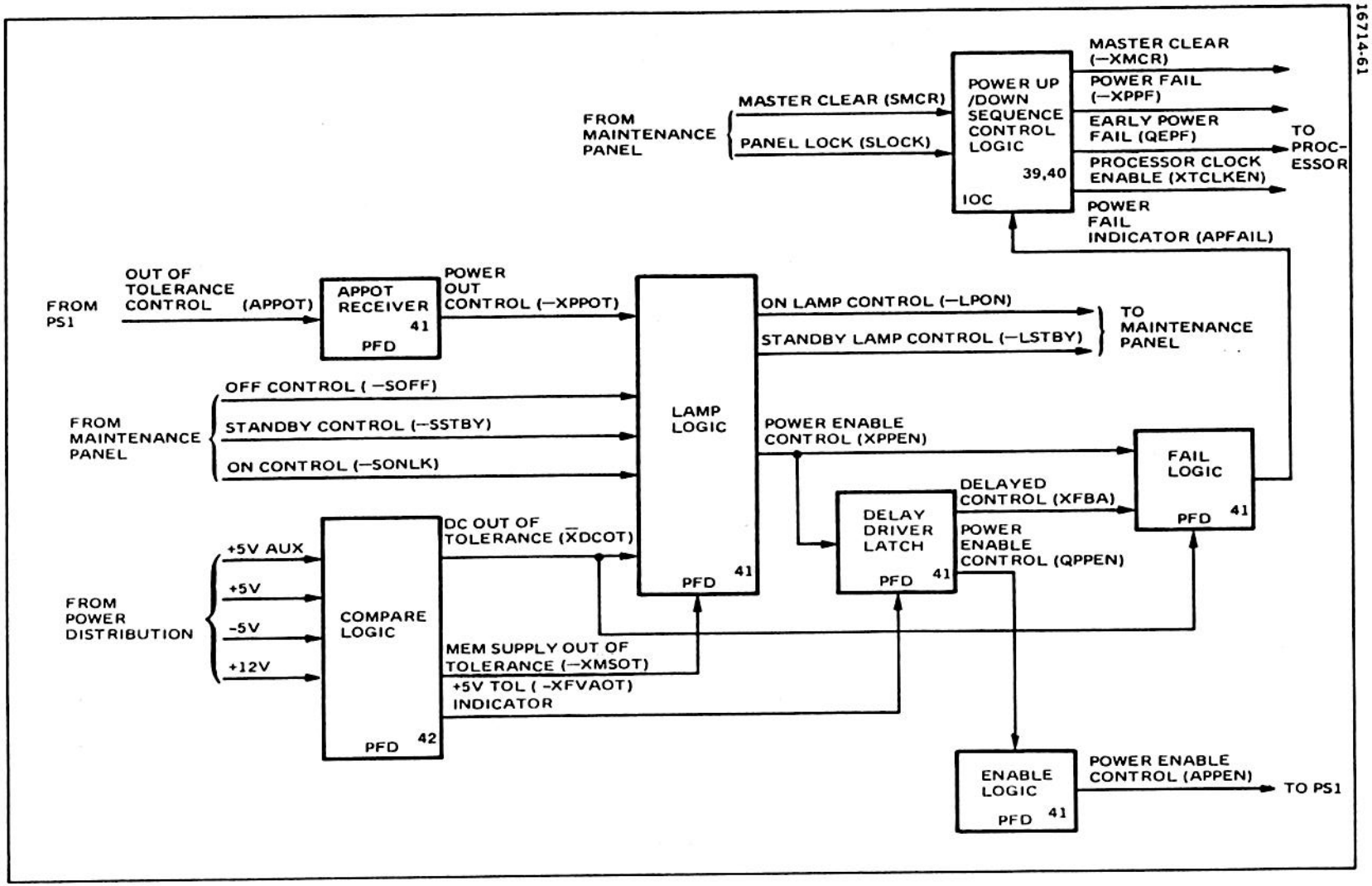


Figure 11-1. Power Fault Detect Block Diagram

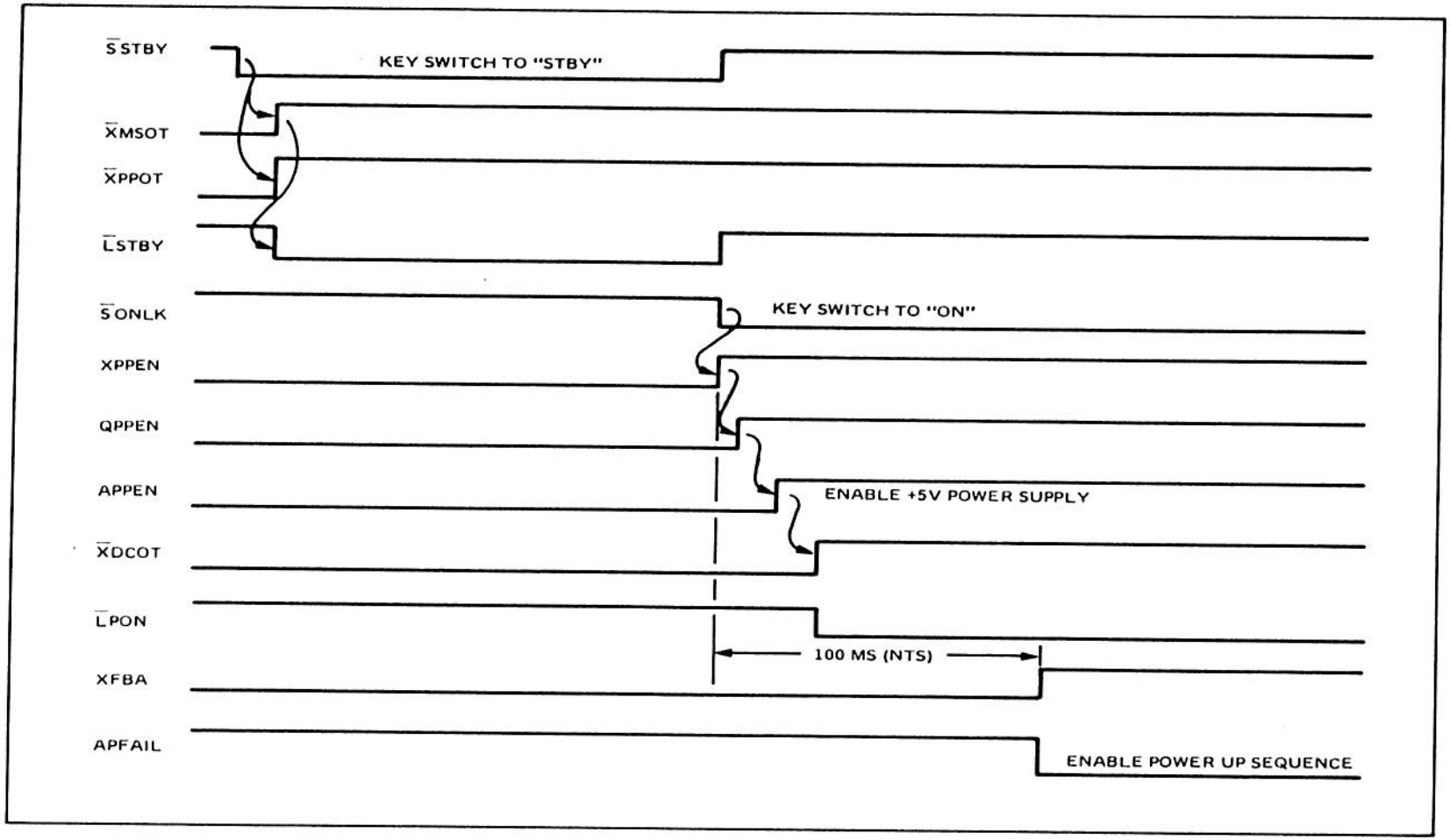


Figure 11-2. Standby/On Timing

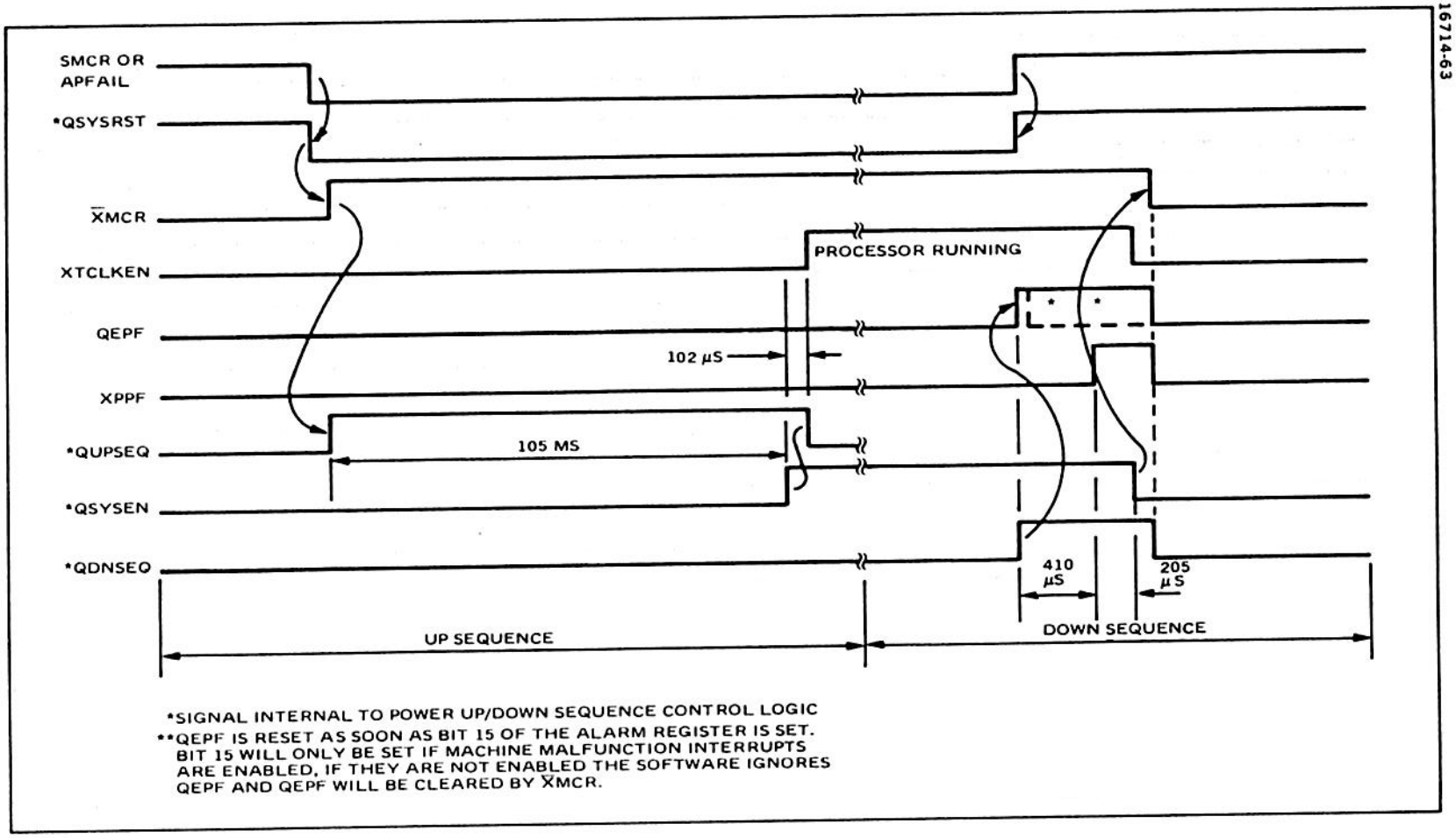


Figure 11-3. Power Up/Down Timing

Appendix A

DEFINITION OF MNEMONICS

APFAIL	Power failure imminent
APPEN	Processor power enable control
APPOT	Power out of tolerance
CLKERD	External control to load data into RDR
CLKEXT	External 20-MHz oscillator
CLKMO	Free running 5-MHz clock
CLK0	Inhibitable 5-MHz clock
CLK100	Inhibitable 5-MHz clock delayed by 100 nanoseconds
CLK2	Inhibitable 5-MHz clock delayed by 100 nanoseconds
CLK20	20-MHz clock
KMCR	Master clear control to external devices
LMOP	Output to MEMORY VALID lamp
LPON	Output to ON lamp
LSTBY	Output to STBYlamp
LWAIT	Output to WAIT lamp
QALM12 thru QALM15	Alarm register controls
QCHIA thru QCHID	Serial I/O channels A thru D
QCLKENB	Clocks inhibited
QCSV	Carry save control
QDMARQA thru QDMARQD	DMA memory request controls
QEPF	Early power failure
QEPMA	Enable processor-to-memory address control
QEPMD	Enable processor-to-memory data control
QFSYN	False sync control
QG	Compare greater than flag
QL	Compare less than flag
QLDCHA thru QLDCHD	Load channels A thru D for serial I/O channels A thru D
QMAG	Memory access granted for processor request
QMARA, QMARB	Memory bank extension controls
QMEMVIO	Protected memory violated
QMOP	Memory operational indicator
QNPT	Interrupt control (branch and link)

