HP 1000 E-Series and F-Series Computer Microprogramming

Reference Manual

HEWLETT hp PACKARD

HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA, 95014

Library Index Number
2MICRO.320.02109-90004

PART NO. 02109-90004

Printed: July 1978
Printed in U.S.A.

LIST OF EFFECTIVE PAGES

Changed pages are identified by a change number adjacent to the page number. Changed information is indicated by a vertical line in the outer margin of the page. Original pages do not include a change number and are indicated as change number 0 on this page. Insert latest changed pages and destroy superseded pages.

Change 0 (Original) ..... July 1978

PREFACE

Are you looking for a better way to accomplish your applications program tasks? Have you used all the programming methods you can think of to make your library subroutines run as efficiently as possible in your Real Time Executive (RTE) Operating System environment? Maybe its time to look into microprogramming.

Primarily, microprogramming is the use of a discrete language to effect control of a specific computer at the closest possible level without hardware redesign so that you may have the advantage of executing selected main memory programs at the fastest possible rate available in the computer. Some other purposes for microprogramming that may be of interest to you are mentioned in section 1 of this manual.

This manual consists of four parts and eight appendixes that will provide you with the information necessary to prepare and integrate your microprograms into HP 1000 E-Series or F-Series Computers, then execute them when desired. You will find subjects organized as follows:

Part I - Why Microprogramming?

- Program analysis.
- An overview of microprogramming.
- Microprogrammable functions of HP 1000 E-Series and F-Series Computers.

Part II - Microprogramming Methods.

- Microinstruction formats, definitions, and timing.
- Gaining access to your microprogramming area.
- How to prepare microprograms.

Part III Microprogramming Support Software and Hardware.

- How to microassemble and load object microprograms.
- Using microprogramming support software such as the:
  - Microdebug Editor (MDE).
  - Writable Control Store (WCS) I/O Utility Routine (WLOAD) and WCS Real Time Executive (RTE) Driver DVR36.
  - Programmable Read-Only-Memory (pROM) Tape Generator.
- Using pROM hardware facilities.
- Using extra features of the E-Series and F-Series Computers.

Part IV Microprogramming Examples.

Appendixes

- Microprogramming reference material.
- The HP 1000 E-Series Computer base set microprogram listing and F-Series jump tables.

This manual is written for those individuals who have experience as Assembly language programmers and are familiar with Hewlett-Packard RTE Operating Systems.

The documentation map that follows is a diagram of related manuals. Parts II and III of this manual contain additional information about microprogramming support software.

This manual is written for those individuals who have experience as Assembly language programmers and are familiar with Hewlett-Packard RTE Operating Systems.

Parts II and III of this manual contain additional information about microprogramming support software and manuals.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

ILLUSTRATIONS

TABLES

PART I
Why Microprogramming?

Section 1
MICROPROGRAMMING CONCEPT

MICROPROGRAMMING CONCEPT

SECTION 1

Why microprogramming? Because microprograms and microprogramming techniques can be used to. .

- Reduce program execution time. By microprogramming often-used routines you can significantly decrease the program execution time. Large reductions in execution time are enabled because:
  - Many instruction fetches are eliminated.
  - Microinstructions execute (typically) four to ten times faster than Assembler instructions.
  - Multiple operations can occur during a single microinstruction.
  - The microinstruction word width (24 bits) provides a larger instruction repertoire than available with the Assembler word width (16 bits).
  - Many more registers and functions at the microinstruction level are available to you than to the higher level language programmer.
- Implement customized computer instructions. Designing customized instructions (i.e., microprograms) can provide facilities not otherwise readily available. Examples are:
  - Postindexing and/or preindexing.
  - Stack instructions.
  - Special arithmetic instructions (double integer, decimal, etc.).

What types of applications can be microprogrammed?

- Sort routines (e.g., bubble, shell, radix-exchange, and quicksort).
- High-speed or specialized input/output (I/O) transfer operations.
- Table searches (e.g., sequential, binary, and link-list).
- Arithmetic Floating Point Calculations.
- Transcendental functions (e.g., sine, square root, and logarithms).
- Fast Fourier Transform (FFT).

You may also create microprograms to control your own customized hardware. References for microprogrammable algorithms for many of the above applications are given in part IV.

Then why not microprogram everything?

- Microprogramming everything would be an unwieldly and unprofitable project. An analysis should be made to determine those areas that need to be microprogrammed.

- Microprograms are not relocatable in control memory.

- Microprograms run separately from the operating system and, when invoked, are in complete control of the computer. Therefore, if you don't plan carefully, the operating system's peripheral devices, memory, and computer management can be defeated, or even aborted.

Although additional effort is required to become more familiar with the computer in order to write microprograms, the results will be well worth the effort. The following paragraphs outline the considerations involved when you decide to microprogram.

1-1. MICROPROGRAMMING OVERVIEW

What is the first thing to consider? Typically, an application program, or perhaps a library routine running in an RTE environment, may need to have a faster execution speed. This may or may not be obvious in external operation (i.e., waiting time is too long for a line printer output when a certain calculation is performed, terminal response too slow, etc.). Whether the excessive time taken is obvious or not, some method must be used to analyze the programming environment so that you can identify these areas. Three basic methods can be considered to determine which areas of the program (memory) are consuming the most computer time:

- Programming analysis devices may be attached to the computer; this is the most accurate but most expensive method.

- A programmatical analysis method may be used as a middle-of-the-road approach.

- The computer can be checked manually at periodic intervals (i.e., every 10 or 15 seconds) by halting and recording the program counter (P-register) contents. A profile can thus be obtained, and a map of the "busy" areas generated; however, this is a tedious and time-consuming task, but a minimum of material cost is involved.

In summary, it can be seen that the first step is to find out what you're going to microprogram. The point is that if you spend your time microprogramming some seldom-used library routine, you cannot expect to realize a significant gain in software efficiency.

1-2. SELECTING AN ANALYSIS METHOD

The analysis method we'll consider in this manual is a middle-of-the-road approach. That is, an activity profile generation type of program. For example, you can:

- Use an I/O device capable of generating interrupts and cause periodic interrupts to the operating system.
- Reserve a "word block counter" for (as an example) every 500 words of main memory.

Each time the device interrupts, the P-register could be sampled and the count incremented for the associated "word block counter". That is, a record is generated for the program location counter at periodic intervals. This can be done several hundred thousand times and, at the end of the sample period, a percentage of time spent in each area of memory can be obtained. Then. . .

- The load map of the program being analyzed can be examined to determine which part(s) of the program could possibly be microprogrammed to decrease the execution time.
- The resolution for your analysis program could be changed, as could other parameters in the program, to obtain the desired profile.

This is the general idea of how an activity profile generation program could be used. Also, you may want to refer to the *Contributed Library Catalog*, part no. 22999-90040, for programs you may be able to use.

Once your activity profile generation program output is analyzed, it may be found that some specific routines (perhaps library subroutines) are indeed consuming too much computer time. Once the analysis is complete, you're ready to concentrate on a particular area. But remember that:

- The maximum benefit of microprogramming will not be realized by simply imitating the Assembly language instructions in microroutines.
- In order to determine specifically what to microprogram, the computer functions and program intent should be studied before you begin to write your microprogram. The final result will be a microprogrammed solution that executes in much less time and is totally or at least partially transparent.

Now, what steps are necessary to get your microprogram into operation? An overview of the process follows.

1-3. THE MICROPROGRAMMING PROCESS

Figure 1-1 provides an overview of the steps involved in microprogramming and some explanation of the illustration may be helpful:

- After a program analysis has been accomplished, the entry point (address) for the control memory module that you'll be using must be determined.
- The microprogram is then written using the information given in part II of this manual.
- The microprogram source file can be prepared and stored on disc.

- The microassembler (program MICRO, which can be placed in the RTE system at generation time) is loaded into main memory.

- The microprogram source is then microassembled by MICRO and a listing and an object file can be obtained.

- At this point the Microdebug Editor (program MDEP, which can also placed in the RTE system at generation time) can be loaded into main memory. (The Microdebug Editor may also be called from your programs in the RTE environment by the name MDES.)

- The object microprogram may then be loaded into Writable Control Store (WCS) using the MDE. (Microprograms can also be loaded into WCS using other programs, such as WLOAD.)

- The microprogram can be debugged, edited, and checked out interactively using the MDE and WCS.

NOTE

The HP 13197A Writable Control Store Kit is an integral part of microprogramming. Information on writing microprograms to be stored in WCS is the primary purpose of this manual; however, installation and additional reference information on WCS will be found in the *HP 13197A Writable Control Store Reference Manual*, part no. 13197-90005. Information on the driver (necessary for operation of WCS in the RTE environment) and on the WCS I/O Utility routine WLOAD is included in the *RTE Driver DVR36 for HP 12978A/13197A Writable Control Store Board Programming and Reference Manual*, part no. 13197-90001.

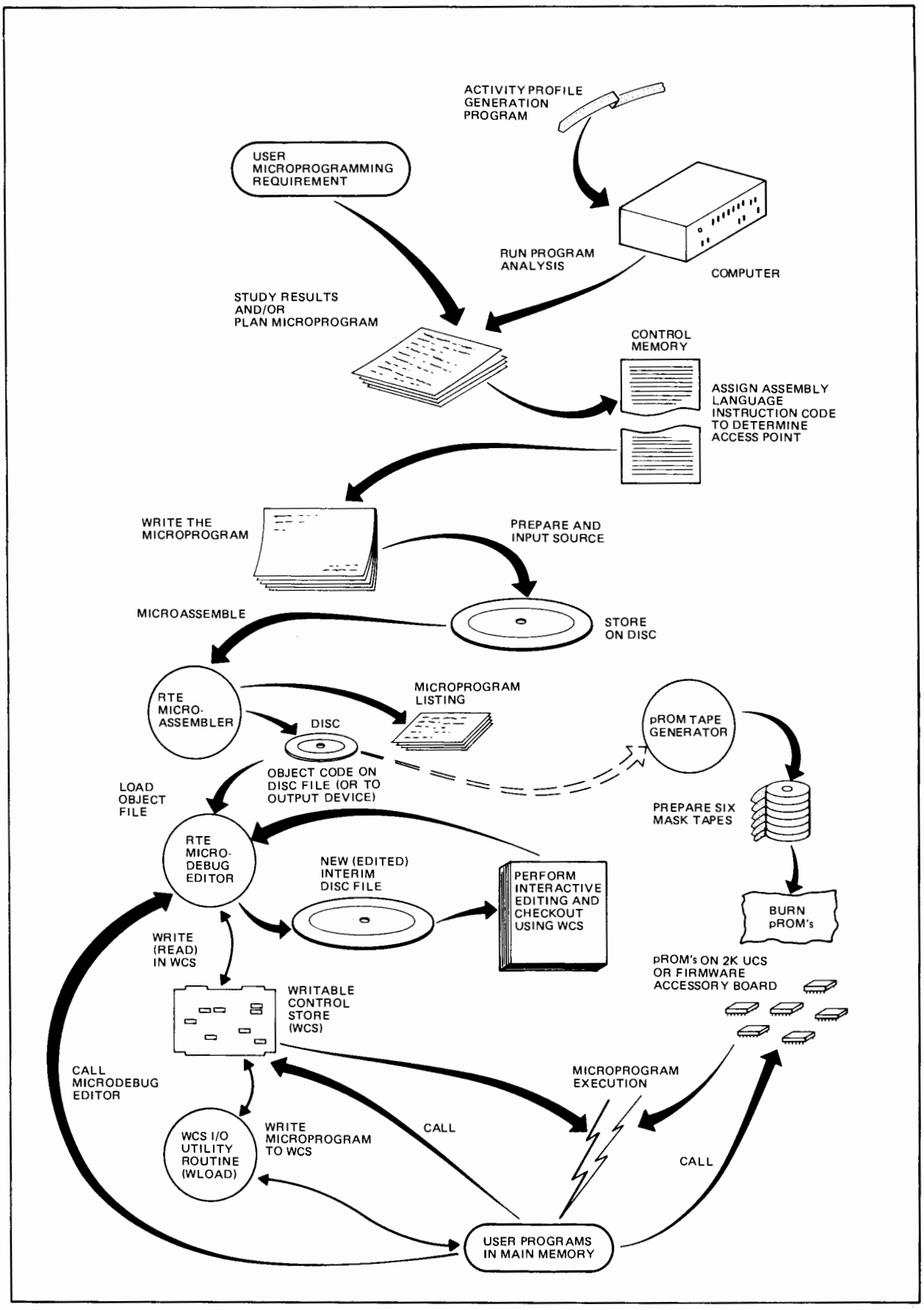
The ready-to-run microprogram can be stored in one of two ways:

- It can be left in WCS.

- You can create a permanent microprogram through the use of the pROM Tape Generator microprogramming support software. This software, in turn, can be used to generate several different types of mask tapes that can be used to have Programmable Read Only Memory (pROM's) fused (burned). The pROM's can then be installed on the HP 13304A Firmware Accessory Board (FAB) (attached to the CPU) or on the HP 13047A User Control Store (UCS) Kit (in the I/O card cage).

NOTE

Information on the pROM Tape Generator (as well as on the RTE Microassembler and RTE Microdebug Editor microprogramming support software) is included in this manual. Information you will need for using pROM's can be found in the *HP 13304A Firmware Accessory Board Installation and Service Manual*, part no. 13304-90001 and the *HP 13047A User Control Store Kit Installation and Service Manual*, part no. 13047-90001.



7115-1

Figure 1-1. Microprogramming Implementation Process

The advantages of executing microprograms from WCS are:

- WCS may be reused for many microprograms.
- WCS may be used to dynamically swap microprograms in and out of the system to suit a variety of users.

The disadvantages are:

- Microprograms in WCS can be destroyed by an errant user of the system.
- When computer power is removed, your microprogram is lost and must be reloaded.
- Each WCS board requires an I/O slot in the computer.

The advantage of fusing (burning) pROM's is:

- The pROM's are permanently fused and the computer will not lose the microprogram when power is removed.

The disadvantage is:

- There is much more involved in changing the microprogram with pROM's than there is with WCS.

1-4. EXECUTING YOUR MICROPROGRAM

If your microprogram is stored in pROM's, it can be executed immediately through User Instruction Group (UIG) instructions (105xxx or 101xxx) that link Assembly language routines to microprograms. The hardware and firmware map each UIG instruction to a unique control memory destination.

If WCS is being used, your microprogram must initially be contained in WCS before execution. Microprograms that reside in WCS execute at the same speed as pROM's. Both WCS and pROM resident microprograms can be used along with the base set in control memory. (The base set is defined as the computer's standard instruction set microprograms.)

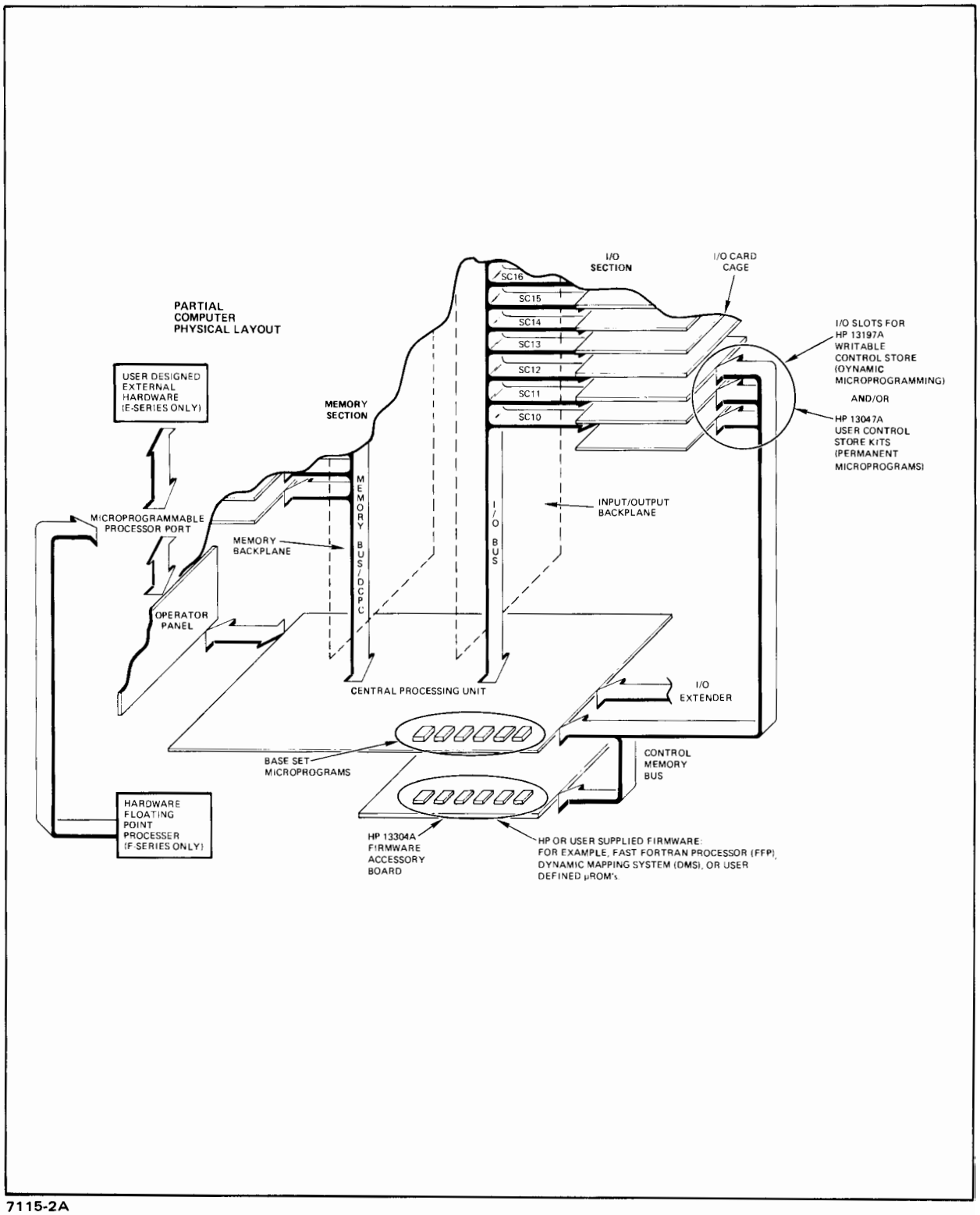
Either the WCS I/O Utility routine WLOAD can be used to load WCS (through a call from FORTRAN, ALGOL, or Assembly language) or the MDE can be used to load WCS. The microprogram can then be called for execution from the main program in the same manner as described for a pROM stored microprogram. To summarize, your microprograms (when loaded) can be executed in the following ways:

- Under MDE control.
- By using an Assembly language UIG instruction.
- Through calls from FORTRAN or ALGOL.

Now that you have an overview of the microprogramming process, let's look at some microprogramming products.

1-5. SOME MICROPROGRAMMING RELATED PRODUCTS

Several different products have been mentioned in the previous paragraphs that are directly associated with the microprogramming environment. Figure 1-2 illustrates products that can be used for microprogramming your HP 1000 E- or F-Series Computer.



7115-2A

Figure 1-2. Some Microprogramming Products

1-6. SUMMARY

To effectively create a microprogram, the programmer must be equipped with the following:

- An understanding of what to microprogram.
- An understanding of the computer operation and its architecture.
- Knowledge of the methods used to map to and access control memory.
- Knowledge of the microassembly language and microinstruction field effects.
- Knowledge of the appropriate microprogramming hardware and software products.

One way to obtain this information is to attend the Hewlett-Packard Computer Microprogramming course. The above subjects are all expanded upon in the remaining portions of this manual but remember that *the most important step* you must take first is to find out *what you should microprogram.*

Section 2
CONTROLLABLE FUNCTIONS

CONTROLLABLE FUNCTIONS

SECTION 2

Now that the "busy areas" of the program have been identified, you are ready to gain some detailed knowledge of the computer that is needed before you read information about the microprogramming language. The following paragraphs describe:

- The hardware functions controlled by microinstructions.
- Aspects of the base set microprogrammed operation that will be important for your microprogramming.
- Enough about Hewlett-Packard products to enable you to take advantage of them (and interface with them) in your own microprogramming.

To implement your own microprograms you will not need to know the computer design to the "gate" level. The information in this book should be entirely sufficient for your needs. The base set discussion will help you to become aware of the existing microprogram's operation. Below is a look at the overall computer followed by details on the registers and other functions.

2-1. COMPUTER FUNCTIONS THAT CAN BE CONTROLLED

Figure 2-1 illustrates the five major sections in the computer. In order of importance, they are the:

- Control Processor.
- Arithmetic/Logic section.
- Main Memory section.
- Input/Output (I/O) section.
- Operator Panel.

Accessories shown in the overall block diagram that are directly associated with microprogramming are the:

- HP 13197A Writable Control Store (WCS).
- HP 13304A Firmware Accessory Board (FAB).
- HP 13047A User Control Store (UCS) Kit.

The important points about these and other accessories will be covered after a look at the "basic" computer.

2-2. CONTROL PROCESSOR

The Control Processor includes a special control memory (made of ROM, pROM, or WCS), registers, logic, and timing signals required to control all of the other sections of the computer. Notice in figure 2-1 that the base set, FAB, WCS, and UCS are all shown associated with the Control Processor by addressing and microinstruction (bus) lines. The base set (the standard instruction set microprogram) is part of the "basic" computer. The 3.5K microword capacity FAB, 1K microword capacity WCS, and 2K microword capacity UCS are accessories that are extensions of control memory you can use for your microprogramming. WCS also communicates with the I/O section to allow microprograms to be written to and read from main memory. Although some signals for control and loading of WCS are passed through the I/O section, both WCS and UCS are connected by cabling to the rest of control memory in an "OR-tied" fashion so that when executing there is no difference in addressing and microinstruction output. No matter how control memory is physically implemented, it all appears as one large microprogram facility to the Control Processor.