QOVF	Overflow flag
QPROGLD	Program load control
QPSWO2	Machine malfunction indicator control
QPSWO8, QPSWO9	Program memory bank select controls
QPSW10, QPSW11	Operand memory bank select controls
QRAR04 thru QRAR15	ROM address
QRDENB	ROM data enable control
QRD00 thru QRD35	ROM data word
QSMTH	Single cycle operation control
RD08 thru RD15	Byte outputs to DISPLAYS 1 and 2 on PMP
SD00 thru SD15	Outputs from DATA/ADDRESS switches
SEXEC	EXECUTE switch output
SFC4 thru SFC7	FUNCTION switch outputs
SLOCK	Power switch in LOCK position
SMCR	MASTER CLEAR output
SMTST	MEMORY TEST output
SOFF	Power switch in OFF position
SONLK	Power switch in LOCK position
$\overline{\text{S}}$PRLD	PROGRAM LOAD switch
SRESET	RESET switch
SRUN	Run switch output
SSGL	Single switch output
SSTBY	Power switch in STBY position
TACL	Computer acknowledges interrupt bit to I/O devices
TACKPCC	Acknowledge control from program control clock option
TACKSCC	Acknowledge control from serial I/O option
TADRS	Acknowledge request bit to I/O device
TATN	Attention bit to computer from I/O device
TCMD	Command bit to I/O device
TDA	Data available control
TDACK	Data acknowledge control
TDMAREQ	DMA request

TDMAST	DMA strobe control
TDMAWRT	DMA write into memory request
TDOA thru TDOD	Channels A thru D data out control
TDR	Data request
TD00 thru TD15	Data and control word to/from I/O device
TEN	Acknowledge control from memory
TMAA, TMAB	Memory address extension control from DMA bus
TMA00 thru TMA14	Memory address word from DMA bus
TMBIA thru TMBID	Channels A thru D message bracket in control
TMBOA thru TMBOD	Channels A thru D message bracket out control
TMD00 thru TMD16	Memory data word from DMA bus
TMEMFLT	Memory power supply fault
TPROPWR	Processor power control indicator to external device
TSR	Status request bit to I/O device
TSYN	Sync control bit from I/O device
TWAIA thru TWAID	Channels A thru D word accept in control
TWAOA thru TWAOD	Channels A thru D word accept out control
TWRIA thru TWRID	Channels A thru D word ready in control
TWROA thru TWROD	Channels A thru D word ready out control
XABORT	Interrupt request
XADRA, XADRB	Memory bank select control from MIF
XADR00 thru XADR14	Address control from MIF
XALUCIN	SLU carry input control
XAB00 thru XAB15	A-bus data
XBB00 thru XBB15	B-bus Data
XBCM1	Microinstruction mask field and flag compare control
XBCM2	Branch on external conditions (active during BCT or BCF instructions)
XCHADRA thru XCHADRD	Channels A thru D address enable control
XCHCMDA thru XCHCMDD	Channels A thru D data enable control
XCLKLSB	Clock LSB portion of ALU control
XCLKMSB	Clock MSB portion of ALU control
XCLRCIA thru XCLRCID	Clear channels A thru D interrupt request control