2-3. ARITHMETIC/LOGIC SECTION

The Arithmetic/Logic section of the computer includes most of the hardware required to actually carry out the commands of the microinstructions. It contains all working registers in the Central Processing Unit (CPU) and provides the logic to perform arithmetic and logical operations on data.

NOTE

> The CPU consists of not only the Arithmetic/Logic section but the Control Processor and I/O section. These functions are all physically located on the board called the CPU.

2-4. MAIN MEMORY SECTION

All programs and data reside in the Main Memory section consisting of one controller and a set of semiconductor memory modules with which it is designed to operate. The instructions from main memory are all decoded by the Control Processor.

2-5. INPUT/OUTPUT SECTION

The Input/Output (I/O) section serves as an interface between the computer and external devices. The I/O hardware responds either to Control Processor stimuli (for computer-initiated data or control operations) or to device stimuli (for device-signaling attention requests), and hence becomes the active communication link between the computer and peripheral devices.

2-6. OPERATOR PANEL

This is the basic interface between you and the computer. The panel has two registers, several indicators, and many control switches (described in your *Computer Operating and Reference Manual.* The Operator Panel is controlled by base set microroutines. The Operator Panel is also used to route data and command signals through the Microprogrammable Processor Port (MPP) for user designed hardware in E-Series Computers and for the Hardware Floating Point Processor in F-Series Computers.

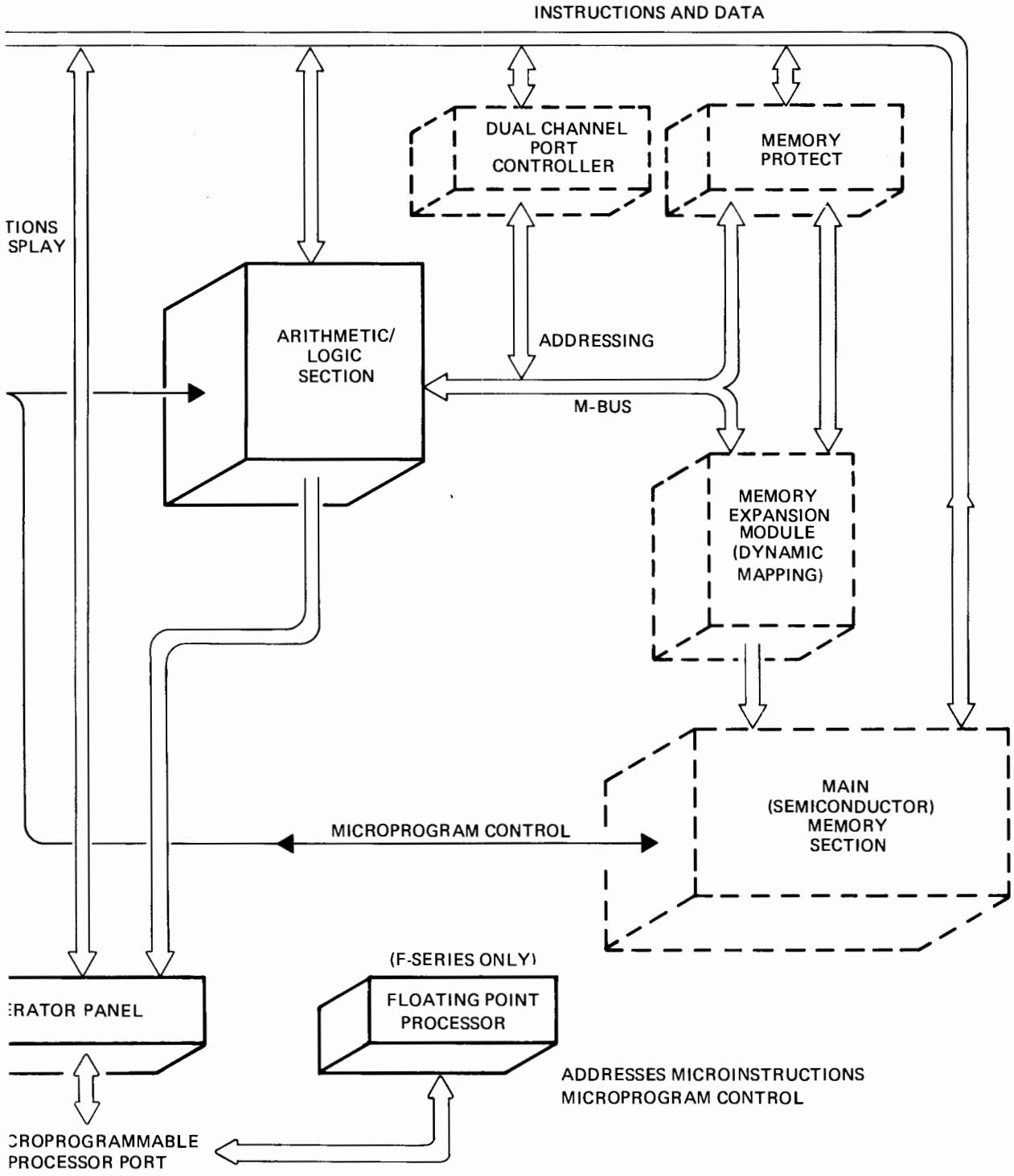
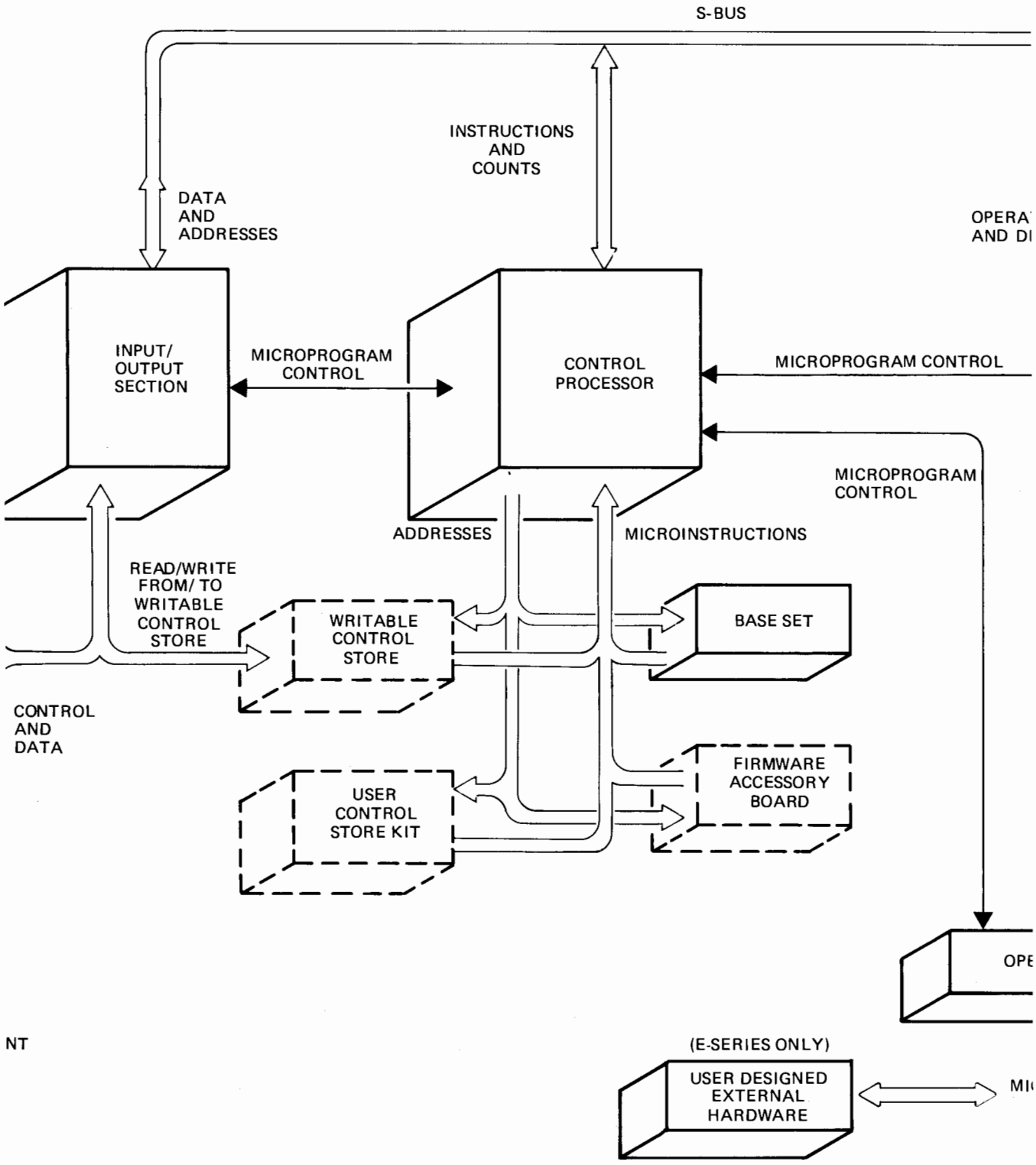
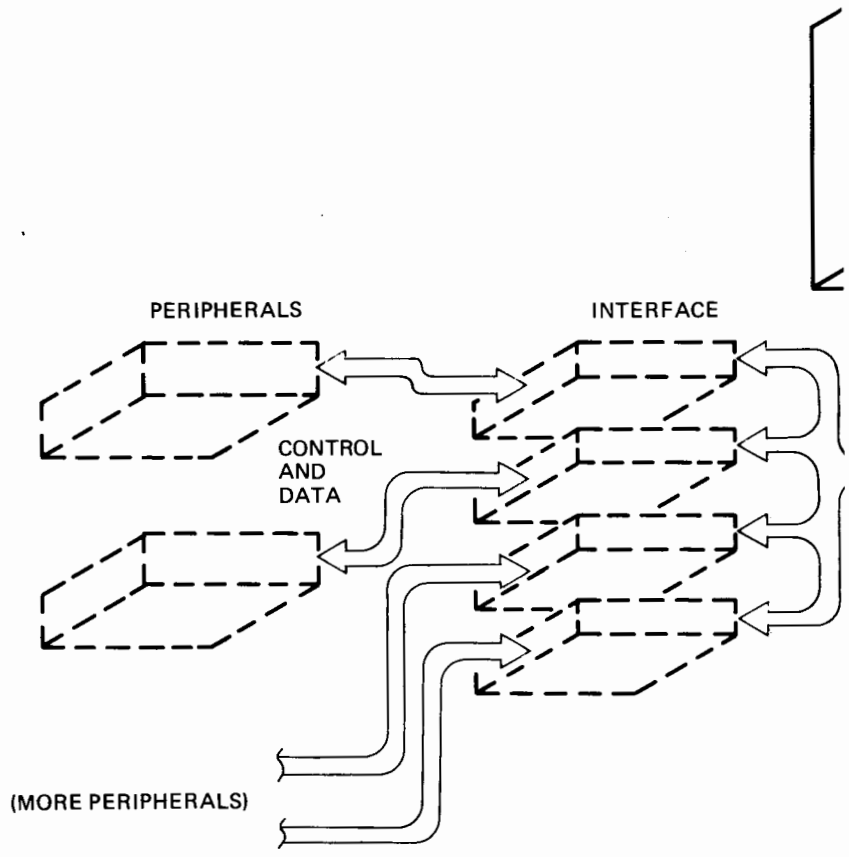


Figure 2-1. HP 1000 E- and F-Series Computer Overall Block Diagram


S-BUS
INSTRUCTIONS AND COUNTS
DATA AND ADDRESSES
OPERA
AND DI
INPUT/ OUTPUT SECTION
MICROPROGRAM CONTROL
CONTROL PROCESSOR
MICROPROGRAM CONTROL
MICROPROGRAM CONTROL
ADDRESSES
MICROINSTRUCTIONS
READ/WRITE FROM/ TO WRITABLE CONTROL STORE
WRITABLE CONTROL STORE
BASE SET
CONTROL AND DATA
FIRMWARE ACCESSORY BOARD
USER CONTROL STORE KIT
OPE
NT
(E-SERIES ONLY)
USER DESIGNED EXTERNAL HARDWARE
MI



NOTE:

DASHED OUTLINES (_ _ _ _) INDICATE EQUIPME
NOT SUPPLIED WITH THE STANDARD COMPUTER.

7115-3A

2-7. MEMORY PROTECT

Memory Protect may interrupt, retain, and report the logical 15-bit address of any instruction that attempts to enter or alter main memory below a programmable fence, execute certain I/O instructions, or execute certain instructions flagged by the Dynamic Mapping System. This accessory will also capture the location of any memory location that may have a parity error. Several circumstances that affect microprogramming in relation to Memory Protect are discussed in part II of this manual.

2-8. DYNAMIC MAPPING SYSTEM

The Memory Expansion Module (MEM) shown in figure 2-1 is part of the HP 13305A Dynamic Mapping System. If installed, the MEM resides (logically) in front of the memory controller and expands the amount of addressable main memory beyond 32K words. The system "windows" a large physical memory down to a logical address space of 32K words. The technique of relating a large physical memory to a logical 32K memory is called "mapping". Since the "maps" involved may be dynamically reloaded, accessibility to the entire physical memory is accomplished. Microprogramming techniques related to the Dynamic Mapping System are discussed in part II of this manual. Note that when the MEM is absent, the M-bus lines are connected directly to main memory.

2-9. DUAL CHANNEL PORT CONTROLLER

The DCPC provides two data paths, software assignable, between main memory and a peripheral device (or devices). High-speed transfers are accomplished in blocks of up to 32K words on an I/O cycle-stealing basis programmatically transparent to the CPU. DCPC microprogramming considerations are also covered in part II of this manual.

2-10. A CLOSER LOOK AT THE FUNCTIONS

In the following paragraphs the computer will be discussed at the level you'll be using to microprogram. Table 2-1 provides you with more detail on functions that can be controlled by microinstructions (and other selected functions) and briefly describes the bus system. You should refer to the detailed block diagram in appendix H when reviewing the table. Once you understand the computer architecture and the effect of micro-orders, you will need only the detailed block diagram and micro-order charts to write microprograms.

Table 2-1. Computer Functions

FUNCTION	DESCRIPTION
CONTROL PROCESSOR	
Instruction Register (IR)	The Instruction Register (IR) is a 16-bit register that usually contains the Assembly (machine) language instructions for execution. (The lower 8 bits of the IR form the counter.)
Control Memory (CM)	Control Memory (CM) receives a 14-bit address from the Control Memory Address Register (CMAR) and offers the corresponding 24-bit microinstruction word to the Microinstruction Register (MIR).
Jump Tables	This ROM is used to map to a CM address from bits contained in the IR.
Microjump Logic (MJL)	The Microjump Logic (MJL) anticipates if and how the Control Memory Address Register (CMAR) will be loaded for a branch.
Control Memory Address Register (CMAR)	The Control Memory Address Register (CMAR) is a 14-bit register that addresses CM. Addressing will progress sequentially (the CMAR is incremented at the beginning of every microcycle) unless a branch or repeat is to occur.
Save Stack	This is a three-level microsubroutine save register. The 14-bit CMAR address is "pushed" onto the stack at the beginning of every microsubroutine branch (JSB). It is "popped" (with the contents loaded into the CMAR) when a microsubroutine return (RTN) is executed. NOTE "Pushing" the Save Stack means placing the return address (the address currently in the CMAR) into the Save Stack. "Popping" the stack means placing the return address into the CMAR and removing it from the Save Stack.
Microinstruction Register (MIR)	The Microinstruction Register (MIR) contains the "current" microinstruction (received from CM).
Field Decoders	Timing and control lines are merged with the field decoders to direct the rest of the computer to execute the microinstruction in the MIR.
ARITHMETIC/LOGIC SECTION	
Arithmetic/Logic Unit (ALU)	The Arithmetic/Logic Unit (ALU) implements all arithmetic and logical operations in the CPU under direction of the Control Processor.
L-Register	The L-register provides the second operand for the ALU.
Rotate/Shifter (R/S)	This function performs left and right shifts and rotates.
Overflow and Extend Registers	These are one-bit registers that participate in ALU and shift/rotate operations.
Conditional Flags	Testable conditional flags associated with the ALU and R/S functions include: ALU Bit 0 Set ALU Bit 15 Set ALU Carry Out ALU Ones ALU Zero CPU Flag

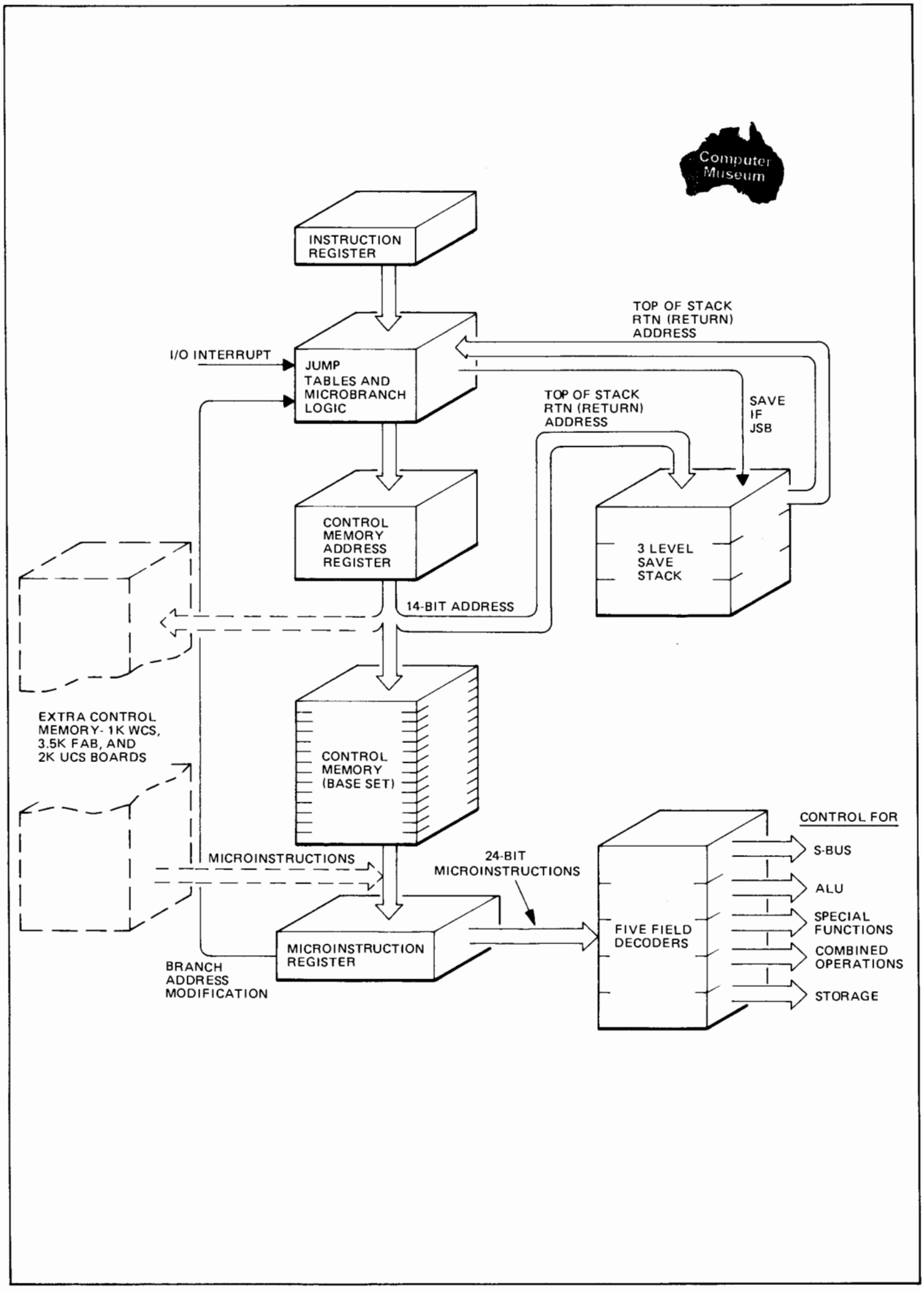
Table 2-1. Computer Functions (Continued)

FUNCTION	DESCRIPTION
ARITHMETIC/LOGIC SECTION (Continued)	
A- and B-Registers	These are the main 16-bit accumulators used for arithmetic, logic, and I/O operations.
RAM Registers	This block of sixteen 16-bit registers is a Random Access Memory (RAM) used for data manipulation and temporary storage of intermediate results. The RAM includes Scratch Registers (S1 through S11), a Stack Pointer register (SP), Index registers (X and Y), the Program Counter (P), and S-register (S).
Loaders	The CPU includes a standard paper tape loader ROM and a standard disc loader ROM. Also included is space for two optional loader ROM's. Each loader can contain up to sixty-four 16-bit instructions. The Remote Program Load (RPL) configuration switches are associated with the loader ROM's.
M-Register	The 15-bit M-register holds the logical address of any computer main memory reference. This 15-bit register is loaded from the S-bus and drives the M-bus. The A-Addressable Flip-Flop (AAF) and B-Addressable Flip Flop (BAF) functions are also controlled by the M-register.
A-Addressable Flip-Flop (AAF) and B-Addressable Flip-Flop (BAF)	These flags determine whether the A-, or B-, or T-register will be used for storing data or directing data to the S-bus. They exist because the A- and B-registers can be addressed as main memory locations 0 and 1, respectively. AAF or BAF is set or cleared depending upon the M-bus data.
MAIN MEMORY SECTION	
Memory Address Register	This register receives the "physical" main memory address from the M-bus for a read or write operation. An address must be present here before the read or write begins. Data is transferred from/to this address on the selected memory module board from/to the T-register.
T-Register	The T-register is the 16-bit data link between the Main Memory section and the CPU or DCPC. Data comes from or goes to the address specified in the Memory Address Register.
INPUT/OUTPUT (I/O) SECTION	
I/O Control and Select Logic	I/O timing and signal generation take place from this function. The interface control signals are generated as a result of the Control Processor executing I/O instructions.
Interrupt Control	Interrupts from devices requesting input or output transfers with the CPU are sequenced for processing by priority logic in this function.
Central Interrupt Register (CIR)	This 6-bit register is loaded with the select code (address) of the interrupting device after an interrupt request is recognized. The CIR passes this address to the S-bus under microprogram control.

Table 2-1. Computer Functions (Continued)

FUNCTION	DESCRIPTION
OPERATOR PANEL	
Display Register (DSPL)	The Display Register is the 16-bit Operator Panel register associated with the panel switches.
Display Indicator (DSPI)	This Operator Panel register indicates which register is being displayed by the DSPL register.
BUS SYSTEM	
S-bus	This is the main 16-bit data transfer bus in the computer. (See the block diagram and note the functions that have two-way and one-way transfer capability.)
T-bus	This is the 16-bit resultant data bus in the Arithmetic/Logic section.
M-bus	This is a 15-bit memory address bus used by both the CPU and the DCPC.
I/O bus	This is a 16-bit bus for data transfers, or for control and status exchanges to and from external devices.
Select Code (SC) bus	This 6-bit bus carries the select code of a device being referenced by the I/O section or DCPC.
Interrupt Address (I/A) bus	This 6-bit bus carries the address (select code) of any I/O device requesting CPU service.

Figure 2-2 is a simplified block diagram of the Control Processor. In a "conventional" computer control section, specific hardware is dedicated to each function performed by the instruction set. The major advantage of the "conventional" control section is speed for the instruction set. The major disadvantage is the loss of flexibility for special applications or for enhancements. In the microprogrammed computer, all distinct logical functions are separated from the sequence in which those functions are performed. That is, the logical functions are defined by microinstructions (composed of micro-orders) held in control memory. Because functions can be individually defined by microinstructions, the microprogrammed computer is much more flexible than the "conventional" type computer. At one time this caused the microprogrammed computer to be slower in executing some portions of the instruction set. However, the Computer Control Processor executes microinstructions at a rate that is fast enough to keep main memory busy practically all the time so, the speed penalty for using the microprogrammed architecture is essentially not a factor, especially in the base set. Also, since the Control Processor in the E-Series and F-Series Computers is completely microprogrammable, user programs can be made to execute much faster with the application of user microprogramming. These combined factors provide this computer with the final advantage over any conventional control section (hardwired component) type of computer.



7115-4

Figure 2-2. Simplified Control Processor Block Diagram

2-11. SOME DEFINITIONS AND TIMING POINTS

Now to clarify some definitions about control and timing, and then discuss a little more about the computer's interrelated functions and operation.

- The Control Processor executes "microcoded" "microinstructions" during "microcycles".
- One microcycle (also called a "T" period) is the time interval required to completely execute a microinstruction.
- A microinstruction is a 24-bit coded word (code definition is called the microcode) that defines specific hardware operations to be performed by the computer.
- Each microinstruction is composed of at least one, and up to five micro-orders. Each micro-order defines a specific operation to be performed in the computer. Some micro-orders accomplish multiple operations by themselves.
- Microinstructions physically reside in control memory and are the basic building blocks of microprograms.
- Segments of microprograms may be called microroutines.
- A portion of microcode called from a microroutine will be referred to as a microsubroutine.

Part II of this manual provides specific information on timing that you will need for microprogramming.

2-12. HOW DO ALL THESE FUNCTIONS INTERRELATE?

All the functions described in the preceding paragraphs are interrelated in an operational sense through the microprogrammed operation of the computer. Here are a few points to remember:

- The computer is always under microprogram control and executing microinstructions at all times when power is applied, (except when temporarily suspended by DCPC or main memory contentions).
- A microroutine in the base set reads ("fetches") Assembly language instructions stored in main memory. The instructions are loaded into the IR and data is directed to the appropriate destinations by the microprogram invoked.
- Each Assembly language *instruction* from main memory is interpreted as a "pointer" (address) to a microroutine, resident in control memory, that implements the instruction by executing a sequence of microinstructions.

A few other points should be considered before examining what control memory can accomplish:

- The Control Processor decodes each microinstruction into fields, then executes the indicated micro-orders in the proper sequence.
- Each micro-order performs a distinct operation and the micro-orders are not necessarily related to each other in each microinstruction.

Keep the above points in mind as you read through the following steps of how "generally" the Control Processor might operate in a microroutine:

- The "standard" microinstruction (in the MIR) typically calls for the contents of a register to be enabled onto the S-bus. Then certain ALU and/or rotate/shift operations take place during the microcycle and, at the end of the microcycle, a specified destination register is "clocked" to receive the prevailing data from its input lines.

- While a microinstruction presently in the MIR is being executed, the CMAR is incremented to present the next sequential address to CM *or* the MJL determines another address to load the CMAR.

- If a microbranch to a microsubroutine is executed, the incremented address is loaded into the Save Stack and the branch address is loaded into the CMAR.

- Several branch-on tests exist (e.g., conditions of carry, the sign, a zero result, presence of a particular bit or Operator Panel setting, etc.) that provide branches to microroutines designed to react to the condition.

- When a microprogram completes, it usually returns to control memory location 0 (addresses in octal are five digits, i.e., 00000) to complete fetching (obtaining) the next Assembly language instruction to be executed from main memory.

You should not be concerned if the details of Control Processor and microprogram operation are not clear at present. You will gain more knowledge and understanding of the computer operation as you learn the microprogramming language by progressing through the manual and writing microprograms. Some further points:

- If the microprogram execution time exceeds the interval between pending interrupts allowed by your particular system application, the interrupts can be lost. Your microprogram must be written to test for pending interrupts.

- When a pending interrupt is detected, the microprogram must yield control to the Halt-Or-Interrupt (HORI) microroutine (CM location 6 in the base set).

Microprogrammed interrupt handling techniques will be fully described in section 7. Now, what about control memory content?

2-13. CONTROL MEMORY

Roughly, you can look at control memory as being devoted to serving three areas:

- The standard base set.
- HP microprogrammed accessories.
- The user's microprogramming area.

All 16,384 addressable (24-bit) words of control memory are logically partitioned into sixty-four 256-word modules numbered 0 through 63. Figures 2-3 and 2-4 show the control memory map (represented in basic 1K separations) and identifies the "modules" mentioned above. Notice that modules 0 through 3 are dedicated to the standard base set shipped with every computer. The other 60 modules are available for additional microprograms written by you or supplied by Hewlett-Packard.

CONTROL MEMORY MODULE ALLOCATION	MODULE NO.	ADDRESS DECIMAL	ADDRESS OCTAL	SOFTWARE ENTRY POINT	
HP BASE SET	0	0-00255	00000-00377	YES	1K
	1	00256-00511	00400-00777	YES	
	2	00512-00767	01000-01377	YES	
	3	00768-01023	01400-01777	YES	
AVAILABLE FOR USER MICROPROGRAMMING	4	01024-01279	02000-02377	NO	2K
	5	01280-01535	02400-02777	NO	
	6	01536-01761	03000-03377	NO	
	7	01762-02047	03400-03777	NO	
	8	02048-02303	04000-04377	NO	3K
	9	02304-02559	04400-04777	NO	
	10	02560-02815	05000-05377	NO	
	11	02816-03071	05400-05777	NO	
	12	03072-03327	06000-06377	NO	4K
	13	03328-03583	06400-06777	NO	
	14	03584-03849	07000-07377	NO	
	15	03850-04095	07400-07777	NO	
	16	04096-04351	10000-10377	NO	5K
	17	04352-04607	10400-10777	NO	
	18	04608-04863	11000-11377	NO	
	19	04864-05119	11400-11777	NO	
	20	05120-05375	12000-12377	NO	6K
	21	05376-05631	12400-12777	NO	
	22	05632-05887	13000-13377	NO	
	23	05888-06143	13400-13777	NO	
	24	06144-06399	14000-14377	NO	7K
	25	06400-06655	14400-14777	NO	
	26	06656-06911	15000-15377	NO	
	27	06912-07167	15400-15777	NO	
	28	07168-07423	16000-16377	NO	8K
	29	07424-07679	16400-16777	NO	
HP DYNAMIC MAPPING SYSTEM	30	07680-07935	17000-17377	NO	
	31	07936-08191	17400-17777	NO	
	32	08192-08447	20000-20377	YES	9K
HP FAST FORTRAN PROCESSOR	33	08448-08703	20400-20777	NO	
	34	08704-08959	21000-21377	YES	
	35	08960-09215	21400-21777	YES	
EXTENDED MEMORY AREA DS/1000	36	09216-09571	22000-22377	YES	10K
	37	09572-09727	22400-22777	YES	
	38	09728-09983	23000-23377	YES	
	39	09984-10239	23400-23777	YES	
HP RESERVED	40	10240-10495	24000-24377	YES	11K
	41	10496-10751	24400-24777	NO	
	42	10752-10917	25000-25377	NO	
	43	10918-11263	25400-25777	NO	
	44	11264-11519	26000-26377	YES	12K
	45	11520-11775	26400-26777	YES	
RECOMMENDED FOR USER MICROPROGRAMMING	46	11776-12031	27000-27377	YES	
	47	12032-12287	27400-27777	YES	
	48	12288-12543	30000-30377	YES	13K
	49	12544-12799	30400-30777	YES	
	50	12800-13055	31000-31377	YES	
	51	13056-13311	31400-31777	NO	
	52	13312-13557	32000-32377	NO	14K
	53	13558-13823	32400-32777	NO	
	54	13824-14079	33000-33377	NO	
	55	14080-14335	33400-33777	NO	
	56	14336-14591	34000-34377	YES	15K
	57	14592-14847	34400-34777	YES	
	58	14848-15103	35000-35377	YES	
	59	15104-15359	35400-35777	YES	
	60	15360-15615	36000-36377	YES	16K
	61	15616-15871	36400-36777	NO	
	62	15872-16127	37000-37377	YES	
	63	16128-16383	37400-37777	NO	

7115-5A

Figure 2-3. E-Series Control Memory Map

CONTROL MEMORY MODULE ALLOCATION	MODULE NO.	ADDRESS DECIMAL	ADDRESS OCTAL	SOFTWARE ENTRY POINT	
HP BASE SET	0	0-00255	00000-00377	YES	1K
	1	00256-00511	00400-00777	YES	
	2	00512-00767	01000-01377	YES	
	3	00768-01023	01400-01777	YES	
HP RESERVED	4	01024-01279	02000-02377	YES	2K
	5	01280-01535	02400-02777	NO	
	6	01536-01761	03000-03377	YES	
	7	01762-02047	03400-03777	NO	
	8	02048-02303	04000-04377	YES	3K
	9	02304-02559	04400-04777	NO	
	10	02560-02815	05000-05377	NO	
	11	02816-03071	05400-05777	NO	
	12	03072-03327	06000-06377	YES	4K
	13	03328-03583	06400-06777	NO	
	14	03584-03849	07000-07377	NO	
	15	03850-04095	07400-07777	NO	
	16	04096-04351	10000-10377	YES	5K
	17	04352-04607	10400-10777	NO	
	18	04608-04863	11000-11377	NO	
	19	04864-05119	11400-11777	NO	
	20	05120-05375	12000-12377	NO	6K
	21	05376-05631	12400-12777	NO	
	22	05632-05887	13000-13377	NO	
	23	05888-06143	13400-13777	NO	
	24	06144-06399	14000-14377	NO	7K
	25	06400-06655	14400-14777	NO	
	26	06656-06911	15000-15377	NO	
	27	06912-07167	15400-15777	NO	
AVAILABLE FOR USER MICROPROGRAMMING	28	07168-07423	16000-16377	NO	8K
	29	07424-07679	16400-16777	NO	
	30	07680-07935	17000-17377	NO	
	31	07936-08191	17400-17777	NO	
HP DYNAMIC MAPPING SYSTEM →	32	08192-08447	20000-20377	YES	9K
HP FAST FORTRAN PROCESSOR	33	08448-08703	20400-20777	NO	
	34	08704-08959	21000-21377	YES	
	35	08960-09215	21400-21777	YES	
EXTENDED MEMORY AREA	36	09216-09571	22000-22377	YES	10K
	37	09572-09727	22400-22777	NO	
DS/1000	38	09728-09983	23000-23377	YES	
	39	09984-10239	23400-23777	NO	
SCIENTIFIC INSTRUCTION SET	40	10240-10495	24000-24377	YES	11K
	41	10496-10751	24400-24777	NO	
	42	10752-10917	25000-25377	YES	
	43	10918-11263	25400-25777	NO	
HP RESERVED	44	11264-11519	26000-26377	NO	12K
	45	11520-11775	26400-26777	NO	
RECOMMENDED FOR USER MICROPROGRAMMING	46	11776-12031	27000-27377	YES	
	47	12032-12287	27400-27777	YES	
	48	12288-12543	30000-30377	YES	13K
	49	12544-12799	30400-30777	YES	
	50	12800-13055	31000-31377	YES	
	51	13056-13311	31400-31777	NO	
	52	13312-13557	32000-32377	NO	14K
	53	13558-13823	32400-32777	NO	
	54	13824-14079	33000-33377	NO	
	55	14080-14335	33400-33777	NO	
	56	14336-14591	34000-34377	YES	15K
	57	14592-14847	34400-34777	YES	
	58	14848-15103	35000-35377	YES	
	59	15104-15359	35400-35777	YES	
	60	15360-15615	36000-36377	YES	16K
	61	15616-15871	36400-36777	NO	
	62	15872-16127	37000-37377	YES	
	63	16128-16383	37400-37777	NO	

7115-5B

Figure 2-4. F-Series Control Memory Map

Several modules have already been allocated to established Hewlett-Packard firmware packages which are shown in figure 2-3 for E-Series Computers and figure 2-4 for F-Series Computers. In addition, some modules have been reserved by Hewlett-Packard for potential future enhancements.

The rest of control memory is for user microprogramming and modules 46 through 63 are recommended. Section 6 of this manual describes how you can enter CM (through the software entry points shown in the map) by using Assembly language User Instruction Group (UIG) instructions.

NOTE

With the exception of modules 0 through 3 (base set instructions), there is no restriction on which modules you may use (see figure 2-3) to implement your microprograms. However, Hewlett-Packard may also use other modules (in addition to those already reserved) for future firmware accessories.

2-14. LET'S TALK ABOUT THE BASE SET

The complete base set listing, including the Jump Tables, is shown in appendix G for E-Series Computers. For F-Series Computers modules 0, 1, and 2 are the same except for the jump tables and these differences are provided in appendix G. Module 3 in F-Series Computers is used by the Hardware Floating Point Processor. There isn't a great amount of detail about the base set here because:

- You're probably not yet familiar with all the micro-orders and word types.
- The overall microprogram sequence of operation actually depends upon the sequence of Assembly language instructions fetched from main memory.
- It's assumed that you're primarily interested in doing your own microprogramming.

You will, however, be referring occasionally to the base set for examples of microprogramming techniques that you may want to use in your own microprograms. (You'll also find plenty of applications type examples in parts II through IV.) Also, you will want to have a basic understanding of how certain microroutines of the base set can act as utility microroutines for your microprograms.

The base set microprogram provides the capability to execute all the basic Assembly language instructions described in your *Computer Operating and Reference Manual*. In modules 0 and 1 of the base set are:

- Microroutines to execute instructions in the
  - Memory Reference Group.
  - Alter-Skip Group.
  - Shift-Rotate Group.
  - Input/Output Group.
  - Extended Arithmetic Group.

- Microroutines that
  - Control the Operator Panel.
  - Load the Initial Binary Loader (from the selected Loader ROM).
  - Execute the built-in firmware diagnostics.
  - Handle interrupts.
  - Fetch indirect operands.

Also in the base set, modules 2 and 3 contain:

- Microroutines for all the instructions in the Extended Instruction Group (EIG).
- Microroutines to execute all the Floating Point instructions.

The Jump Tables (shown in the block diagram, appendix H) map the data in the IR to the appropriate location in CM to initiate instruction execution.

Some "typical" operations performed by the base set microprogram include:

- A power-up sequence.
- A "short form" diagnostic check of the CPU and main memory.
- An initial binary loading sequence.
- Operator Panel sequences such as scanning the pushbuttons by making conditional tests and updating the DSPI and DSPL registers.
- Performing a read (fetch) operation to execute an instruction (e.g., Memory Reference Group, Floating Point, etc.), then fetching the data to perform an ALU operation, and finally storing in a register.
- Performing a write operation (e.g., an ISZ instruction).
- Performing I/O operations (e.g., CPU-initiated transfers, or device-initiated transfers of data with Halt-Or-Interrupt microroutine transitions).
- Reading UIG instructions from main memory that map to the "user" microprogramming area in control memory.

The timing relationships involved in operations such as the above mentioned will be discussed in sections 5 and 7. Now, a brief look at how two of these operations are carried out by the base set.

2-15. AN OPERATIONAL OVERVIEW

The base set microprogram (with computer timing) accomplishes the tasks that, in the past, were performed by "hardwired" portions of the computer control section. The following discussion provides an overview of how the Computer Control Processor performs several operations in parallel in the base set. The microroutines for the Assembly language XOR and ADA instructions are used as examples in

this discussion to illustrate several techniques that you should be aware of to effectively execute your own microprograms. You may find it helpful to look again at the detailed block diagram in appendix H.

2-16. FETCHING. "Fetching" (as briefly defined in paragraph 2-12) means obtaining the "next" instruction to be executed from main memory. In this computer, a "look-ahead" technique is used for this process. That is, fetching is *begun* while simultaneously completing the execution of the "current" instruction; fetching is *completed* while preparing for execution of this "next" instruction. Usually this is accomplished by starting a read operation (of the main memory address contained in the M-register) just prior to termination of the "currently" executing instruction microroutine.

For illustrative purposes, suppose that the "currently" executing microroutine is for an XOR instruction (that had been obtained from main memory location 2000). The M-register has already been incremented so that as the microroutine for XOR is completing its execution, the read that is initiated is for main memory location 2001. (Assume that with the completion of the XOR execution, an augend is left in the A-register and that at main memory location 2001 there is an Assembly language ADA instruction.)

Upon termination of this "current" Assembly language instruction's microroutine, control passes to a Fetch microroutine at the beginning of the base set which completes the read operation by storing the instruction read from main memory into the IR. In this manner of "look-ahead" reading, the overhead required for instruction fetching is minimized. Your user microprograms must be designed to terminate in a similar manner and you will see specifically how to do this from information you will read in section 7.

To continue, in the Fetch microroutine, in addition to completing the read operation by storing the main memory instruction in the IR, an operand address is always formed in the M-register and another read operation is started immediately. This is in anticipation that the instruction stored in the IR is of the Memory Reference Group. If later it is determined that the instruction is of a different type, the information arriving in the T-register will not be used.

In the example being used, an ADA instruction from main memory location 2001 has been stored in the IR and an operand address (assume the address is 300) has been formed in the M-register. So the read operation initiated at the beginning of the Fetch microroutine is obtaining the operand (the addend) for the ADA instruction from main memory location 300 but the information has not yet arrived in the T-register.

Next (still in the base set Fetch microroutine), the P- and M-registers are adjusted. During normal execution P and M are always two and one (respectively) ahead of the current instruction's address (the instruction that is executing). After the read operation is initiated (to obtain the operand), the P-register content is stored in M and P is then incremented.

In the example being used, recall that before the operand address (300) was formed in the M-register it contained address 2001 (the address of the ADA instruction) and the P-register (if the rules stated above are followed) contained 2002. Now the content of P is put on the S-bus, stored in M and incremented through the ALU and stored back in the P-register. Thus, M is now adjusted to 2002 and P is adjusted to 2003 in preparation for the read operation that will be initiated as the microroutine for the ADA instruction (from main memory location 2001) is being executed.

You can see from the above example that you are now prepared to read the next sequential instruction from main memory with the P-register one ahead of M and two ahead of the instruction being executed (preparation to execute the example ADA instruction is being made as will be explained in the next paragraph). When you study the micro-orders and word types in part II you will see that, for proper operation, the situation for P and M (just described) will also have to exist for your own microprograms.

Finally in the Fetch microroutine, the Instruction Register (IR) bits are examined to determine the instruction type. That is, the upper eight bits of the IR are examined to determine where in control memory to branch to execute the "current" instruction. This branch can be in the base set (as it is in the example being used), or within the User's area, or within the Hewlett-Packard microprogrammed accessories area. Decoding via the Jump Tables (CM mapping) forces Control Processor operation to the appropriate CM address to implement the instruction contained in the IR.

In the ADA instruction example being used, the special purpose base set micro-orders used cause the upper eight bits of the IR to be applied as an address to the Jump Tables (ROM's) which store the ADA instruction's microroutine address into the MJL. The MJL stores this address into the CMAR which reads the first microinstruction for the ADA microroutine into the MIR. Simultaneously, the special purpose base set micro-orders enable the interrupt logic and initialize the Save Stack. This is all done to facilitate branches to microsubroutines which can be made to three levels. This completes the fetch process. When the appropriate CM address has been reached, "execute" begins.