XCLRFLR	Clear flag register
XCLRRQA thru XCLRRQD	Clear channels A thru D request control
XCRYINM	ALU carry
XCSV	Carry save control
XCTONE	Count equals one
XCTZRO	Count equals zero
XDECTR	Decrement repeat counter
XDMAWP	DMA write request protect control
XEDSTP	External destination clock inhibit control
XEED	Enable destination clocks control
XEIFCH	Enable fetch instruction operation
XENBAB	External control to enable A-bus onto B-bus
XENMAN	Insert/extract control
XENRD	Enable ROM data
XEPFDC	Enable primary function decode
XERARAB	Enable B-bus multiplexer
XERDB	Enable ROM data onto B-bus control
XERDSTP	External RDR clock inhibit control
XESFDC	Enable secondary function decode
XEVNPTY	Parity error indicator control
XFEQOL	ALU output (from ARC card) equals zero (0)
XLA thru XLD	Enables DISPLAY LEDS
XLDCTR	Load request counter
XLDIR	Load instruction register
XLDMAR	Load memory address register
XLDMDRL	Load memory data register lower
XLDMDRU	Load memory data register upper
XLDPSW	Load power status word
XLDRAM	Load RAM
XLRXX	Enable external decode address to A-bus
XMALF	Machine malfunction control
XMCR	Master clear control
XOSCINH	Inhibit 20-MHz oscillator
XPHAS02	Memory phase count 2

XPPF	Primary power failure
XPROCWP	Processor write protect control
XQ15	LSB of 32-bit shift word
XRDSA	Enable extra clock for two-cycle operation
XRDSTP	Inhibit RDR clock for multicycle instruction
XRD00 thru XRD35	ROM input to RDR
XSELAB	Input RAR or A-bus data onto B-bus select
XSNGL	Single command control
XTCLKEN	Inhibit clocks

Appendix B

MICROPROGRAM LISTING DESCRIPTION

MICROPROGRAM PROGRAM LISTING DESCRIPTION

The computer uses a set of machine instructions which are stored in a Read-Only Memory (ROM). Sequences of these machine instructions, or microinstructions, are used to form microinstructions. Table B-1 summarizes the microinstruction set of the computer. Items that are underlined are optional and may be omitted along with the preceding comma. Items in lowercase letters are supplied as parameters by the user; these are explained below. Uppercase items and punctuation are written as shown. Keyword modifiers ("DEST=") may be coded in any order, but must, as a group, follow the last of the positional operands; the keyword and equal sign are written as shown with the user's choice of modifier following the equal sign.

The three columns on the right of Table B-1 show how the flags can be set by each instruction. By coding C, V, F, or any combination of these letters in the "flr" modifier, the user specifies which flags (if any) he wants loaded during the operation. F indicates that both the G and L flags are loaded; C and V refer to the C and V flags respectively. Although the loading of the flags depends on the "flr" modifier, the value to be loaded into each flag depends on the individual instructions, as shown in these three columns.

Cout denotes the carry out of the ALU' -Cout denotes the inverse of Cout, usually interpreted as "borrow out." Q(15) denotes the LSB of Q; regs (00) denotes the MSB of regs. 0 indicates that a value of zero is used when loading the flag. 0v indicates the overflow output of the ALU. V+Sh0v indicates that the shift overflow function is 0Red into the V flag; shift overflow occurs when regs (00) is different from the current contents of the C flag. Fsync indicates that V is loaded from the False Sync input to the Micro-1632; note that this loading is unconditional, since there is no "flr" modifier on Input or Output. On Input, when FLR is the destination, Fsync is 0Red with the corresponding input bit. SP indicates that the setting of G and L is for a single precision result; these instructions should be used for single precision calculations, and for the least significant word of multiple precision calculations. DP denotes multiple precision setting of G and L; in this case, the new values of G and L depend on the current ALU output, as well as the previous state of G and L. (For example, a multiple precision result is not zero unless every word of the result was zero.) In the shift instructions, SP (regs) indicates that the setting of these flags is based on the unshifted contents of registers.

Table B-1. Summary of Microinstructions

OP-Code	Operands and Modifiers	C	V	F
LR	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	SP
LRTC	regd,regs,flr,MOD=mod,DEST=INHIB	-Cout	0v	SP
LROC	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	SP
IR	regd,regs,flr,MOD=mod,DEST=INHIB	Cout	0v	SP
DR	regd,regs,flr,MOD=mod,DEST=INHIB	-Cout	0v	SP
* AR	regd,regs,flr,MOD=mod,DEST=INHIB	Cout	0v	SP
* SR	regd,regs,flr,MOD=mod,DEST=INHIB	-Cout	0v	SP
ARQR	regd,regs,flr,MOD=mod,DEST=INHIB	Cout	0v	SP
* AQRR	regd,regs,flr,MOD=mod,DEST=INHIB	Cout	0v	SP
* XR	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	SP
* NR	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	SP
* OR	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	SP
LR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
LRTC	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
LROC	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
IR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
DR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
* AR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
* SR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
ARQR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
* AQRR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
* XR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
* NR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
* OR	regd,regs,ST=st,DC=dc,MC=mc,RPT=rpt			
* CR	regd,regs,flr,MOD=mod	-Cout	0v	SP
LRTCM	regd,regs,flr,MOD=mod,DEST=INHIB	-Cout	0v	DP
LRM	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	DP
IRC	regd,regs,flr,MOD=mod,DEST=INHIB	Cout	0v	DP
DRB	regd,regs,flr,MOD=mod,DEST=INHIB	-Cout	0v	DP
* ARC	regd,regs,flr,MOD=mod,DEST=INHIB	Cout	0v	DP
* SRB	regd,regs,flr,MOD=mod,DEST=INHIB	-Cout	0v	DP

Table B-1. Summary of Microinstructions (Continued)

OP-Code	Operands and Modifiers	C	V	F
* XRM	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	DP
* NRM	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	DP
* ORM	regd,regs,flr,MOD=mod,DEST=INHIB	0	0	DP
* CRB	regd,regs,flr,MOD=mod	-Cout	0v	DP
CMD	ST=st,DC=dc,MC=mc			
NOP				
$ MUL	regd,regs,flr	Q(15)	0v	SP
$ MULC	regd,regs,flr	Q(15)	0v	SP
$ DIV	regd,regs,flr	-Cout	0v	SP
$ RL	regd,regs,flr,DEST=INHIB	regs(00)	V+Sh0v	SP(regs)
$ PLA	regd,regs,flr,DEST=INHIB	regs(00)	V+Sh0v	SP(regs)
$ PLL	regd,regs,flr,DEST=INHIB	regs(00)	0	SP(regs)
$ RR	regd,regs,flr,DEST=INHIB	Q(15)	V+Sh0V	SP(regs)
$ PRA	regd,regs,flr,DEST=INHIB	Q(15)	V+Sh0v	SP(regs)
$ PRL	regd,regs,flr,DEST=INHIB	Q(15)	0	SP(regs)
$ PLZ	regd,flr,DEST=INHIB	0	0	0
$ PRZ	regd,flr,DEST=INHIB	Q(15)	0	0
$ CRYZ	regd,flr,DEST=INHIB	0	0	0
LI	regd,imm,flr,MOD=mod,DEST=INHIB	-	0	SP
LITC	regd,imm,flr,MOD=mod,DEST=INHIB	0	0v	0
LIOC	regd,imm,flr,MOD=mod,DEST=INHIB	-	0	SP
* AI	regd,imm,flr,MOD=mod,DEST=INHIB	-	0v	SP
* SI	regd,imm,flr,MOD=mod,DEST=INHIB	-	0v	SP
* CI	regd,imm,flr,MOD=mod	-	0v	SP
* NI	regd,imm,flr,MOD=mod,DEST=INHIB	-	0	SP
* OI	regd,imm,flr,MOD=mod,DEST=INHIB	-	0	SP
* XI	regd,imm,flr,MOD=mod,DEST=INHIB	-	0	SP
ARQI	regd,imm,flr,MOD=mod,DEST=INHIB	-	0v	SP
B	addr			
BCT	cond;addr			

Table B-1. Summary of Microinstructions (Continued)

OP-Code	Operands and Modifiers	C	V	F
BCF	cond,addr			
$ BI	addr,reg			
$ BICF	cond,addr,reg			
$ BLK	addr,reg			
OUT	reg,oflag,iflag,MOD=mod,BC=bc	-	FSync	-
INP	reg,oflag,iflag,MOD=mod,BC=bc	-	FSync	-

Legend:

regd = destination register; regs = source register; flr = flag register

Table B-2 shows the function of each instruction.