2-17. EXECUTION. Execution of the Assembly language instruction is carried out by the specific micro-orders contained in the individual microinstructions of the appropriate microroutines as they are decoded from the MIR.

Again, using the ADA instruction as an example, the first of the two microinstructions for ADA immediately begins a read operation from the main memory address (2002) in the M-register (in the "look-ahead" manner previously described) to obtain the next Assembly language instruction. But, how do you get the addend from main memory to add to the A-register? Recall that the Fetch microroutine has already begun a read operation. This read operation gets the ADA operand (addend) from main memory (via the T-register), places it on the S-bus, routes it "as is" through the ALU, and stores it in the L-register. So, for Memory Reference Group instructions, the read operation started in the Fetch microroutine will be used to obtain operands by storing the T-register data in the desired register.

The last action in the execution of the example ADA instruction occurs as the CMAR increments to the next CM location (in a branching type microinstruction, other actions can occur) and CM loads the MIR with the next microinstruction. Through action of the field decoders, the A-register content is gated onto the S-bus and routed through the ALU with an "add" function enabled. This causes the S-bus content (the augend from the A-register) to be added to the content of the L-register (the addend). The microinstruction simultaneously enables a test for an overflow or carry-out condition then stores the resultant data back in the A-register. In addition, this second microinstruction forces a return of Control Processor operation to control memory location 0 to complete another main memory fetch and prepare for another execution operation. (Remember that the read operation had been started in a similar manner for the ADA instruction. You can see that a considerable amount of work can be done with a single microinstruction.

To summarize, the main points that you should remember from the above discussion are that:

- A read operation begins in a "look-ahead" manner while the execution of the previous instruction is carried out. Once a branch to your microprogram is made (by decoding a UIG type instruction), it is possible for you to stay in the user microprogramming area until it is desired to return to the fetch microroutine. Before returning, however, you should terminate your microprogram properly.

- Some other considerations also exist for write operations and these will be discussed in section 7.

- In regard to staying in your microprogram as long as desired (as mentioned previously in this section), there is a danger of lost interrupts if you stay too long. These considerations should be taken into account when you design your microprogram.

- The base set fetch microroutine acts as a utility microroutine for the main memory instruction fetch and execute preparation. It also takes care of the P- and M-register adjustments. You should make use of this microroutine in designing your microprograms. Also, in regard to interrupts, the base set Halt-Or-Interrupt microroutine can be used as another microprogramming aid to handle interrupts in your microprograms.

Interrupt examples were not included in the operational overview just presented; interrupts are covered in part II of this manual.

2-18. MICROPROGRAMMED ACCESSORIES

In paragraph 2-13 you found that a few modules have already been reserved for Hewlett-Packard microprogrammed accessories. Remember that all accessories for the computer do *not* require additional microprograms but if they do, the microprograms will *generally* be supplied as pROM's to be mounted on the FAB or on another CM extension (e.g., 2K UCS). Some accessories requiring microprograms may be supplied in a form that will require writing the microprogram to WCS before the instructions involved can be executed. DCPC and Memory Protect do not require additional microprograms. The mapping facility for all Hewlett-Packard microprogrammed accessories is in the base set. For further information on accessories, see the appropriate manuals. Other microprogramming features such as, the Microprogrammable Processor Port (MPP), Hardware Floating Point Processor (FPP), and the block I/O transfer feature of the Computer are described in section 13.

2-19. SUMMARY

Sections 1 and 2 of part I have provided you with the following:

- Reasons for microprogramming.

- An awareness of what to microprogram.

- An overall look at the microprogramming procedure.

- A complete look at the computer hardware controlled by microprograms.

- Introductory information on some Hewlett-Packard accessories directly and indirectly associated with microprogramming.

- An overview of control memory identifying the user's area.

- A brief look at some base set operations.

In part II you will learn the microprogramming language and methods for microprogramming up through preparation with the microassembler.

PART II
Microprogramming Methods

Section 3
MICROPROGRAMMING PREPARATION STEPS

MICROPROGRAMMING PREPARATION STEPS

SECTION 3

Assuming that you have analyzed your programming environment (as suggested in section 1) and have decided to microprogram a portion of your program(s), there are certain steps necessary to prepare your RTE operating system to accept the microprogramming environment. These are not precisely the same steps to preparation as shown in figure 1-1 (Microprogramming Implementation Process), but deal with the "background" situation. That is, as you can surmise from a review of part I, a certain hardware/software situation must be made to exist in the RTE system which includes:

- Installation of some additional control memory "hardware" for storage of the additional microprograms (above those used in the base set). Normally this extra control memory must also be in addition to that which you may have for microprogrammed accessories (such as DMS).

- Installation of microprogramming support software for microprogram development. It must be realized that, as outlined in part I, it is not necessary to have "extra" software for microprogramming once your microprogram has been "installed" in control memory (CM). The "extra" software is necessary for development and, when WCS is used for the added CM, a driver and utility routine are needed for dynamic loading of CM before microprogram execution.

This section outlines the RTE environment and the necessary hardware and microprogramming support software installation steps.

3-1. ENVIRONMENT

The RTE Microprogramming Support Software package (described in paragraph 3-3) operates in the RTE II or IV system environment with a software revision date code of 1631 or later. Therefore, your RTE operating system must basically exist as defined in the *Real-Time Executive IV Software System Programming and Operating Manual,* part no. 92067-90001 or *Real-Time Executive II Software System Programming and Operating Manual*, part no. 92001-93001.

Microprogramming hardware that is to be added (outlined in paragraph 3-2) must conceptually be installed before system generation. Some microprogramming support software must be installed during system generation and some may be installed just before use. (Section 8 and part III in this manual provide instructions as to when certain programs may be installed other than at system generation time.) Paragraph 3-3 describes system requirements for individual microprogramming support software items.

3-2. MICROPROGRAMMING HARDWARE

The HP 13197A Writable Control Store Kit is the acceptable hardware for microprogram development and it can, of course, be used for "normal operation" of your microprograms. It must be installed before system configuration. Two additional WCS (or UCS) boards may be installed. (The total number of control memory boards that can be installed is dependent upon the computer used.) Control memory boards in the I/O section should be installed starting at SC 10. The operational states, hardware

supplied, and installation guidelines for WCS boards are contained in the *HP 13197A Writable Control Store Reference Manual,* part no. 13197-90005. Additional information on the installation of the driver for WCS follows in paragraph 3-3.

If you are going to install pROM's, the microprograms must be developed, tapes prepared, and the pROM's fused before they can be installed. This means you will have to install WCS (as mentioned above) first, and the required microprogramming software (mentioned in paragraph 3-3) before the pROM's are ready for installation. Then, depending upon whether you select UCS or the FAB, your RTE system will have to be disassembled to a certain extent to install the pROM's.

If you select the HP 13304A Firmware Accessory Board for pROM installation, you will not have to use an I/O slot and reconfigure the RTE system, but you will have to remove the FAB board, install the pROM's, configure jumpers, and reinstall the FAB in the computer under the CPU.

NOTE

With an RTE IV system, the HP 13305A Dynamic Mapping System (DMS) will probably be installed, and control memory module 32 (dynamic mapping instructions) is installed on the FAB. You will therefore already have the FAB and its cable. You may or may not have the FAB with an RTE II system.

NOTE

With an F-Series Computer with RTE IV and DS/1000 the space on the FAB will probably be completely used up by the following HP-supplied microcode:

Dynamic Mapping System
Fast Fortran Processor
Extended Memory area
DS/1000
Scientific Instruction Set

The FAB will then not be available for user microprogramming.

To install pROM's and configure CM address jumpers on the FAB or UCS board, refer to the following documents.

- Your *Computer Series Installation and Service Manual.*
- *HP 13304A Firmware Accessory Board Installation and Service Manual,* part no. 13304-90001.

If you select the HP 13047A User Control Store Kit for your microprogram installation, the pROM's must be prepared then installed on the board following the instructions in the *HP 13047A User Control Store Kit Installation and Service Manual,* part no. 13047-90001. You must then devote an I/O slot (SC 10) in the backplane to UCS and reconfigure the RTE operating system as necessary following instructions in the RTE System Operating Manual. (Refer to paragraph 3-1).

3-3. MICROPROGRAMMING SUPPORT SOFTWARE

In order to develop and run microprograms in a dynamic manner in the RTE operating system environment you will need some, and possibly all, of the *HP 92061 RTE Microprogramming Support Software Package*. The total package is outlined below.

- RTE Microassembler Program
- RTE Microassembler Cross-Reference Generator Program
- RTE Microdebug Editor Program
- RTE Microdebug Editor Subroutine
- RTE Driver DVR36
- WCS I/O Utility Routine WLOAD
- pROM Tape Generator program.

These programs, the driver, and utility routines are described below the applicable part numbers, installation guides, and appropriate references. Note that to receive the microprogramming support software on a magnetic tape cartridge you should specify option 020 for the HP 92061 package.

3-4. THE RTE MICROASSEMBLER

This program converts a source microprogram into binary object code which may be directed to an output device and/or recorded on a disc file. The source may be input from an input device or the system LS area. The object code may be produced in either a standard format recognized by the Microdebug Editor program and the WLOAD routine or a special format for the HP ROM Simulator. The microassembler can also generate a symbol table and listing of source records with the respective octal code. The RTE system name for the program is MICRO. The program object part number of MICRO is 92061-16001. In the RTE system, the microassembler can run with or without the File Manager (FMGR) and requires about 8K words of background. Actually, to use the microassembler purely for microassembling, no additional microprogramming hardware (i.e., WCS) is needed. All information on preparation with the microassembler and on microassembler output is contained in sections 8 and 9 of this manual.

3-5. MICROASSEMBLER CROSS-REFERENCE GENERATOR

The cross-reference generator is used (usually with the microassembler) to generate a cross-reference table of symbols-to-CM addresses. The program can be run using a microassembler parameter list option or separately using its RTE system name MXREF. The program object part number is 92061-16002. More detail on the RTE Microassembler Cross-Reference Generator is contained in section 9 of this manual.

3-6. RTE MICRODEBUG EDITOR

This program allows you to debug and execute microprogram object code. The object code may be input from a paper tape reader or a disc file, or it may be resident in WCS. The Microdebug Editor (MDE) allows you to delete or replace microinstructions, set breakpoints, change registers, and so on. Information on the use of the Microdebug Editor is contained in section 10 of this manual. In the RTE system, the MDE requires about 8K words of background. When the MDE is user scheduled it is

identified by the program name MDEP. When it is called as a utility in the RTE system environment it is identified by the progrtam name MDES. The program object (part number) of the MDE is supplied in two parts: Microdebug Editor Program MDEP, part no. 92061-16004, and subroutine MDES, part no. 92061-16005. The HP 13197A WCS board is used with the MDE, which uses driver DVR36 and WCS I/O Utility subroutine WLOAD for operation.

3-7. DRIVER DVR36

Driver DVR36 must be configured into the RTE system during system generation to provide software linking between the MDE, WLOAD, or Assembly (or FORTRAN) language programs and WCS.

NOTE

> The other microprogramming support software can be included either during system generation or loaded into the system when required.

DVR36 drives the HP 13197A WCS board(s) for reads and writes (from and to main memory) and allows control of WCS board functions. The driver implements some resource protection mechanisms which include ensuring that no two WCS boards are enabled with the same CM address spaces. The driver utilizes DCPC, if so configured, and transfers data at the fastest rate permitted by the DCPC. Non-DCPC transfers will take longer; the driver periodically suspends itself to ensure that interrupts are not held off for too long.

The object part number of the driver is 13197-16001. When configured in the RTE system, the select code (SC) number of the first WCS should be SC 10 because of hardware constraints. (More details on DVR36 appear in section 11 of this manual and the driver manual is referenced in table 3-3.) In the system, the driver can be called directly with an EXEC call, or through the WLOAD routine. Introductory information on WLOAD follows.

3-8. WLOAD

The WCS I/O Utility Routine WLOAD (object part no. 13197-16003) uses DVR36 and transfers microprogram object code into WCS when called by the MDE or by the Assembly (or FORTRAN) language program. Section 11 in this manual and table 3-3 contain more information on WLOAD.

3-9. LOADING THE MICROPROGRAMMING SUPPORT SOFTWARE

The microprogramming support software can be loaded during System Generation or on line, using the RTE LOADR. The exception to this is the driver, DVR36, which must be loaded at System Generation time. (Refer to *RTE Driver DVR36 Programming and Operating Manual,* part no. 13197-90001.) The two subroutines WLOAD and MDES can be included at System Generation so that they will be available when calling programs are loaded on line.

With RTE disc based systems it is possible to load programs into different partitions depending on the program type. Table 3-1 lists the program partitioning capability.

Table 3-1. Program Partitioning Capability

PROGRAM NAME	RTE TYPE	II OR IV			II	IV
	PGM TYPE	1	2	3	4	4
MICRO		NO	YES	YES	NO	NO
MXREF		NO	YES	YES	NO	NO
MDEP		NO	YES	YES	NO	YES
PTGEN		NO	YES	YES	NO	NO
MDES (see note)		NO	YES	YES	NO	YES
WLOAD (see note)		NO	YES	YES	NO	YES

NOTE: MDES and WLOAD are subroutines. This table refers to the type of calling program.

3-10. pROM TAPE GENERATOR

The pROM Tape Generator program (object part no. 92061-16003) may be used to generate mask tapes for fusing ("burning") pROM's from the object code produced by the microassembler. For additional information on the pROM Tape Generator, refer to section 12 in this manual.

3-11. PREPARATORY STEPS

Condensed information on your preparatory steps for microprogramming appear in table 3-2 with references to the sections of this manual (or to applicable documents) for details. The letters in the reference column are keyed to entries in table 3-3. Numerals refer to sections in this manual. WCS boards to be used for microprogramming must be initialized before use. Section 14 provides examples of the procedure that you may use.

Table 3-2. Preparatory Steps

STEP	TASKS	REFERENCE (Table 3-3 or manual sections)
1	Establish your microprogramming goal. (Develop your own microprogram directly or try one of the supplied examples first. For example, run a short microprogram from start to finish by referring to section 14.	1, 14
2	Become familiar with the computer and steps to microprogramming (hardware, timing, and CM mapping).	2, 3, 5, 6
3	Establish desired CM module and mapping scheme.	6, 8
4	Plan, develop, and write first-pass microprogram (or if desired simple sample microprogram).	U, 4, 7, 8, 14
5	Plan, develop, and write main memory linking program (Assembly language).	C, L, U, V, 6, 7, 14
6	Place RTE system off-line and power down if not already in this state.	C
7	Install the desired number of HP 13197A WCS boards in the computer starting at SC 10.	A, B, C
8	Generate and configure the RTE system including at *least* DVR36. (It is probably desireable to also include at least WLOAD during system generation).	C, D, E, F
9	Load the necessary (desired) microprogramming support software (from the following list) into the RTE system.	3,C
	— WLOAD (if not already loaded)	F
	— Microassembler	G
	— Cross-Reference Generator	H
	— Microdebug Editor (MDEP)	I
	— Microdebug Editor (MDES)	J
10	Microassemble your source.	9
11	If necessary, correct errors either at the source and microassemble again or debug your microprogram using MDE and WCS. **CAUTION** It is possible to execute your microprogram from the MDE. Ensure that the RTE system you are using for microprogramming development does not have critical programs or production type programs running concurrently.	9, 10, 11
12	Load main memory program that links to microprogram.	C

Table 3-2. Preparatory Steps (Continued)

STEP	TASKS	REFERENCE (Table 3-3 or manual sections)
13	Execute microprogram from main memory program (or MDE). **CAUTION** Before executing development microprograms, ensure that your RTE system is not involved in running production programs.	C, 10, 11
14	If necessary, correct any logical errors discovered during microprogram execution. Fix source (by microassembling again) or use MDE.	9, 10, 11
15	If you are preparing to fuse pROM's you must do so from a corrected microassembled object program (can not be done from an MDE corrected version). Correct source, microassemble and execute microprogram again. Go to step 16. — OR — If you are going to use dynamic microprogramming and your microprogram executes properly it can be used through WCS. Development complete at this point unless this was an example program. To develop your actual microprogram, go to step 1. If you have special applications (not fusing pROM's) go to step 20, 21, or 22 as appropriate.	8, 9 10
16	To prepare mask tapes for pROM generation, load the pROM Tape Generator program.	C, K, 12
17	Prepare mask tapes and have pROM's prepared.	12
18	Select appropriate accessory for pROM's and mount them.	M or N
19	Place RTE system off-line, power down, install pROM facilities, then start up and/or reconfigure the system (as appropriate).	B, C, M, or N
20	If you are going to use the special microprogramming facilities (MPP, FPP, or block I/O), begin your microprogram development at step 1 with reference to the appropriate material listed to the right.	B, P, 2, 4, 7, 13
21	If you are going to be microprogramming for system use, start at step 1 with special reference to the appropriate material listed to the right.	B, P, Q, 2, 4, 7, appendix C
22	If you are going to be microprogramming using HP accessories such as DCPC, Memory Protect, or DMS, start at step 1 with reference to the appropriate material listed to the right.	R, S, T, 4, 7

Table 3-3. Manual/Software Reference

REFERENCE (from table 3-2)	MANUAL/SOFTWARE
A	*HP 13197A Writable Control Store Reference Manual*, part no. 13197-90005.
B	Your *Computer Series Installation and Service Manual.*
C	*Real-Time Executive IV Software System Programming and Operating Manual,* part no. 92067-90001, or *Real-Time Executive II Software System Programming and Operating Manual,* part no. 92001-93001.
D	*RTE Driver DVR36 for HP 12978A/13197A Writable Control Store Board Programming and Reference Manual*, part no. 13197-90001.
E	Driver DVR36, object part no. 13197-16001.
F	WCS I/O Utility Routine, object part no. 13197-16003.
G	RTE Microassembler, object part no. 92061-16001.
H	RTE Microassembler Cross-Reference Generator, object part no. 92061-16002
I	RTE Microdebug Editor (stand-alone program, MDEP), object part no. 92061-16004.
J	RTE Microdebug Editor (callable subroutine MDES), object part no. 92061-16005.
K	RTE pROM Tape Generator, object part no. 92061-16003.
L	Your *Computer Series Operating and Reference Manual.*
M	*HP 13304A Firmware Accessory Board Installation and Service Manual*, part no. 13304-90001.
N	*HP 13047A User Control Store Kit Installation and Service Manual*, part no. 13047-90001.
P	*HP 21MX/21MX E-Series Computer I/O Interfacing Guide*, part no. 02109-90006.
Q	Your *Computer Series Engineering and Reference Documentation.*
R	*HP 12897B Dual-Channel Port Controller Installation Manual*, part no. 12897-90005.
S	*HP 12892B Memory Protect Installation Manual,* part no. 12892-90007.
T	*HP 13305A Dynamic Mapping System Installation Manual*, part no. 13305-90001.
U	Your *RTE Guide for New Users.*
V	*HP RTE Assembler Reference Manual,* part no. 92067-90003.

Section 4
MICROINSTRUCTION FORMATS

MICROINSTRUCTION FORMATS

SECTION 4

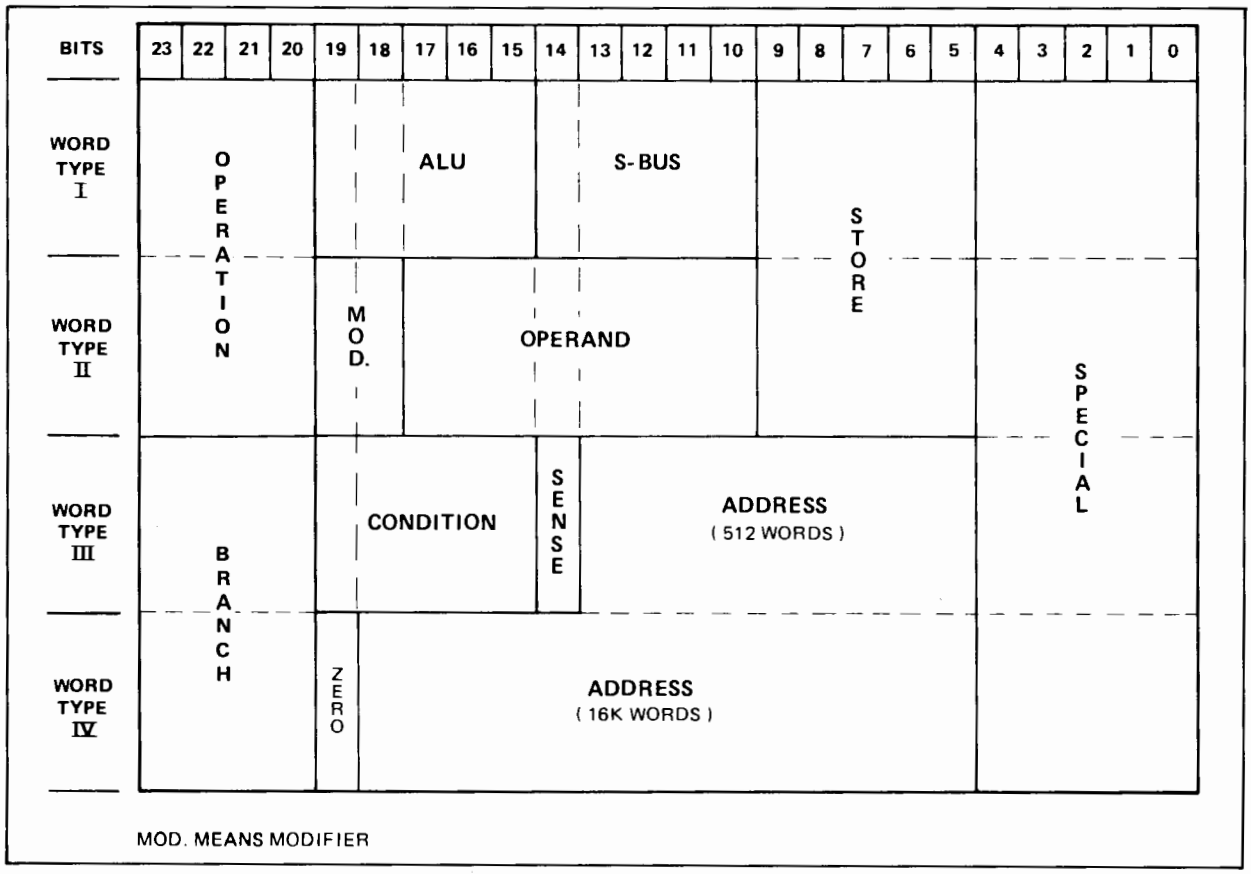
Before going further into microprogramming, you must learn the "language" in order for discussions on microaddressing, timing, etc., to be meaningful. In this section you will find:

- The microinstruction word types.
- The 24-bit microinstruction field divisions for each word type.
- The microassembler formats.
- The definitions and uses for all micro-orders.
- The binary format for each micro-order.