Table B-2. Function of Microinstructions

Op-Code	Name	Function
LR	Load Register	regd ← MOD(regs)
LRTC	Load Reg Twos Complement	regd ← 0-MOD(regs)
LROC	Load Reg Ones Complement	regd ← (-1) XOR MOD(regs)
IR	Increment Reg	regd ← MOD(regs)+1
DR	Decrement Reg	regd ← MOD(regs)-1
AR	Add Reg	regd ← regd+MOD(regs)
SR	Subtract Reg	regd ← regd-MOD(regs)
ARQR	Load Reg With Sum	regd ← Q+MOD(regs)
AQRR	Load Q With Sum	q ← regd+MOD(regs)
XR	Exclusive OR Reg	regd ← regd XOR MOD(regs)
NR	AND Reg	regd ← regd AND MOD(regs)
OR	OR Reg	regd ← regd OR MOD(regs
CR	Compare Reg	calculate: regd-MOD(regs)
LRM	Load Reg Multiple	regd ← (MOD(regs)
LRTCM	Load Reg Twos Comp Multiple	regd ← 0-MOD(regs)-C
IRC	Increment Reg With Carry	regd ← MOD(regs)+C

Table B-2. Function of Microinstructions (Continued)

Op-Code	Name	Function
DRB	Decrement Reg With Borrow	regd ← MOD(regs)-C
ARC	Add Reg With Carry	regd ← regd+MOD(regs)+C
SRB	Subtract Reg With Borrow	regd ← regd-MOD(regs)-C
XRM	Exclusive-OR Reg Multiple	regd ← regd XOR MOD(regs)
NRM	AND Reg Multiple	regd ← regd AND MOD (regs)
ORM	OR Reg Multiple	regd ← regd OR MOD(regs)
CRB	Compare Reg With Borrow	calculate: regd-MOD(regs)-C
CMD	Command	specified keyword function only
NOP	No Operation	no operation
MUL	Multiply Step	if C=0: (regd,Q) ← (regd,Q)/2 if C=1: (regd,Q) ← (regd+regs,Q)
MULC	Multiply Correction	(regd,Q) ← (regd-regs,Q)/2
DIV	Divide Step	if C=0: (regd,Q) ← (regd-regs,Q)*2+Cout if C=1: (regd,Q) ← (regd+regs,Q)*2+Cout
RL	Rotate Left	(regd,Q) ← (regs,Q)*w+regs(00)
PLA	Position Left Arithmetic	(regd,Q) ← (regs,Q)*2
PLL	Position Left Logical	(regd,Q) ← (regs,Q)*2
RR	Rotate Right	(regd,Q) ← Q(15),(regs,Q)/2
PRA	Position Right Arithmetic	(regd,Q) ← C,(regs,Q)/2
PRL	Position Right Logical	(regd,Q) ← 0,(regs,Q)/2
PLZ	Position Left Zeros	(regd,Q) ← (0,Q)*2
PRZ	Position Right Zeros	(regd,g) ← 0,(0,g)/2
CRYS	Carry Save	regd ← C0/2
LI	Load Immediate	regd ← MOD(imm)
LITC	Load Immediate Twos Comp	regd ← 0-MOD(imm)
LIOC	Load Immediate Ones Comp	regd ← (-1) XOR MOD (imm)
A1	Add Immediate	regd ← regd+MOD(imm)
S1	Subtract immediate	regd ← regd-MOD(imm)
C1	Compare Immediate	calculate: regd-MOD(imm)
NI	AND Immediate	regd ← regd AND MOD(imm)

Table B.-2. Function of Microinstructions (Continued)

Op-Code	Name	Function
OI	OR Immediate	regd ← regd OR MOD(imm)
XI	Exclusive-OR Immediate	regd ← regd XOR MOD(imm)
ARQI	Load Reg With Immediate Sum	regd ← A + MOD(imm)
B	Branch	RAR ← addr
BCT	Branch on Condition True	cond false: non operation cond true: RAR ← addr
BCF	Branch on Condition False	cond false: RAR ← addr cond true: no operation
BI	Branch Indexed	RAR ← addr+reg
BICF	Branch Indexed Cond False	cond false: RAR ← addr+reg cond true: no operation
BLK	Branch and Link	reg ← RAR, RAR ← addr
OUT	Output	output bus ← reg
IN	Input	reg ← input bus

Table B-3 shows the form for each operand and modifier. In the explanation of branch conditions, + indicates that the selected flag register bits are ORed together. Branch condition 'CTR' specifies a mask of all zeros; the hardware may be configured to test an external counter when a Branch on Condition True instruction specified this condition. In the "st" operands, actions in parentheses are not actually implemented within the hardware. The notation "(arbitrary meaning)" indicates other functions which are not implemented in the basic computer.