Additional information that you will need to use the microassembler is presented in sections 8 and 9.

4-1. MICROINSTRUCTION BINARY STRUCTURES

Figure 4-1 shows basically how the four microinstruction word types are related. This is an overall comparison that may help while studying figure 4-2.



7115-6

Figure 4-1. Word Type/Binary Format Summary

Figure 4-2 shows the binary format of all the micro-orders in their assigned fields. Specific microinstructions are constructed from the available micro-orders for the particular word type. For example,

READ	NOR	P	S1	L1
(1001	11110	11110	10000	10010)

is a word type I microinstruction as it would appear in the microinstruction register (MIR).

Note that for word type I in figure 4-2, the S-bus and Store field micro-order mnemonics are nearly the same. Where there are differences between the two fields, spaces are intentionally included to keep the similar micro-order mnemonics lined up to simplify the use of the chart.

All micro-order definitions are given in table 4-1. The table can be used in conjunction with figure 4-2, the binary format, or with figure 4-4, the microassembler format. You'll be using the microassembler format most, but the bits have to be looked at if you want to find the address of a branch (jump) using a microassembler listing, want to check the value of a constant, or look at the bit pattern of a microinstruction to calculate the micro-orders. Appendix C contains a listing of binary fields-to-micro-orders that will aid you in these tasks.

BITS	23 22 21 20	19 18 17 16 15	14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
FIELDS	OPERATION (OP)	ALU	S-BUS	STORE	SPECIAL
WORD TYPE I	ARS 0001 CRS 0010 DIV 0101 ENV 1010 ENVE 1011 LGS 0011 LWF 0110 MPY 0111 NOP 0000 NRM 0100 READ 1001 RTN 1111 WRTE 1000	ADD 00110 AND 10100 CMPL 11010 CMPS 11111 DBLS 00011 DEC 00000 INC 01111 IOR 10001 NAND 11011 NOR 11110 NSAL 11101 NSOL 10111 ONE 10011 OP1 01110 OP2 01101 OP3 01011 OP4 01010 OP5 01000 OP6 00111 OP7 00101 OP8 00100 OP10 00010 OP11 00001 OP13 11100 PASL 10101 PASS 10000 SANL 11000 SONL 10010 SUB 01001 XNOR 10110 XOR 11001 ZERO 01100	A 00011 B 00100 CAB 00001 CIR 01010 CNTR 01011 DES 01110 DSPI 00111 DSPL 00110 IOI 00101 LDR 01100 M 01101 MEU 01001 MPPA 00010 MPPB 01000 NOP 01111 P 11110 S 11111 SP 11011 S1 10000 S2 10001 S3 10010 S4 10011 S5 10100 S6 10101 S7 10110 S8 10111 S9 11000 S10 11001 S11 11010 TAB 00000 X 11100 Y 11101	A 00011 B 00100 CAB 00001 CNTR 01011 DSPI 00111 DSPL 00110 IOO 00101 IRCM 01100 L 01010 M 01101 MEU 01001 MPPA 00010 MPPB 01000 NOP 01111 P 11110 PNM 01110 S 11111 SP 11011 S1 10000 S2 10001 S3 10010 S4 10011 S5 10100 S6 10101 S7 10110 S8 10111 S9 11000 S10 11001 S11 11010 TAB 00000 X 11100 Y 11101	ASG 11000 CLFL 01110 COV 01011 DCNT 10101 FTCH 11011 IAK 11001 ICNT 10110 INCI 11100 IOFF 11111 IOG 00110 ION 00011 JTAB 00001 L1 10010 L4 10011 MESP 01010 MPCK 11110 MPP1 11010 MPP2 01001 NOP 00111 PRST 01101 RJ30 00100 RPT 10111 RTN 00000 R1 10100 SHLT 11101 SOV 01100 SRG1 10001 SRG2 10000 SRUN 01000 STFL 01111

FIELDS	OPERATION (OP)	MODIFIER	OPERAND	STORE	SPECIAL
WORD TYPE II	IMM 1110	CMHI 11 CMLO 10 HIGH 01 LOW 00	(ANY 8-BIT CONSTANT TO THE S-BUS MODIFIED BY BITS 18 AND 19)	(SAME AS ABOVE)	(SAME AS ABOVE)

7115-7

Figure 4-2. Micro-Order Binary Formats (Sheet 1 of 2)

BITS	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

WORD TYPE III

FIELDS	BRANCH	CONDITION	BRANCH SENSE	ADDRESS	SPECIAL
	JMP 1101 JSB 1100 RTN 1111	ALZ 00000 AL0 00011 AL15 01111 CNT4 01000 CNT8 01101 COUT 00010 E 11001 FLAG 11000 HOI 00111 IR8 11110 IR11 01001 L0 00100 L15 00101 MPP 01100 MRG 11111 NDEC 10011 NINC 10010 NINT 11010 NLDR 10000 NLT 10101 NMDE 10111 NMLS 01011 NRT 10100 NSFP 01110 NSNG 11100 NSTB 10001 NSTR 10110 ONES 00001 OVFL 11011 RUN 00110 RUNE 01010 SKPF 11101	RJS 1	(ANY ADDRESS IN CURRENT 512 WORD BLOCK. IF THE MICROINSTRUCTION IS LOCATED IN THE LAST LOCATION OF A 512_{10} WORD BLOCK THE TARGET ADDRESS IS DEFINED AS THE NEXT 512_{10} WORD BLOCK. SEE TABLE 4-1.)	CNDX 00010

WORD TYPE IV

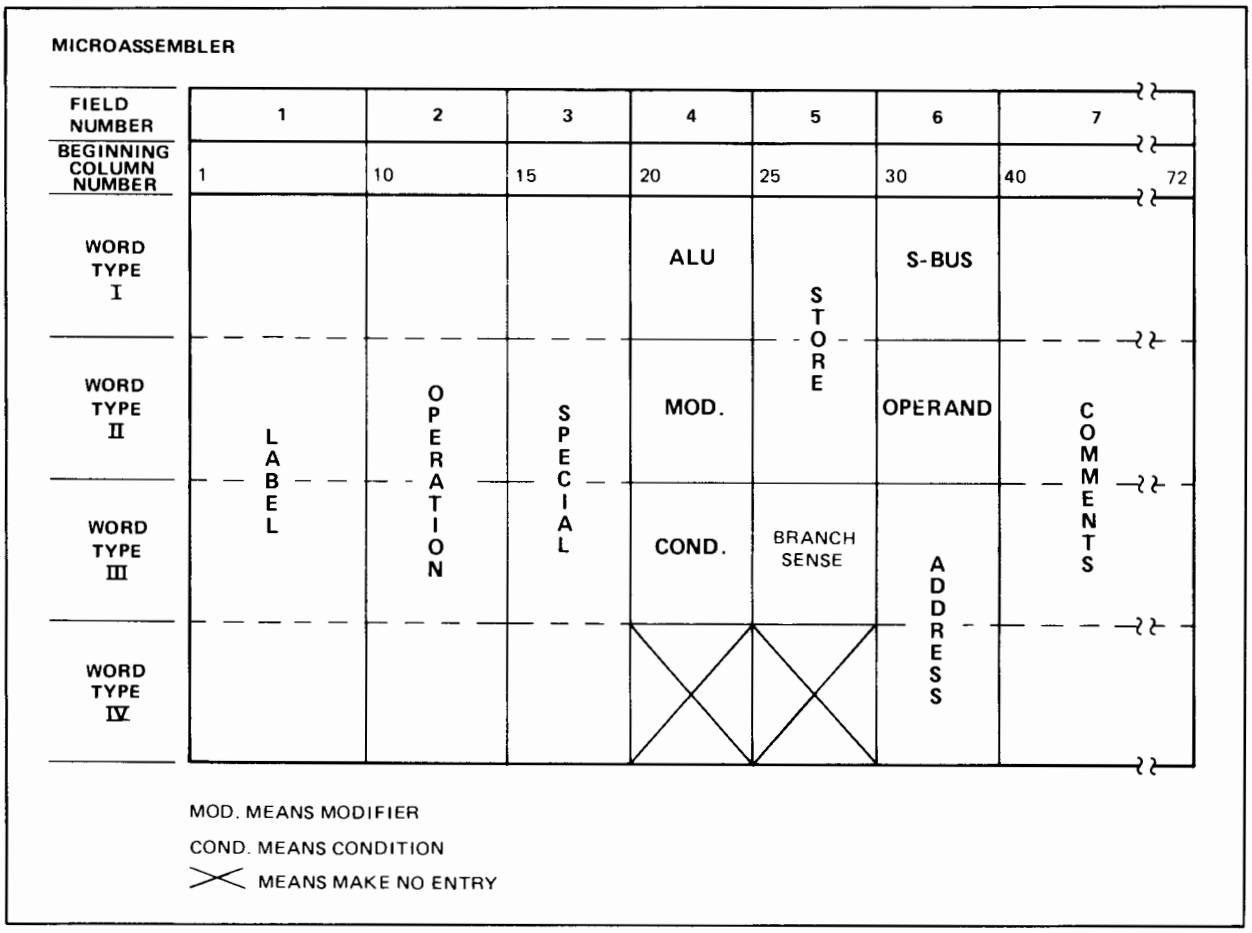
FIELDS	BRANCH	ZERO	ADDRESS	MODIFIER/ SPECIAL
	JMP 1101 JSB 1100	0	(ANY ADDRESS IN THE 16K WORD CONTROL MEMORY)	IOFF 11111 IOG 00110 ION 00011 J74 00101 NOP 00111 RJ30 00100 RPT 10111 STFL 01111

7115-8

Figure 4-2. Micro-Order Binary Formats (Sheet 2 of 2)

4-2. MICROASSEMBLER FORMATS

Figure 4-3 is similar to figure 4-1, but is arranged by the microassembler format. (The base set listing, appendix G, is an example of the microassembler format.) You will be encoding your microprograms for the RTE Microassembler this way. Note that the microassembler accepts a 72 column format.



7115-9

Figure 4-3. RTE Microassembler Word Format Summary

Figure 4-4 shows all micro-orders in their respective fields. When you have a good idea what each micro-order does, you can use this figure and the block diagram (appendix H) to microprogram expeditiously. Some microinstructions have requirements for the field entries, but the primary considerations in determining their effect are generally:

- Word type
- S-bus action
- Specials and OP codes
- Store field action
- Branch conditions, if word type III or IV

4-3. WORD TYPE I

Word type I is used to execute data transfers and operations between main memory, the I/O section, Operator Panel, Microprogrammable Processor Port (MPP), and the computer registers. The S-bus field specifies a register to be enabled onto the S-bus, the ALU field specifies an operation to be performed between this data and the L-register, and the Store field specifies what register will receive data at the end of the microcycle. The Special and Operation (OP) fields specify additional operations (e.g., the Special field can command the Rotate/Shift logic). ALU and condition flags are set or cleared after each word type I or II execution (if used) and remain in this state until changed by another microinstruction. Also for word type I and II, the Special field may contain any one of the special micro-orders except CNDX and J74. Summarizing word type I, you can handle:

- Arithmetic and logic functions
- Shifts and rotates
- Register manipulations
- Reading from and writing into memory
- Input and output operations
- Interrupts
- Subroutine returns
- Loaders
- Memory Protect
- Dynamic Mapping System operations
- Microprogrammable Processor Port functions*

*The microprogrammable Processor Port (MPP) is used to pass command and data signals to and from user designed hardware in E-Series Computers, F-Series Computers use the MPP functions to access the Hardware Floating Point Processor.

4-4. WORD TYPE II

Word type II is used for constant generation and storage. The data in the Operand (or Constant) field is enabled to the S-bus as either the upper byte (bits 15 through 8) or lower byte (bits 7 through 0) while the alternate byte becomes all logical ones. The IMM micro-order must appear in the OP field. The four micro-orders that can appear in the Modifier field control formation of the constant. As shown in figure 4-2, bit 18 controls which byte is selected for the constant. (Logical 1 means upper byte.) The ALU can either pass or complement the entire 16-bit word. Bit 19 (figure 4-2) controls the ALU action. (Logical 1 complements the word.) The Store and Special field entries are identical to those for word type I.

4-5. WORD TYPE III

Word type III is used for conditional microbranches. A microbranch is executed only if the state in the Condition field is met. You must *always* have CNDX coded in the Special field for this word type. If CNDX is not in the Special field, it becomes a word type IV (an unconditional microbranch). The Branch Sense field may be set (bit 14 a logical 1) by encoding RJS in the field and this will switch the sense of the condition for the microbranch. (See figure 4-2.) The target address that gets put in the Control Memory Address Register (CMAR) is always within the current 512_{10} microword addressing space (except for conditional branches executed in the last location of the current 512_{10} microword block, which will cause a branch into the next higher 512_{10} block (target address + 512).) The return

4	5	6	7
20	25	30	40 — 72
ALU ADD, AND, CMPL, CMPS, DBLS, DEC, INC, IOR, NAND, NOR, NSAL NSOL, ONE, OP1, OP2, OP3, OP4, OP5, OP6, OP7, OP8, OP10 OP11, OP13, PASL, PASS, SANL, SONL, SUB, XNOR, XOR, ZERO	**STORE** A, B, CAB, CNTR, DSPI, DSPL, IOO, IRCM, L, M, MEU MPPA, MPPB, NOP, P, PNM, S, SP, S1, S2, S3, S4 S5, S6, S7, S8, S9, S10, S11, TAB, X, Y	**S-BUS** A, B, CAB, CIR, CNTR, DES, DSPI, DSPL, IOI, LDR, M MEU, MPPA, MPPB, NOP, P, S, SP, S1, S2, S3, S4 S5, S6, S7, S8, S9, S10, S11, TAB, X, Y	
MODIFIER CMHI, CMLO HIGH, LOW	**STORE** (SAME AS ABOVE)	**OPERAND** (DECIMAL OR OCTAL CONSTANT)	COMMENTS
CONDITION ALZ, AL0, AL15, CNT4, CNT8, COUT, E, FLAG, HOI, IR8, IR11 L0, L15, MPP, MRG, NDEC, NINC, NINT, NLDR, NLT, NMDE, NMLS NRT, NSFP, NSNG, NSTB, NSTR, ONES, OVFL, RUN, RUNE, SKPF	**BRANCH SENSE** RJS (OR NO ENTRY)	**ADDRESS** (ANY IN CURRENT 512 WORD BLOCK. IF RTN IS ENTERED IN OP FIELD, THIS FIELD MUST BE BLANK). *IF THE MICRO-INSTRUCTION IS LOCATED IN THE LAST LOCATION OF A 512_{10} WORD BLOCK THE TARGET ADDRESS IS DEFINED AS THE NEXT 512_{10} WORD BLOCK (SEE TABLE 4-1).	
		ADDRESS (ANY ADDRESS IN CONTROL MEMORY) *	

Figure 4-4. Microassembler Format Micro-Orders

7115-10

FIELD NUMBER	1	2	3
BEGINNING COLUMN NUMBER	1	10	15
FIELDS		**OPERATION**	**SPECIAL**
WORD TYPE I		ARS CRS DIV ENV ENVE LGS LWF MPY NOP NRM READ RTN WRTE	ASG CLFL COV DCNT FTCH IAK ICNT INCI IOFF IOG ION JTAB L1 L4 MESP MPCK MPP1 MPP2 NOP PRST RJ30 RPT RTN R1 SHLT SOV SRG1 SRG2 SRUN STFL
FIELDS		**OPERATION**	**SPECIAL**
WORD TYPE II	LABELS	IMM	(SAME AS ABOVE)
FIELDS		**BRANCH**	**SPECIAL**
WORD TYPE III		JMP JSB RTN	CNDX (MUST BE ENTERED)
FIELDS		**BRANCH**	**MODIFIER/SPECIAL**
WORD TYPE IV		JMP JSB	IOFF IOG ION J74 NOP RJ30 RPT STFL

NOTES: *SEE TABLE 4-1 FOR ALLOWABLE ADDRESS ENTRIES
ONLY ONE ENTRY PER FIELD
⨯ MEANS NO ENTRY ALLOWED
ENTRIES LEFT JUSTIFIED TO BEGINNING COLUMN OF FIELD

address is saved for JSB's. If a RTN micro-order is encoded in the OP field, the address field *must* be empty. Table 4-1 outlines what kind of address entries can be made for the microassembler format. Summarizing word type III, you can accomplish:

- I/O Interrupt sensing
- Data and Arithmetic/Logic section condition sensing
- Operator Panel pushbutton operation sensing

4-6. WORD TYPE IV

Word type IV is used for unconditional microbranches. Unconditional microbranches are *always* executed. As in word type III, a return address is not saved when JMP is encoded in the OP field. A microbranch modifier may appear in the Modifier/Special field and only seven (IOFF, IOG, ION, J74, RJ30, RPT, and STFL) are available. Only four of the micro-orders actually modify the address. Word type IV can be identified by *no* CNDX code. Also, there will only be at most three fields. The microbranch target address can be *anywhere* in the 16K control memory address space. Address field entries are listed in table 4-1.

As mentioned in paragraph 4-1, you might want to be familiar with the microinstruction bit patterns so that you can calculate a microbranch address. When you look at a line of microassembler listing and examine, for example, the octal representation for a JMP microinstruction, you might see:

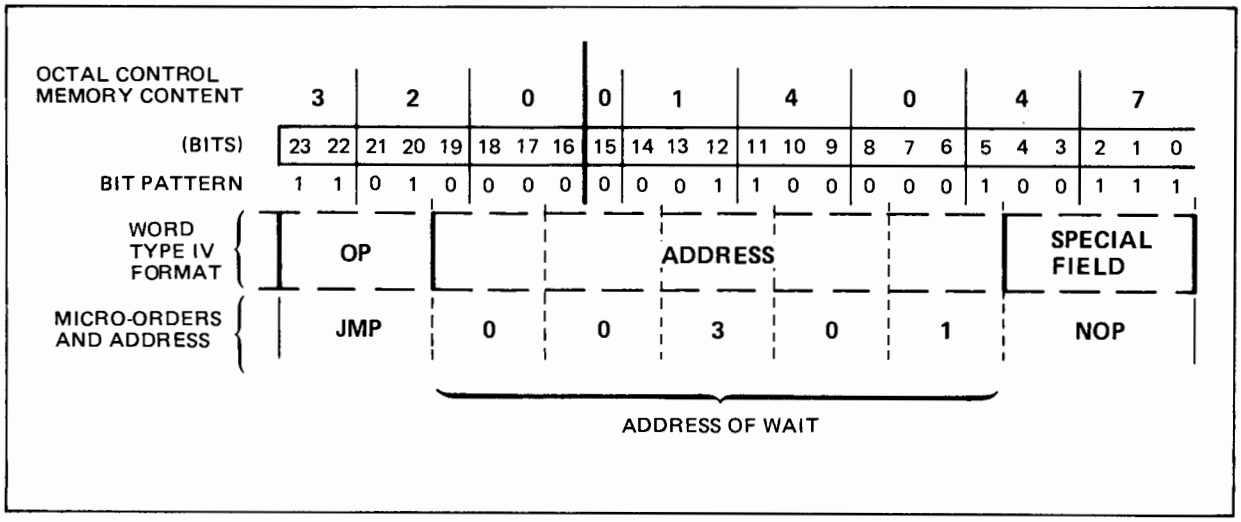
00311 320 014047 JMP WAIT

where:

00311 is the location of this microinstruction and

320 014047 is the coded content at location 00311

By converting the octal control memory content to the 24-bit word, you can determine the label WAIT address to be at 00301 as shown in figure 4-5. Note that the separation point between the three left octal digits and the six right octal digits is between bits 15 and 16. This procedure applies in a similar manner for any octal content conversion. Also see appendix B.



7115-11

Figure 4-5. Jump Address Decoding

4-7. MICRO-ORDER DEFINITIONS

Definitions for each of the micro-orders (binary and microassembler format) appear in table 4-1. Note that the operation codes (OP field) do not necessarily always dictate the entries in the other fields. Also, as previously discussed, some word types share the same micro-orders. These definitions are arranged alphanumerically in the table according to the order of microassembler field occurrence for word type I through word type IV.

Explanations and examples of the use of many of these micro-orders appear in the sections that follow; in particular, section 7. You may not want to read all the micro-order definitions before you start microprogramming. If you have not been involved in microprogramming before and just want to scan the table and look ahead, refer to sections 6 and 7, and parts III and IV of this manual where you will find some microprogramming examples.