Note that a number of instructions can appear in two forms, for example LR and AR. Such instructions can be coded in either form, but no single instance of such an instruction may use modifiers from both formats. The "flr, MOD = ..." format is called RR format; the "ST = st, DC = ..." format is called CT or control format. The instructions which have an immediate operand ("imm") are called RI or register immediate format instructions.

In instructions marked with a dollar sign ($), both registers must be internal scratchpad registers; external registers and Q are not allowed. Branch Indexed, Branch Indexed on Condition False, and Branch and Link are also marked with a dollar sign, signifying that "reg" must be an internal scratchpad register. Asterisk (*) marks instructions in which the user may not specify both registers as external; one or both must be internal or Q. RI instructions marked with asterisks may not have "regd" an external register.

The folloiwng combination is not allowed in RR format instructions:

1. regs internal
2. regd external
3. MOD applied

Another combination forbidden in RR format is:

1. regs internal
2. regd internal
3. MOD applied
4. F included in "flr" (F, CF, VF, or CVF)

Still another combination forbidden in RR format is:

1. regs = Q
2. MOD applied

The following is not allowed in RI format:

C included in "flr" (C, CF, CV, CVF)

Table B-3. Assembler Operands and Modifiers

Parameter	Meaning	Assembler Input
reg	Register Operand	register symbol (internal or external)
regd	Register Operand	register symbol (internal or external)
regs	Register Operand	register symbol (internal or external)
imm	Immediate Operand	literal exper (0<=imm<=255)
flr	Flag Register Control	(default) = no change F = load G,L according to cur instr V = load V according to cur instr VF = load V,G,L according to cur instr C = load C according to cur instr CF = load C,G,L according to cur instr CV = load C,V according to cur instr CVF = load C,V,G,L according to cur instr
mod	Operand Modifier	(default) = no modification SWAP = swap bytes INSP = insert byte (lef or right) EXTB = extract byte (left or right) EXT0 = extract digit 0 (leftmost) EXT1 = extract digit 1 EXT2 = extract digit 2 EXT3 = extract digit 3 (rightmost)

Table B-3. Assembler Operands and Modifiers (Continued)

Parameter	Meaning	Assembler Input
dest	Dest Clock Inhibit	(default) = 'regd' loaded (except Compare's) INHIB = 'regd' not changed
cond	Branch Condition	CTR = mask is B'0000' L = mask is B'0001' G = mask is B'0010' GL = mask is B'0011' V = mask is B'0100' VL = mask is B'0101' VG = mask is B'0110' VGL = mask is B'0111' C = mask is B'1000' CL = mask is B'1001' CG = mask is B'1010' CGL = mask is B'1011' CV = mask is B'1100' CVL = mask is B'1101' CVG = mask is B'1110 CVGL = mask is B'1111'
addr	Branch Address	address expression
st	Status Control	(default) = no change JMC = CVGL ← 0, (cond Code ← CVGL) ALM = (Cond Code ← Alarms) JAM = (Cond Code ← CVGL) AMC = (Cond Code ← Alarms), Alarms ← 0) CLR = CVGL ← 0 SIE = set interrupt-enable RIE = reset interrupt-enable LFG = CVGL ← A-Bus(0..3) IEL = CVGL ← A-Bus(0..3), set intrpt-
dc	Decode Control	(default) = normal sequencing literal expr = decode function (1<dc<=3)
mc	Memory Control (or any synchronous device)	(default) = 0 literal expr = (arbitrary meaning) (0≤ mc≤ =7)
oflag	Output Flag	literal expr = (arbitrary meaning) (≤ =oflag ≤=31)
iflag	Input Flag	(default) = 0 literal expr = (arbitrary meaning) (=≤=iflag ≤=31)

Table B-3. Assembler Operands and Modifiers (Continued)

Parameter	Meaning	Assembler Input
bc	I/O Bus Control (or any I/O option)	(default) = 0 literal expr = (arbitrary meaning)(0§=bc§=3)
rpt	Instruction Repetition	(default) = 0 literal expr = (arbitrary meaning)(0§=rpt§=1)