4-8. SUMMARY

Now you have references for the:

- Binary formats of the four word types.
- Binary patterns of all micro-orders.
- Microassembler formats of the four word types.
- Definitions for all micro-orders.
- Octal to binary conversion technique that you can reverse to convert micro-orders to the binary format.

Also refer to the binary arrangement summary in appendix C.

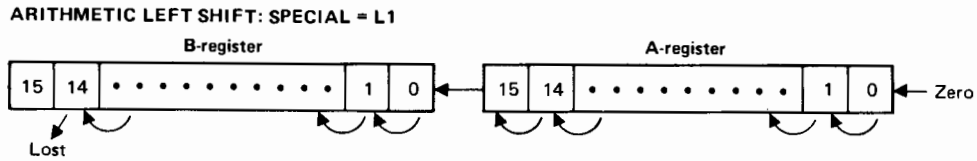
Table 4-1. Micro-Order Definitions

MICRO-ORDER	DEFINITION
	WORD TYPE I OP FIELD
ARS	Meaning: Perform a single bit arithmetic shift of the A- and B-registers combined, with the A-register forming the low-order 16 bits. The direction of the shift is specified in the Special field: L1 for left, R1 for right.

Required micro-order (field) entries:

OP	SPECIAL	ALU	STORE	S-BUS
ARS	L1 or R1	PASS	B	B

If the Special field contains L1, a 0 is shifted into bit 0 of the A-register; bit 14 of the B-register is lost, but the sign bit (bit 15) remains unchanged. The Overflow register bit is set if B-register bits 14 and 15 differ before the shift operation. One left shift multiplies by two, i.e., doubles the number.



If the Special field contains R1, the sign (bit 15) is copied into bit 14 of the B-register and bit 0 of the A-register is lost. B-register bit 15 remains the same.

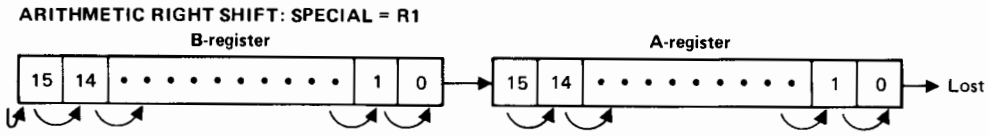


Table 4-1. Micro-Order Definitions (Continued)

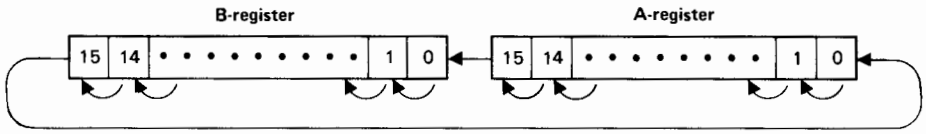
MICRO-ORDER	DEFINITION
	WORD TYPE I - OP FIELD (CONT.)
CRS	Meaning: Perform a single bit circular rotate/shift on the combined A- and B-registers with the A-register forming the low order 16 bits. The direction of the rotate is specified in the Special field: L1 for left, and R1 for right.

Required micro-order (field) entries:

OP	SPECIAL	ALU	STORE	S-BUS
CRS	L1 or R1	PASS	B	B

If the Special field contains L1, bit 15 of the B-register is transferred to bit 0 of the A-register.

CIRCULAR LEFT SHIFT: SPECIAL = L1



If the Special field contains R1, bit 0 of the A-register is transferred to bit 15 of the B-register.

CIRCULAR RIGHT SHIFT: SPECIAL = R1

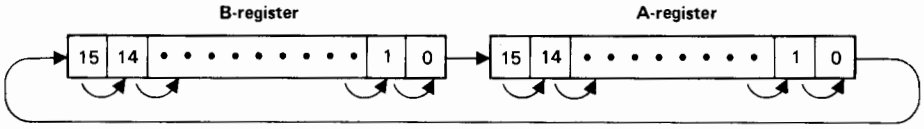


Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I - OP FIELD (CONT.)
DIV	Meaning: Perform a divide step where the divisor is in the L-register and the 32-bit dividend is in the A- and B-registers (least significant bits in the A-register). This microinstruction is usually repeated (16 times for a full word divisor) by specifying the Special field micro-order RPT in the preceding microinstruction. This performs the successive subtractions required in a divide algorithm.

Required micro-order (field) entries:

OP	SPECIAL	ALU	STORE	S-BUS
DIV	L1	SUB	B	B

The divide step is executed as follows:

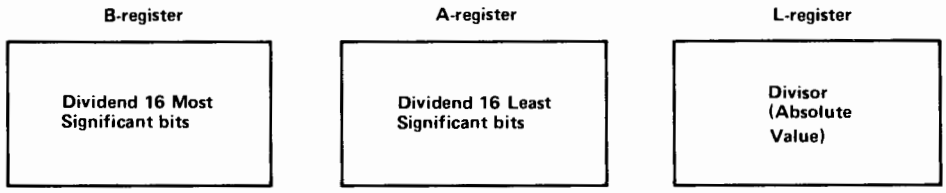
a. Subtract the L-register from the B-register (ALU = B-L)

b. If a borrow is required to complete the subtraction, the ALU Carry Out flag is clear (0). This carry out result means that the divisor (L-register) is too large. The ALU result is not stored. The A-register and B-register are left shifted one bit and the divide step is complete.

c. If a borrow is not required to complete the subtraction, the ALU Carry Out flag is set (1). This means that the divisor is small enough and the result of the subtraction is left shifted one bit and stored back into the B-register. Bit 15 of the A-register shifts into bit 0 of the B-register and bit 0 of the A-register is set to 1 (the carry out result). The divide step is complete.

Usage: The base set divide operation is shown in appendix G under the Extended Arithmetic Group instruction microroutines at label DIV. This can be used as an example in your microprogramming. When performing 16 divide steps, the numbers in the A- and B-registers should have a 32-bit left shift executed before the RPT and the first divide step. This is accomplished for proper bit alignment before the division. Also, the counter should be set for the desired number of repeat steps before the 32-bit left shift. Example:

INITIAL CONTENTS:



(Left Shifted)

AFTER REPEAT 16 TIMES OF DIVIDE STEP:

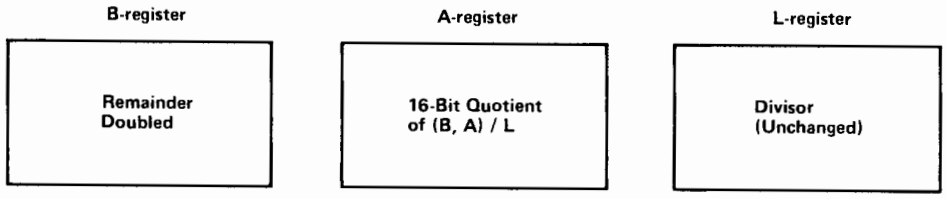


Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I - OP FIELD (CONT.)
ENV	Meaning: Enable the overflow logic for the current ALU operation. If ADD is coded in the ALU field, the Overflow register does not set unless requested. Usage: To detect an overflow (i.e., set the Overflow register bit), ENV or ENVE (see below) must be specified in the OP field of the microinstruction in which the condition is to be tested. The Overflow register is set if the S-bus and L-register bits 15 are the same and bit 15 output from the ALU is different. Caution is advised in the use of DEC (decrement) or INC (increment) in conjunction with ENV. The L-register is always compared with the S-bus. Section 7 provides further information on programmatically setting and clearing the Overflow register.
ENVE	Meaning: Enable the overflow and extend logic for the current ALU operation. Usage: To detect (test for) an overflow (i.e., set the Overflow register bit), ENV (see above) *or* ENVE must be specified in the OP field of the microinstruction in which the condition is to be tested. To set the Extend register as a result of the ALU operation, the ENVE micro-order must be specified in OP field of the microinstruction. The Extend register bit is set if there is a carry generated by the ALU (ALU Carry Out flag = 1). Example: **OP** / **SPECIAL** / **ALU** / **STORE** / **S-BUS** [ENV] [ENVE] / ADD / S3 / S3 See section 7 information on programmatically setting and clearing the Overflow register.

Table 4-1. Micro-Order Definitions (Continued)

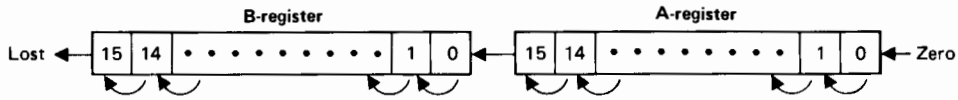
MICRO-ORDER	DEFINITION
	WORD TYPE I - OP FIELD (CONT.)
LGS	Meaning: Perform a single bit logical shift of the A- and B-registers combined, with the A-register forming the low order 16 bits. The direction of the shift is specified in the Special field: L1 for left, R1 for right.

Required micro-order (field) entries:

OP	SPECIAL	ALU	STORE	S-BUS
LGS	L1 or R1	PASS	B	B

If the Special field contains L1, a 0 is shifted into bit 0 of the A-register and bit 15 of the B-register is lost.

LOGICAL LEFT SHIFT: SPECIAL = L1



If the Special field contains R1, a 0 is shifted into bit 15 of the B-register and bit 0 of the A-register is lost.

LOGICAL RIGHT SHIFT: SPECIAL = R1

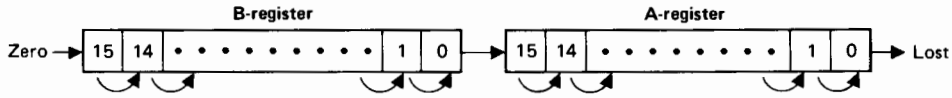


Table 4-1. Micro-Order Definitions (Continued)

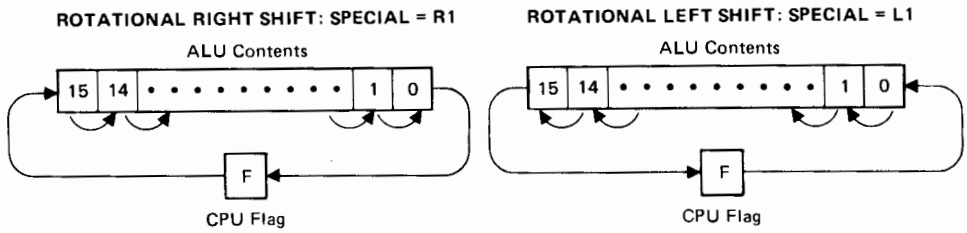
MICRO-ORDER	DEFINITION
	WORD TYPE I - OP FIELD (CONT.)
LWF	Meaning: Perform a one bit rotational shift of a 17-bit operand in the Rotate/Shifter where bit 17 is formed by the CPU flag (link with flag). The data rotates left one bit if L1 is in the Special field, or right one bit if R1 is in the Special field. If neither L1 or R1 are specified, LWF clears the CPU flag and no rotate takes place. 
MPY	Meaning: Perform a multiply step where the multiplier is in the L-register and the multiplicand is in the A-register. Required micro-order (field) entries: OP: MPY; SPECIAL: R1; ALU: ADD; STORE: B; S-BUS: B The multiply step is executed as follows: a. If bit 0 of the A-register is a one, the L-register is added to the S-bus (B-register value). The result is shifted right one bit and stored into the B-register with the ALU Carry Out flag forming bit 15. b. If bit 0 of the A-register is a zero, the S-bus (B-register value) is shifted right one bit and stored back into the B-register with the ALU Carry Out flag forming bit 15. c. In either case, the A-register is shifted right and ALU bit 0 fills vacated bit position 15. Bit 0 of the A-register is lost. The multiply step is complete. Usage: This microinstruction is usually repeated 16 times by specifying the Special field micro-order RPT in the preceding microinstruction. Each step of the multiply algorithm effectively multiplies the L-register by the A-register bit that corresponds to the step; i.e., step one multiplies the L-register by bit 0 of A-register, step two multiplies the L-register by bit 1 of the A-register, etc. Thus to multiply the L-register by all 16 bits of the A-register, MPY must be repeated 16 times. Since the B-register goes through successive right shifts and additions, the initial content of the B-register is added to the final result of the multiply algorithm. If the B-register is not zero before the multiply steps are begun, 16 multiply steps will yield the 32-bit result in the B- and A-registers (where the least significant bits (LSB's) are in the A-register).

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I - OP FIELD (Cont.)
MPY (Continued)	(B,A) = [(AxL) + B] This may be useful in some computational procedures. For example: X(2) = X (1) + (YxZ). Initial Contents: **INITIAL CONTENTS:** **B-register**: Value to be added to the final result **A-register**: Multiplicand **L-register**: Multiplier **AFTER REPEATING THE MULTIPLY STEP 16 TIMES:** **B-register**: (AxL) + B 16 Most Significant bits **A-register**: (AxL) + B 16 Least Significant bits **L-register**: Multiplier (Unchanged)

Table 4-1. Micro-Order Definitions (Continued)

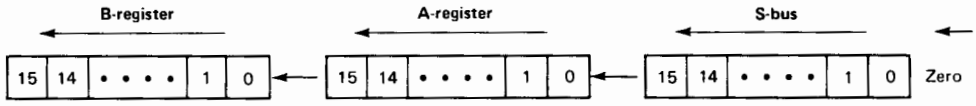
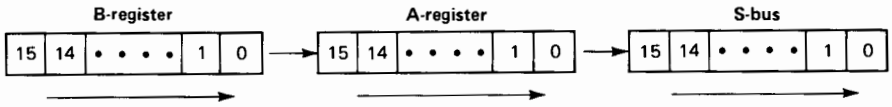
MICRO-ORDER	DEFINITION
	WORD TYPE I - OP FIELD (CONT.)
NOP	Meaning: No operation is specified for the OP field. Usage: This is the default micro-order when the OP field is left blank.
NRM	Meaning: Perform a one bit shift on the 48-bit combined value of the B-register, A-register, and S-bus data (normalize a 48-bit floating point number) as follows. Left shift: The left normalizing shift requires that the following micro-orders be used: **OP** NRM; **SPECIAL** L1; **ALU** PASS; **STORE** *; **S-BUS** * *Desired Register This will arithmetically shift the B-register, A-register, and S-bus data left one bit. If B-register bits 15 and 13 are different before the shift, the Repeat flip-flop is cleared. (Refer to the explanation of normal Repeat flip-flop operation under RPT in the Special field. This operation is an exception.)  Right shift: The right normalizing shift requires that the following micro-orders be used: **OP** NRM; **SPECIAL** R1; **ALU** PASS; **STORE** *; **S-BUS** * *Desired Register This will arithmetically shift the B-register, A-register, and S-bus data right one bit with the sign bit of the B-register preserved. No "special" conditions will clear the Repeat flip-flop (as opposed to the left shift usage). 

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I - OP FIELD (Cont.)
NRM (Continued)	A second application of the NRM micro-order is in "denormalization", or aligning floating point numbers (with different exponents). In this case, one or the other of the numbers is operated on to adjust the exponent and shift the floating point into the proper position. The number of alignment shifts is passed into the counter and the microinstruction below is repeated the appropriate number of times.

OP	SPECIAL	ALU	STORE	S-BUS
NRM	R1	PASS	S1	S1

Usage: The use of NRM in the left shift application is not as obvious as right shift. For example, assume a 48-bit two's complement number in the B-, A-, and S1-registers is to be quickly normalized. The following demonstrates the process:

LABEL	OP	SPECIAL	ALU/ COND.	STORE	S-BUS-ADDRESS
NRM48	IMM		LOW	CNTR	0
			DBLS	L	B
			XOR		B
	JMP	CNDX	AL15		*+4
		RPT			
	NRM	L1	PASS	S1	S1
	JMP				NRM48+1

Upon exit, the number is normalized and the counter contains the two's complement of the number of shifts performed.

NOTE

Floating point numbers are considered normalized when the mantissa sign bit and adjacent bit are opposite in polarity and the mantissa falls in a range of a set of numbers between zero and everything up to but not including one.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I OP FIELD (CONT.)
READ	Meaning: Read data from main memory at the address specified in the M-register and store into the T-register. The CPU will pause if main memory is busy. Usage: The M-register must be loaded prior to or during the microinstruction containing the READ micro-order. The data from main memory must be removed from the T-register within three microinstructions after the READ. Optimum performance is realized when the maximum number of microinstructions allowable are used between READ and TAB. Refer to section 7 for READ micro-order use considerations.
RTN	Meaning: Jump to the return address, i.e., branch by "popping" the "top" address in the Save Stack into the CMAR. Note that there can be three levels of microsubroutines (JSB's). Usage: For word type I, CNDX is *not allowed* in the Special field so the "pop" operation and branch are unconditionally made.
WRTE	Meaning: Write the data in the T-register into the main memory address specified in the M-register. The CPU will pause if main memory is busy. Usage: The T-register must be loaded during the microinstruction containing the WRTE micro-orders. Refer to section 7 for WRTE micro-order use considerations

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I AND II - SPECIAL FIELD
ASG	Meaning: Bits 6 and 7 of the Instruction Register (IR) determine which of the following functions are to be performed: IR bit — Alter/Skip Group 7 6 — Instruction 0 1 — (CLE) Clear Extend register 1 0 — (CME) Complement Extend register 1 1 — (CCE) Set Extend register Also, this micro-order loads the top of the Save Stack into the CMAR if the Alter/Skip Group conditions are not satisfied. It does *not* "pop" the Save Stack (i.e., the address also remains in the stack). The operation specified in the ALU field is forced to a PASS if IR bit 2 is a zero. Usage: This micro-order is used in the base set microprogram to implement the Alter/Skip Group instructions. It will not normally be used by the microprogrammer. Refer to section 7 use considerations.
CLFL	Meaning: Clear the CPU flag.
COV	Meaning: Clear the Overflow register. Refer to section 7 for information on programmatically setting and clearing the Overflow register.
DCNT	Meaning: Decrement the counter (the lower 8 bits of the IR) by one.
FTCH	Meaning: This micro-order (for use only in the base set) adjusts the Save Stack and performs other operations in relation to Memory Protect. If you are going to perform system emulation you will find further details on this micro-order in appendix C. Otherwise, it is not to be used for "normal" microprogramming.
IAK	Meaning: Freeze the computer until time period T6 and then load the interrupt address into the Central Interrupt register (CIR) and generate an IAK signal to the I/O section. Clears the Indirect Counter in Memory Protect. Also places the dynamic mapping into the system map. This microorder should not be used in a microinstruction with a READ or WRITE. Usage: Not normally used by the user microprogrammer. Refer to section 7 for interrupt handling techniques.
ICNT	Meaning: Increment the counter (the lower 8 bits of the IR) by one. Must not be followed by a word type III with a CNT4 or CNT8.
INCI	Meaning: Increment the Indirect Counter in Memory Protect (if installed) by one. Usage: Used by microprograms that implement indirect addressing. If INCI is executed three times before the next FTCH or IAK appears in the Special field, the Interrupt Enable flag is set to allow the CPU to recognize interrupts. Used to prevent multiple indirect addressing levels from holding off recognition of I/O interrupt requests. If the following microinstruction includes a JTAB in the Special field, the actual branch called by JTAB is made only if the condition mapped by bits 19 through 14 of that microinstruction are met. Refer to section 7 for interrupt handling techniques.

Table 4-1. Micro-Order Definitions (Continued)

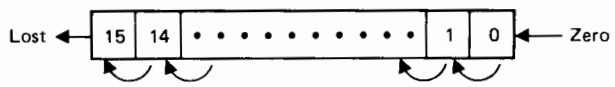
MICRO-ORDER	DEFINITION
	WORD TYPE I AND II - SPECIAL FIELD (CONT.)
IOFF	Meaning: Turn off the Interrupt Enable flag to disable recognition of power fail and I/O interrupts (does not disable Memory Protect or parity interrupts). Usage: After the occurrence of a JTAB or three occurrences of INCI (if Memory Protect is installed) interrupts are again recognized. IOFF should be used with caution since holding off interrupts could cause the loss of input and output data. Refer to section 7 for interrupt handling techniques.
IOG	Meaning: Freeze the CPU until time period T2. Then enable the generation of I/O timing signals dependent upon the instruction in the IR. Usage: Microprogrammed input and output require cooperation between the I/O section and microprogram control. Familiarity with the I/O system is mandatory. Refer to section 7 for information on forming and executing I/O microinstructions.
ION	Meaning: Turn on the Interrupt Enable flag and allow the CPU to recognize power fail and I/O interrupts until the micro-order IOFF is executed. Usage: An interrupt from any I/O device can be detected in two ways: a. If a JTAB micro-order is executed and an interrupt is pending or the Run flip-flop is clear, execution is forced to control memory (CM) location 6 (the Halt-Or-Interrupt microroutine). b. A test for interrupt pending or Run flip-flop clear can be performed by the executing microprogram by having an HOI encoded in the Condition field of a word type III microinstruction. Or, a test for a pending interrupt can be made by having NINT encoded in a word type III Condition field. The micro-order ION allows interrupts to be recognized. However, interrupts are not generated by the interrupt system unless an STF 0 I/O control command has been executed. Refer to the discussion of the interrupt system in your *Computer Series Operating and Reference Manual.* Refer to section 7 of this manual for interrupt handling considerations.
JTAB	Meaning: This micro-order (for use only in the base set) maps instructions in the IR to the proper location in CM. If you are going to perform system emulation, you will find further details on this micro-order in appendix C. Otherwise, it is not to be used for "normal" microprogramming.
L1	Meaning: Left shift one bit command to the Rotate/Shifter.  Usage: Refer to MPY, DIV, CRS, LGS, ARS, NRM, and LWF. Without one of the previous OP field micro-orders, L1 performs a one bit logical left shift on data leaving the ALU.

Table 4-1. Micro-Order Definitions (Continued)

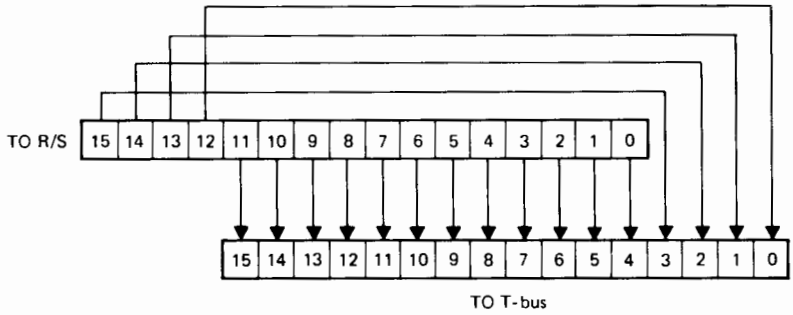
MICRO-ORDER	DEFINITION
	WORD TYPE I AND II - SPECIAL FIELD (CONT.)
L4	Meaning: Four bit left rotate command to the Rotate/Shifter. 
MESP	Meaning: Dynamic Mapping System (DMS) signal generation micro-order used in conjunction with the MEU micro-order in the Store and S-bus fields. Eight different functions are performed (designated Q0 through Q7 for reference) by combinations of MESP and MEU. The combinations of these signals and their functions are described in section 7. Usage: The DMS must be installed for the MESP and MEU micro-orders to be used. The DMS installation includes availability of the "standard" DMS Assembly language instructions which invoke the HP-written DMS microroutines. The MESP and MEU micro-orders are available for you to write microprograms using your DMS facility. You should thoroughly understand the DMS before using these micro-orders.
MPCK	Meaning: Request a Memory Protect check of the address in the M-register for a Memory Protect fence or DMS violation. Usage: This micro-order is used with any instruction that may cause a Memory Protect or DMS violation by entering or modifying protected memory. It need not be used if Memory Protect is not installed in the computer. It is subject to the following: a. Micro-orders IRCM, M, or PNM can not be specified in the Store field. b. The M-register must have the address to be checked when the microinstruction using MPCK is executed. (MPCK is usually used with the WRTE micro-order in the OP field.) Refer to section 7 for reading, writing and I/O considerations using MPCK. c. If there is not a READ or WRTE micro-order in the OP field (of the same microinstruction), the MPCK *must* follow the microinstruction containing a READ or WRTE by one or two microinstructions. The MPCK must *never* be further than two microinstructions away if Dual-Channel Port Controller (DCPC) is installed in the computer. The microinstruction below demonstrates a typical use of MPCK. **OP** WRTE; **SPECIAL** MPCK; **ALU** PASS; **STORE** TAB; **S-BUS** S1

Table 4-1. Micro-Order Definitions (Continued)

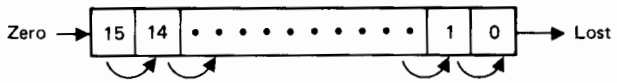
MICRO-ORDER	DEFINITION
	WORD TYPE I AND II - SPECIAL FIELD (CONT.)
MPP1	Meaning: Generate a signal $\overline{\text{PP1SP}}$ to the Microprogrammable Processor Port (MPP). Usage: Refer to the *HP 21MX/21MX E-Series Computer I/O Interfacing Guide* for further information. Example microprogrammed use can be found in section 13 of this manual.
MPP2	Meaning: Generate a signal PP2SP to the MPP. Usage: Refer to the *HP 21MX/21MX E-Series Computer I/O Interfacing Guide* for further information. Example microprogrammed use can be found in section 13 of this manual.
NOP	Meaning: No operation in the Special field. Usage: This is the default operation if no other micro-order is specified in the Special field.
PRST	Meaning: This micro-order will clear the A- and B-Addressable flip-flops (AAF and BAF). Usage: This may be used by the microprogrammer to gain access to main memory locations 0 and 1. Refer to section 7 for read and write operation considerations.
RJ30	Meaning: When used in a word type I or II microinstruction (available also in word type IV), the definition of RJ30 is identical to that of a READ micro-order in a word type I OP field (i.e., a read operation takes place and no address modification action is defined).
RPT	Meaning: Repeat the next microinstruction for the number of times specified by the positive number in the least significant four bits of the IR counter. Usage: The next microinstruction must be a word type I and must not contain RTN in the OP field or RTN or JTAB in the Special field. The Repeat flip-flop is set by this micro-order which prevents the updating of the Microinstruction Register (MIR) and CMAR at the end of the next microinstruction. The counter decrements after each execution of the next microinstruction and, when the lower four bits are all zeros, the Repeat flip-flop is cleared. (Refer to the NRM, OP field micro-order for exception.) If the four least significant bits of the counter are zeros, the next microinstruction will be repeated 16_{10} (20_8) times.
RTN	Meaning: Return from a microsubroutine; i.e., branch to the CM address in the Save Stack. This address is loaded into the CMAR. If the Save Stack is empty (no microsubroutine previously executed), a return is made to CM location 0 (zero). Usage: Three levels of microsubroutines are the maximum allowable. RTN overrides the effect of a JMP or JSB in the OP field which are not allowable with RTN encoded in the Special field.
R1	Meaning: Right shift one bit command to the Rotate/Shifter.  Usage: Used in conjunction with the shift and rotate micro-orders. Refer to MPY, DIV, ARS, NRM, CRS, LGS, and LWF. Without one of the previous micro-orders, a single bit logical right shift is executed.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I AND II SPECIAL FIELD (CONT.)
SHLT	Meaning: Clear the Run flip-flop. Usage: The Run flip-flop and RUN LED on the Operator Panel is actually cleared at the completion of the word type I or II microinstruction following the one specifying SHLT. This micro-order should be used with caution by the microprogrammer.
SOV	Meaning: Set the Overflow register. Refer to section 7 for information on programmatically clearing and setting the Overflow register.
SRG1	Meaning: Execute the shift/rotate function specified by bits 6 through 9 of the IR. (Refer to your *Computer Series Operating and Reference Manual.*) The shift-rotate function is performed on the data that leaves the ALU. If IR bit 5 is set, clear the E-register (Extend register) after the shift. The function performed in the Rotate/Shifter is determined by IR bits 6 through 9 as follows: **BITS 9 8 7 6** — **FUNCTION PERFORMED IN ROTATE/SHIFTER** 1000 — Arithmetic left shift one bit. 1001 — Arithmetic right shift one bit. 1010 — Rotational left shift one bit. 1011 — Rotational right shift one bit. 1100 — Arithmetic left shift one bit, clear sign (bit 15). 1101 — Rotational right shift one bit with E-register forming bit 16 17th bit). 1110 — Rotational left shift one bit with E-register forming bit 16 (the 17th bit). 1111 — Rotational left shift four bits. 0xxx — No shift (bits 8, 7, and 6 can have any setting) except if bits 8, 7, and 6 are 101 or 110 and E-register could be undesirably updated. (Refer to your *Computer Series Operating and Reference Manual* Shift/Rotate Group information for instructions on how to avoid this situation.) Usage: Refer to section 7 for considerations when using SRG1.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I AND IISPECIAL FIELD (CONT.)
SRG2	Meaning: Execute the shift/rotate function specified by bits 0, 1, 2, and 4 of the IR. (Refer to your *Computer Series Operating and Reference Manual.*) The shift/rotate function is performed on the data that leaves the ALU. The top of the Save Stack is loaded into the CMAR unless IR bit 3 was set (a logical 1) and bit 0 of the T-bus was zero during the last word type I or II microinstruction executed. The function performed in the Rotate/Shifter is determined by IR bits 0, 1, 2, and 4 as follows: **BITS 4 2 1 0** — **FUNCTION PERFORMED IN ROTATE/SHIFTER** 1 0 0 0 — Arithmetic left shift one bit. 1 0 0 1 — Arithmetic right shift one bit. 1 0 1 0 — Rotational left shift one bit. 1 0 1 1 — Rotational right shift one bit. 1 1 0 0 — Arithmetic left shift one bit, clear sign (bit 15). 1 1 0 1 — Rotational right shift one bit with E-register forming bit 16 (the 17th bit). 1 1 1 0 — Rotational left shift one bit with E-register forming bit 16 (the 17th bit). 1 1 1 1 — Rotational left shift four bits. 0 x x x — No shift (bits 2, 1, and 0 can have any setting) except if bits 2, 1, and 0 are 101 or 110, the E-register could be undesirably updated. (Refer to your *Computer Series Operating and Reference Manual* Shift/Rotate Group information for instructions on how to avoid this situation.) Usage: Refer to section 7 for considerations when using SRG2.
SRUN	Meaning: Set the Run flip-flop. Usage: The RUN condition is not actually set until the next word type I or II is executed.
STFL	Meaning: Set the CPU flag.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I ALU FIELD
	NOTE Symbols used in the following ALU field equations are defined here for reference. + means arithmetic function + − means arithmetic function − • means logical function "and". + means logical function "or". ⊕ means logical function "exclusive or". $\bar{S}$ or $\bar{L}$ means the one's complement of the S-bus or the one's complement of the L-register.
ADD	Meaning: Add the data placed on the S-bus to the contents of the L-register.
AND	Meaning: Logical "and" the L-register and S-bus: (L• S).
CMPL	Meaning: Ones Complement the L-register.
CMPS	Meaning: Ones complement data on the S-bus.
DBLS	Meaning: Perform the following arithmetic function in the ALU with the S-bus: S plus S.
DEC	Meaning: Decrement data on the S-bus by one.
INC	Meaning: Increment data on the S-bus by one.
IOR	Meaning: Logical "inclusive or" the L-register and S-bus: (L+ S).
NAND	Meaning: Logical "nand" the L-register and S-bus: ($\overline{L \cdot S}$).
NOR	Meaning: Logical "nor" the L-register and S-bus: ($\overline{L+S}$).
NSAL	Meaning: Logical "and" the complement of the S-bus and the L-register: ($\bar{S} \cdot L$).
NSOL	Meaning: Logical "or" the complement of the S-bus and the L-register: ($\bar{S}+L$).
ONE	Meaning: Set all 16 bits (logical one's) input to the Rotate/Shift logic.
OP1	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: (S+ L) plus 1.
OP2	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: ($S+\bar{L}$) plus 1.
OP3	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: S plus ($S \cdot \bar{L}$) plus 1.
OP4	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: (S+ L) plus ($S \cdot \bar{L}$) plus 1.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I - ALU FIELD (CONT.)
OP5	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: (S• L). This micro-order has the same effect as the SANL micro-order.
OP6	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: S plus (S• L).
OP7	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: (S+ $\overline{L}$) plus (S• L).
OP8	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: (S• L) minus 1.
OP10	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: (S+ L) plus S.
OP11	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: (S+ $\overline{L}$) plus S.
OP13	Meaning: Pass all zeros to the Rotate/Shifter. This micro-order has the same effect as the ZERO micro-order.
PASL	Meaning: Pass the L-register's contents to the Rotate/Shifter.
PASS	Meaning: Pass the S-bus data to the Rotate/Shifter. PASS is the default micro-order (NOP) in the ALU field. If no micro-order is encoded in the ALU field in a word type I microinstruction, a PASS will be inserted during microassembly. Data is not modified when a PASS appears in the ALU field.
SANL	Meaning: Logical "and" the S-bus and the complement of the L-register (S• $\overline{L}$); pass the result to the Rotate/Shifter. This micro-order has the same effect as the OP5 micro-order.
SONL	Meaning: Logical "or" the S-bus and the complement of the L-register (S+ $\overline{L}$); pass the result to the Rotate/Shifter.
SUB	Meaning: Subtract the L-register from the S-bus and pass the result to the Rotate/Shifter.
XNOR	Meaning: Logical "exclusive nor" the L-register and S-bus ($\overline{L \oplus S}$); pass result to the Rotate/Shifter.
XOR	Meaning: Logical "exclusive or" the L-register and S-bus (L$\oplus$S); pass the result to the Rotate/Shifter.
ZERO	Meaning: Pass all zeros to the Rotate/Shifter. This micro-order has the same effect as the OP13 micro-order.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I AND II- STORE FIELD
A	Meaning: Store the data on the T-bus in the A-register.
B	Meaning: Store the data on the T-bus in the B-register.
CAB	Meaning: Store the data on the T-bus in the A- or B-register according to the value of IR bit 11: IR bit 11 zero means A-register. IR bit 11 one means B-register.
CNTR	Meaning: Store the lower eight bits of the S-bus (bits 0-7) in the counter (lower 8 bits of the IR). Usage: Refer to section 7 use considerations.
DSPI	Meaning: Store the one's complement of the lower eight bits of the S-bus in the Display Indicator on the Operator Panel. (Note that only the least significant six bits are displayed.) This display indicates which register (or function) information appears in the Operator Panel Display Register. Refer to your *Computer Series Operating and Reference Manual* for details on the Operator Panel and its operation in the normal and special modes. The six indicators on the Operator Panel are associated with the S-bus as follows:

Display Indicator (S- bus) bit	7	6	5	4	3	2	1	0
Register Displayed in Normal Mode	-	-	S	P	T	M	B	A
Function Displayed in Special Mode	-	-	s	f	t	m	y	x

NOTE: Bits 7 and 6 not used.

Usage: The Operator Panel Display Indicator or Indicators can be lit by bits 5 through 0 from the S-bus as follows:

OP	SPECIAL	MOD.	STORE	OPERAND
IMM		LOW	DSPI	373B

Lights indicator pointing to M-register.

whereas:

OP	SPECIAL	MOD.	STORE	OPERAND
IMM		LOW	DSPI	010B

Lights all indicators (Special mode) except the function "t" mode (i.e., indicates that DMS map content is displayed in the Display Register).

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I AND II - STORE FIELD (CONT.)
DSPL	Meaning: Store the data on the S-bus in the Operator Panel Display Register. This information should be coordinated with the Display Indicator.
IOO	Meaning: Enable the S-bus onto the I/O bus. Usage: To be used properly, this micro-order must be issued at T4 and T5 after an IOG (Special field) micro-order for I/O operation. The IOO micro-order is not the same as the IOO backplane signal. Refer to section 7 use considerations.
IRCM	Meaning: Store the S-bus in the IR. Record the type of Assembly language instruction stored in the IR in Memory Protect hardware for use in determining any error conditions that occur during execution of the instruction. Store the least significant ten bits of the S-bus into the least significant ten bits of the M-register and clear the upper five bits of the M-register if S-bus bit 10 is zero. Usage: Refer to section 7 for information on interfacing with Memory Protect.
L	Meaning: Store the data at the output of the ALU into the L-register. Usage: The L-register is used as the second operand in arithmetic functions.
M	Meaning: Store the data on the S-bus in the M-register. Usage: Do not store into the M-register between the READ micro-order and the subsequent TAB if references to the A- or B-registers are possible. Refer to section 7 for TAB micro-order use considerations.
MEU	Meaning: DMS signal generation micro-order used in conjunction with Special field micro-order MESP and S-bus field micro-order MEU. Eight different functions are performed (designated Q0 through Q7 for reference) by combinations of MESP and MEU. The combinations of these signals and their functions are described in section 7. Usage: The DMS must be installed for the MEU and MESP micro-orders to be used. The DMS installation includes availability of the "standard" DMS Assembly language instructions which invoke the HP-written DMS microroutines. The MEU and MESP micro-orders are available for you to write microprograms using your DMS facility. You should thoroughly understand the DMS before using these micro-orders.
MPPA and MPPB	Meaning: Generate the signals $\overline{\text{MPPAST}}$ and MPBST to the MPP. Usage: Refer to the *HP 21MX/21MX E-Series Computer I/O Interfacing Guide* for further information. Example microprogram use can be found in section 13 of this manual.
NOP	Meaning: No store operation is performed; this is the default micro-order when the Store field is left blank.
P	Meaning: Store the data on the T-bus in the P-register (Program Counter).

Table 4-1. Micro-Order Definitions (Continued)

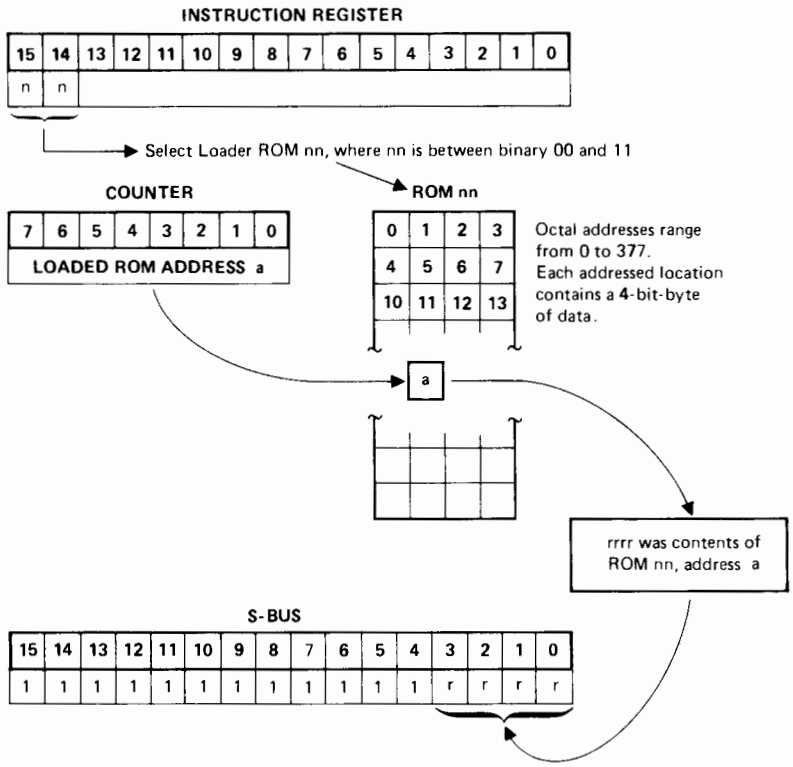
MICRO-ORDER	DEFINITION
	WORD TYPE I AND II - STORE FIELD (CONT.)
PNM	Meaning: Store the data on the T-bus in the P-register (Program Counter), and the data on the S-bus in the M-register. Usage: Useful in microprograms which perform multiword READ operations from main memory, where the P-register points to the address in main memory to be read. In a single microinstruction, the microprogram can store P into the M-register via the S-bus and then increment P via the T-bus. An example of such an application is as follows: **OP** READ; **SPECIAL**; **ALU** INC; **STORE** PNM; **S-BUS** P Refer to section 7 for the use of PNM in microinstructions with READ and WRTE micro-orders. If MPCK is used in the Special field, PNM cannot be used in the Store field.
S	Meaning: Store the data on the T-bus in the S-register.
SP	Meaning: Store the data on the T-bus in the SP-register.
S1 thru S11	Meaning: Store the data on the T-bus in the indicated Scratch Register (S1 through S11).
TAB	Meaning: Store the data on the T-bus in the A-register if the AAF (A-Addressable flip-flop) is set; store the data on the T-bus in the B-register if the BAF (B-Addressable flip-flop) is set; store the data on the S-bus in the T-register (Memory Data Register) if neither AAF nor BAF is set. Data on the M-bus (as it loads the M-register) determines the setting of AAF or BAF as follows: M-bus address when M-register store is specified / FF States: AAF, BAF / Register referenced by TAB in store (or S-bus) field. 0 / 1 / 0 / A 1 / 0 / 1 / B Any other value / 0 / 0 / T Note that the PRST micro-order clears the AAF and BAF flip-flops. Usage: This micro-order must occur concurrently when a WRTE micro-order is used. The T-register is internal to the Main Memory section. It must not be used as a working register. TAB may not be in both the Store and S-bus fields. Refer to section 7 for microprogramming considerations and the use of TAB.
X	Meaning: Store the data on the T-bus in the X-register.
Y	Meaning: Store the data on the T-bus in the Y-register.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I - S-BUS FIELD
A	Meaning: Place the contents of the A-register on the S-bus.
B	Meaning: Place the contents of the B-register on the S-bus.
CAB	Meaning: Place the contents of the A- or B-register on the S-bus according to the value of IR bit 11: IR bit 11 zero means A-register. IR bit 11 one means B-register.
CIR	Meaning: Place the contents of the CIR on the S-bus (bits 5 through 0).
CNTR	Meaning: Place the contents of the counter (lower 8 bits of the IR) on the lower 8 bits of the S-bus; the upper 8 bits are ones. See "NOTE" under IOI, below, and TAB "Usage", page 4-34.
DES	Meaning: Enable the Remote Program Load Configuration Switches onto the S-bus. These are a set of eight programmable switches that place data on the S-bus as follows: NOTE An open switch represents a logical 1 on the S-bus. Switch No. 8 7 6 5 4 3 2 1 S-Bus bit 15 14 10 9 8 7 6 0 Undriven S-bus bits are logical ones. Usage: Used in the base set microprogrammed bootstrap routine. Refer to your *Computer Series Operating and Reference Manual* operating procedures for additional loader information. Also refer to section 7 of this manual. See "NOTE" under IOI, below, and TAB "Usage", page 4-34.
DSPI	Meaning: Place the eight bits of the Operator Panel Display Indicator (complemented) on the S-bus. The upper eight bits of the S-bus are set to ones. Usage: Refer to the DSPI Store field definition for Display Indicator bit significance.
DSPL	Meaning: Place the contents of the Operator Panel Display Register on the S-bus.
IOI	Meaning: Enable the I/O bus onto the S-bus. Usage: This is used to transfer data from an I/O device to the S-bus. See section 7 for considerations in I/O microprogramming. NOTE When IOI is used in conjunction with select code 01, 02, 03, 04, or 05, the following microinstruction's S-bus field must not have CNTR, DES, or LDR if the unspecified (and assumed to be "1") S-bus bits must be in a known state; similarly, the microinstruction must not be word type II (IMM).

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I - S-BUS FIELD (CONT.)
LDR	Meaning: Place four bits from a Loader ROM on the S-bus. The address of these four bits in the ROM is contained in the counter. Determination of which of the four available Loader ROM's is specified by bits 15 and 14 in the Instruction Register. Example sequence:



Usage: Refer to the base set microroutine (appendix G), Initial Binary Loader for an example of the LDR micro-order use. Guidelines for writing loaders appear in section 7. See "NOTE" under IOI, page 4-32, and TAB "Usage", page 4-34.

M	Meaning: Place the 15-bit contents of the M-register on the S-bus. Bit 15 of the S-bus is zero.
MEU	Meaning: DMS signal generation micro-order used in conjunction with Special field micro-order MESP and Store field micro-order MEU. Eight different functions are performed (designated Q_0 through Q_7 for reference) by combinations of MESP and MEU. The combinations of these signals and their functions are described in section 7. Usage: The DMS must be installed for the MEU and MESP micro-orders to be used. The DMS installation includes availability of the "standard" DMS Assembly language instructions which invoke the HP-written DMS microroutines. The MEU and MESP micro-orders are available for you to write microprograms using your DMS facility. You should thoroughly understand DMS before using these micro-orders.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE I - S-BUS FIELD (CONT.)
MPPA and MPPB	Meaning: Generate signals $\overline{\text{MPAEN}}$ and MPBEN. (MPAEN is not used.) Usage: Refer to the *HP 21MX M-Series and E-Series Computers I/O Interfacing Guide* for further information. Example microprogram use can be found in section 13 of this manual.
NOP	Meaning: All ones are on the S-bus. Usage: This is the default micro-order when the S-bus field is not specified in a microinstruction.
P	Meaning: Place the content of the P-register on the S-bus.
S	Meaning: Place the content of the S-register on the S-bus.
SP	Meaning: Place the contents of the SP-register on the S-bus.
S1 thru S11	Meaning: Place the contents of the indicated Scratch Register (S1 through S11) on the S-bus.
TAB	Meaning: Place the contents of the T-register (Memory Data Register) on the S-bus if neither AAF (A-Addressable flip-flop) nor the BAF (B-Addressable flip-flop) is set; place the contents of the A-register on the S-bus if the AAF is set; place the contents of the B-register on the S-bus if the BAF is set. Data on the M-bus (as it loads the M-register) determines the setting of AAF or BAF. Refer to AAF, BAF flip-flop setting information under the Store field TAB micro-order. Usage: TAB may not be used in the S-bus and Store fields simultaneously. Data in the T-register must be removed within three microinstructions after the READ micro-order is used. A microinstruction with a TAB micro-order in the S-bus field must not be followed by a microinstruction with a DES, CNTR, or LDR S-bus field micro-order where the unspecified (and therefore, assumed to be "1") S-bus bits are required to be in a known state. The S-bus field TAB also must not be followed by a word type II microinstruction where the byte that is not the Operand is required to be in a known "1" state. Refer to section 7 for considerations when using TAB.
X Y	Meaning: Place the contents of the X-register on the S-bus. Meaning: Place the contents of the Y-register on the S-bus.
	WORD TYPE II - OP FIELD
IMM	Meaning: Place 16 bits on the S-bus consisting of the 8-bit binary Operand and 8 bits of ones. Determination of which 8 bits of the S-bus receive the Operand and which 8 bits receive all ones is made by the Modifier field. Usage: Refer to the word type II Modifier field micro-orders for Operand examples.
	WORD TYPE II - SPECIAL FIELD
	(All Special field micro-orders are the same as for word type I.)

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE II - MODIFIER FIELD
CMHI	Meaning: The 16 bits received by the S-bus consist of the following: Bits 15 through 8 = Operand. (Refer to the information on word type II Operand.) Bits 7 through 0 = all ones. The S-bus data is then complemented as it passes through the ALU. Usage: See below. MICROINSTRUCTION: OP: IMM; SPECIAL: ; MODIFIER: CMHI; STORE: L; OPERAND: 367B S-bus — BIT NO.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0; CONTENT: 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 OPERAND (367B) Result Out of ALU — BIT NO.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0; CONTENT: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 OPERAND Complemented
CMLO	Meaning: The 16 bits received by the S-bus consist of the following: Bits 15 through 8 = all ones. Bits 7 through 0 = Operand. (Refer to the information on word type II Operand.) The S-bus data is then complemented as it passes through the ALU. Usage: See below. MICROINSTRUCTION: OP: IMM; SPECIAL: ; MODIFIER: CMLO; STORE: S2; OPERAND: 020B S-bus — BIT NO.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0; CONTENT: 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 OPERAND Result Out of ALU — BIT NO.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0; CONTENT: 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 OPERAND Complemented

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE II - MODIFIER FIELD (CONT.)
HIGH	Meaning: The 16 bits received by the S-bus consist of the following: Bits 15 through 8 = Operand. (Refer to the information on word type II Operand.) Bits 7 through 0 = all ones. The S-bus data is then passed through the ALU without modification. Usage: See below. **MICROINSTRUCTION:** OP: IMM; SPECIAL: ; MODIFIER: HIGH; STORE: S5; OPERAND: 232B S-bus and Result Out of ALU: BIT NO. 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 CONTENT 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 OPERAND
LOW	Meaning: The 16 bits received by the S-bus consist of the following: Bits 15 through 8 = all ones. Bits 7 through 0 = Operand. (Refer to the information on the word type II Operand.) The S-bus data is then passed through the ALU without modification. Usage: See below. **MICROINSTRUCTION:** OP: IMM; SPECIAL: ; MODIFIER: LOW; STORE: S11; OPERAND: 111B S-bus and Result Out of ALU: BIT NO. 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 CONTENT 1 1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 OPERAND

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE II - STORE FIELD
	(All Store field micro-orders are the same as for word type I.)
	WORD TYPE II - OPERAND FIELD
	The Operand (eight bits) must be an integer (used as a constant). The integer can be an octal or decimal number within the following constraints: a. The decimal number must be in the range 0 to 255. b. The octal number must be in the range 0 to 377, followed by "B". Examples: 117B, 117, 198, 5, 10B
	WORD TYPE III - BRANCH FIELD
JMP	Meaning: Branch to the CM address specified in the Address field of word type III if the condition in the Condition (and Branch Sense) field is met. If the Branch Sense field is blank (RJS not specified), make the microbranch if the condition specified in the Condition field is true. If RJS is specified in the Branch Sense field, make the microbranch if the condition specified in the Condition field is false. Usage: Used in conjunction with Special field micro-order CNDX for word type III to branch in a microprogram if conditions are met as described in the Condition and Branch Sense fields. For example: BRANCH: JMP; SPECIAL: CNDX; CONDITION: AL15; BRANCH SENSE: ; ADDRESS: *+2 A microbranch will occur if bit 15 of the ALU output was set during execution of the last word type I or II microinstruction. BRANCH: JMP; SPECIAL: CNDX; CONDITION: AL15; BRANCH SENSE: RJS; ADDRESS: ADDRESS Here, a microbranch will occur if bit 15 of the ALU output was not set. If bit 15 was set, the next sequential microinstruction will be executed (no microbranch takes place).
JSB	Meaning: Perform a branch to the CM address specified in the Address field of word type III if the condition in the Condition (and Branch Sense) field is met. If RJS is not specified in the Branch Sense field, the microbranch will be made if the condition specified in the Condition field is true. If RJS is specified, the microbranch will be made if the condition is false. If the branch is made, the current microinstruction address plus one is pushed onto the Save Stack to be used as the return address. Usage: Three levels of microsubroutine branches can be made.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
WORD TYPE III - BRANCH FIELD (CONT.)	
RTN	Meaning: Branch to a return address; i.e., branch by "popping" the Save Stack into the CMAR using the address in the Save Stack. Note that there are three levels of microsubroutine branches (JSB's) so there can be three levels of RTN. Usage: For word type III, CNDX is always specified in the Special field and the "pop" operation is made *only* if the state in the Condition and Branch Sense fields is met. Otherwise, the next microinstruction is executed. Also of interest may be the discussions of JSB for word types I and III and special considerations about returns when the word type I Special field mnemonics ASG and SRG2 are used.
WORD TYPE III - SPECIAL FIELD	
CNDX	Meaning: This Special field micro-order specifies word type III - conditional branches and returns. Usage: Used in conjunction with JMP, JSB, or RTN in the Branch field.
WORD TYPE III - CONDITION FIELD	
ALZ	Meaning: The ALU output was equal to zero as a result of the last word type I or II microinstruction execution.
AL0	Meaning: Bit zero of the last output from the ALU was set by the last word type I or II microinstruction execution.
AL15	Meaning: Bit 15 of the last output from the ALU was set by the last word type I or II microinstruction execution.
CNT4	Meaning: The last four bits of the counter are zeros. Previous instruction must not contain an ICNT instruction.
CNT8	Meaning: All eight bits of the counter (lower byte of the IR) are zeros. Previous instruction must not contain an ICNT instruction.
COUT	Meaning: The ALU Carry Out flag bit was set by the last ALU operation in the last word type I or II microinstruction execution.
E	Meaning: The Extend (E) register bit is set.
FLAG	Meaning: The CPU flag bit is set.
HOI	Meaning: The Operator Panel RUN/HALT switch is not set to RUN or there is an interrupt pending (i.e., halt-or-interrupt). Usage: This micro-order is used to check for interrupts. Use is necessary because microprograms cannot be interrupted unless a check for interrupts is made. Refer to section 7 for considerations in using HOI.
IR8	Meaning: Bit 8 of the IR is set.
IR11	Meaning: Bit 11 of the IR is set.
L0	Meaning: Bit zero of the L-register is set.
L15	Meaning: Bit 15 of the L-register is set.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE III - CONDITION FIELD (CONT.)
MPP	Meaning: Test for a signal $\overline{MPP}$ received at the MPP. The L-register must not be changed in the microinstruction immediately preceeding the microinstruction containing MPP. Usage: Used in conjunction with the MPP1 and MPP2 Special field micro-orders and with MPPA and MPPB Store and S-bus field micor-orders of word type I microinstructions. Refer to the *HP 21MX M-Series and E-Series Computers I/O Interfacing Guide* for further information. Example microprogram use will be found in section 13 of this manual.
MRG	Meaning: A Memory Reference Group instruction is in the IR; i.e., IR bits 14, 13, and 12 are not all zero.
NDEC	Meaning: The Operator Panel DEC M/m pushbutton is not actuated.
NINC	Meaning: The Operator Panel INC M/m pushbutton is not actuated.
NINT	Meaning: An interrupt is not pending.
NLDR	Meaning: The Operator Panel IBL/TEST pushbutton is not actuated.
NLT	Meaning: The Operator Panel Register Select (left) pushbutton is not actuated.
NMDE	Meaning: The Operator Panel MODE pushbutton is not actuated.
NMLS	Meaning: Memory was not lost as a result of the last power down or power failure.
NRT	Meaning: The Operator Panel Register Select (right) pushbutton is not actuated.
NSFP	Meaning: A standard Operator Panel is not installed on the computer.
NSNG	Meaning: The Operator Panel INSTR STEP pushbutton is not actuated.
NSTB	Meaning: None of the following Operator Panel pushbuttons are actuated: INSTR STEP Register Select right (→) Register Select left (←) MODE IBL/TEST INC M/m DEC M/m STORE RUN PRESET
NSTR	Meaning: The Operator Panel STORE pushbutton is not actuated.
ONES	Meaning: All 16 bits of the last output from the ALU were set (tested before the Rotate/Shifter) as a result of the last word type I or II microinstruction execution.
OVFL	Meaning: The Overflow register bit is set.
RUN	Meaning: The computer's Run flip-flop is set.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE III - CONDITION FIELD (CONT.)
RUNE	Meaning: The LOCK/OPERATE switch is in the OPERATE position. NOTE In LOCK position, the RUN and HALT switches are disabled.
SKPF	Meaning: The I/O signal SFS is present (I/O time is T3 to T5) and the addressed I/O device flag is set; or, the I/O signal SFC is present (I/O time is T3 to T5) and the addressed I/O device flag is clear. Usage: Refer to section 7 for information on I/O microprogramming considerations for use of the SKPF micro-order.
	WORD TYPE III - BRANCH SENSE FIELD
RJS	Meaning: Perform the branch or return specified in the Branch field if the condition specified in the Condition field is *not* met. The Condition field micro-order specifies the condition under which a branch or return can take place; the RJS micro-order in effect reverses the sense of the condition. For example, if a conditional branch is specified if the Flag bit is set (jump if Flag bit set), the RJS micro-order will reverse the condition so that the branch occurs if the Flag bit is not set. If the Branch Sense field is blank (NOP), the condition sense is not reversed (i.e., is the same as described in each of the Condition field micro-orders).

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION

WORD TYPE III - ADDRESS FIELD

A branch may be made to any address in the current or next 512_{10} word control memory block for word type III. The entry for the microassembler format can be an octal, decimal, or a computed address.

A decimal address (d) must be in the range 0 to 511. An octal address (kB) must be in the range 0B to 777B, where the "B" signifies octal. If the word type III is located in the last address in a 512_{10} word block (i.e., address is $xx777_8$), the range is defined as the next 512_{10} word block. A computed address which is within the decimal or octal range must be in one of the following forms:

*+d
*−d
LABEL +d
LABEL −d
*+kB
*−kB
LABEL +kB
LABEL −kB
LABEL

where:

*	means "this address".
d	means a decimal number.
k	means an octal number (followed by B).
LABEL	means a microinstruction or pseudo-instruction label that is defined elsewhere in the microprogram.

Examples:

BRANCH	SPECIAL	CONDITION	BRANCH SENSE	ADDRESS
JMP	CNDX	NSNG		*+2
JMP	CNDX	FLAG		*−4
JSB	CNDX	CNT4	RJS	FETCH +1
JSB	CNDX	IR8		TIME −4
JMP	CNDX	IR11	RJS	*+7B
JMP	CNDX	L0		*−2B
JMP	CNDX	ALZ		LOOP
RTN	CNDX	ALZ	RJS	

NOTE

When RTN is encoded in the Branch field, no address should be encoded.
The address in the Save Stack is used to load the CMAR.

Except as noted above, the target address of the branch must be within the current 1000 octal (512 decimal) locations (two modules). The complete absolute address must be specified. For example, if a conditional branch microinstruction is within CM addresses 03000 and 03777, no target address may be outside the range 03000 to 03777.

Refer to section 6 for additional information on CM addressing. Refer to section 8 for information on using the RTE Microassembly language.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE IV - BRANCH FIELD
JMP	Meaning: Branch unconditionally to the address (may be modified by a Modifier/Special field micro-order) specified in the Address field. The address may be anywhere in the 16K word CM. Usage: Refer to the Modifier/Special field micro-orders and the Address field discussions.
JSB	Meaning: Branch unconditionally to the microsubroutine located at the CM address (may be modified by a Modifier/Special field micro-order) specified in the Address field. The return address is stored on top of the Save Stack and recalled by the RTN micro-order. Usage: Refer to information in the word type III Branch field JSB description. Also refer to the RTN micro-order discussion for the word type I Special field for additional information.
	WORD TYPE IV - MODIFIER/SPECIAL FIELD
IOFF	Meaning: Turn off the Interrupt Enable flag to disable recognition of normal interrupts. (Does not disable power fail, Memory Protect, or parity interrupts.) Usage: No modification is made to the microbranch address when this micro-order is used in a word type IV microinstruction. After the occurrence of a JTAB or three occurrences of INCI (if Memory Protect is installed) interrupts are again recognized. IOFF should be used with caution since holding off interrupts could cause the loss of input or output data. Refer to section 7 for interrupt handling.
IOG	Meaning: Freeze the CPU until time period T2. Then enable the generation of I/O timing signals dependent upon the instruction in the IR. Perform the JMP or JSB in the word type IV Branch field while modifying the fourth and third bits (bits 8 and 7, figure 4-2) of the Address field (according to the I/O instruction jump table) for the final address. Bits 8, 7, and 6 of the IR determine the microbranch address modification as follows: **ASSEMBLY LANGUAGE INSTRUCTION IN IR** / **IR BITS 8, 7, 6** / **ADDRESS FIELD BITS 8 AND 7 REPLACED BY:** MIA or MIB — 1 0 0 — 0 0 LIA or LIB — 1 0 1 — 0 1 OTA or OTB — 1 1 0 — 1 0 HLT — 0 0 0 — 1 1 CLO or CLF — 0 0 1 — 1 1 STO or STF — 0 0 1 — 1 1 SFC or SOC — 0 1 0 — 1 1 SFS or SOS — 0 1 1 — 1 1 STC or CLC — 1 1 1 — 1 1 Usage: IOG can also be used in the Special field of word type I, but there is no microbranch address modification since the JMP or JSB is not present. Familiarity with the I/O system is mandatory to properly use this micro-order. Refer to section 7 for more information about forming and executing I/O microinstructions.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION
	WORD TYPE IV - MODIFIER/SPECIAL FIELD (CONT.)
ION	Meaning: Turn the Interrupt Enable flag on and allow the CPU to recognize standard device interrupts until the micro-order IOFF is executed. Modify the first and second bits (bits 6 and 5, figure 4-2) of the Address field two least significant bits according to bits 1 and 0 of the IR (i.e., IR bits 1 and 0 replace bits 6 and 5 in the Address field). Usage: An interrupt from any I/O device can be detected in two ways: a. If a JTAB is executed and an interrupt is pending or the Run flip-flop is clear, execution is forced to location 6 in CM. b. A test for interrupt pending or Run flip-flop clear can be performed by the executing microprogram by having an HOI encloded in the Condition field of a word type III microinstruction. Or, a test for interrupt pending can be made by having NINT encoded in the Condition field. The micro-order ION allows interrupts to be recognized. However, interrupts are not generated by the interrupt system unless a STF 0 I/O control command has been executed. Refer to the discussion of the interrupt system in your *Computer Series Operating and Reference Manual.* Refer to section 7 for considerations for interrupt handling.
J74	Meaning: Modify the four least significant bits of the Address field (bits 8, 7, 6 and 5, figure 4-2) with bits 7 through 4 of the IR; i.e., IR bits 7 through 4 replace bits 8 through 5 in the microbranch Address field to determine the actual JMP or JSB address.
NOP	Meaning: No operation. This is the default operation if no other micro-order is specified in the Special field for word type IV. No modification is made to the JMP or JSB address.
RJ30	Meaning: Modify the four least significant bits of the Address field (bits 8, 7, 6 and 5, figure 4-2) with bits 3 through 0 of the IR and begin a READ operation of main memory; i.e., IR bits 3 through 0 replace bits 8 through 5 in the branch Address field to determine the actual JMP or JSB address. The READ operation is the same as described for the word type I OP field. Usage: Refer to the word type I OP field READ micro-order definition for M-register considerations.
RPT	Meaning: Repeat the next microinstruction for the number of times specified by the positive number in the least significant four bits of the (IR) counter. No modification to the microbranch Address field is made. Usage: Same as for the word type I and II Special field RPT micro-order.
STFL	Meaning: Set the CPU flag and then perform the JMP or JSB to the address specified in the Address field. No modification is made to the address.

Table 4-1. Micro-Order Definitions (Continued)

MICRO-ORDER	DEFINITION

WORD TYPE IV - ADDRESS FIELD

A branch may be made to any address in CM. The entry for the microassembler format can be an octal, decimal, or computed address. Same as requirements for the Address field in word type III.

A decimal address (d) must be in the range 0 to 16383. An octal address (kB) must be in the range 0B to 37777B, where the "B" signifies octal. A computed address which is within the decimal or octal range must be in one of the following forms:

*+d

*-d

LABEL +d

LABEL −d

*+kB

*−kB

LABEL +kB

LABEL −kB

LABEL

where:

*	means "this address".
d	means a decimal number.
k	means an octal number (followed by B).
LABEL	means a microinstruction or pseudo-instruction label that is defined elsewhere in the microprogram.

Examples:

BRANCH	MODIFIER/ SPECIAL	(NO ENTRY)	(NO ENTRY)	ADDRESS
JSB	IOFF			*+11
JMP				FETCH

(Refer to the word type III Address field examples.)

Refer to section 6 for additional information on CM addressing. Refer to section 8 for information on using the RTE Microassembly language.

Section 5
TIMING CONSIDERATIONS

TIMING CONSIDERATIONS

SECTION 5

Certain details about computer timing must be considered for microprogramming applications so that you can:

- Intelligently and effectively make the most use of computer time when you execute your microprograms.
- Synchronize microinstructions properly for the operations that you wish to perform with your microprograms.

The information you need about the computer's timing to effectively microprogram can be categorized into four areas:

- Basic definitions of the time periods and an idea of the functions involved in timing.
- Conditions that can vary the speed of execution of your microprograms.
- How to estimate execution time for an individual microcycle and for an I/O cycle.
- How to determine the overall effect of combined timing factors on an executing microprogram.

This section will provide you with all the basic computer timing information that you will need for microprogramming. Section 7 provides additional information on considerations involved in combining micro-orders and microinstructions for synchronizing various operations. The subject of timing involves many aspects of computer operation but the discussions in this manual will be limited to timing only as it relates to your user microprogramming.

5-1. COMPUTER SECTIONS INVOLVED IN TIMING

There are three parts or "functions" of the computer that must be considered when microprogramming:

- The Control Processor and Arithmetic Logic section.
- The Main Memory section.
- The I/O section.

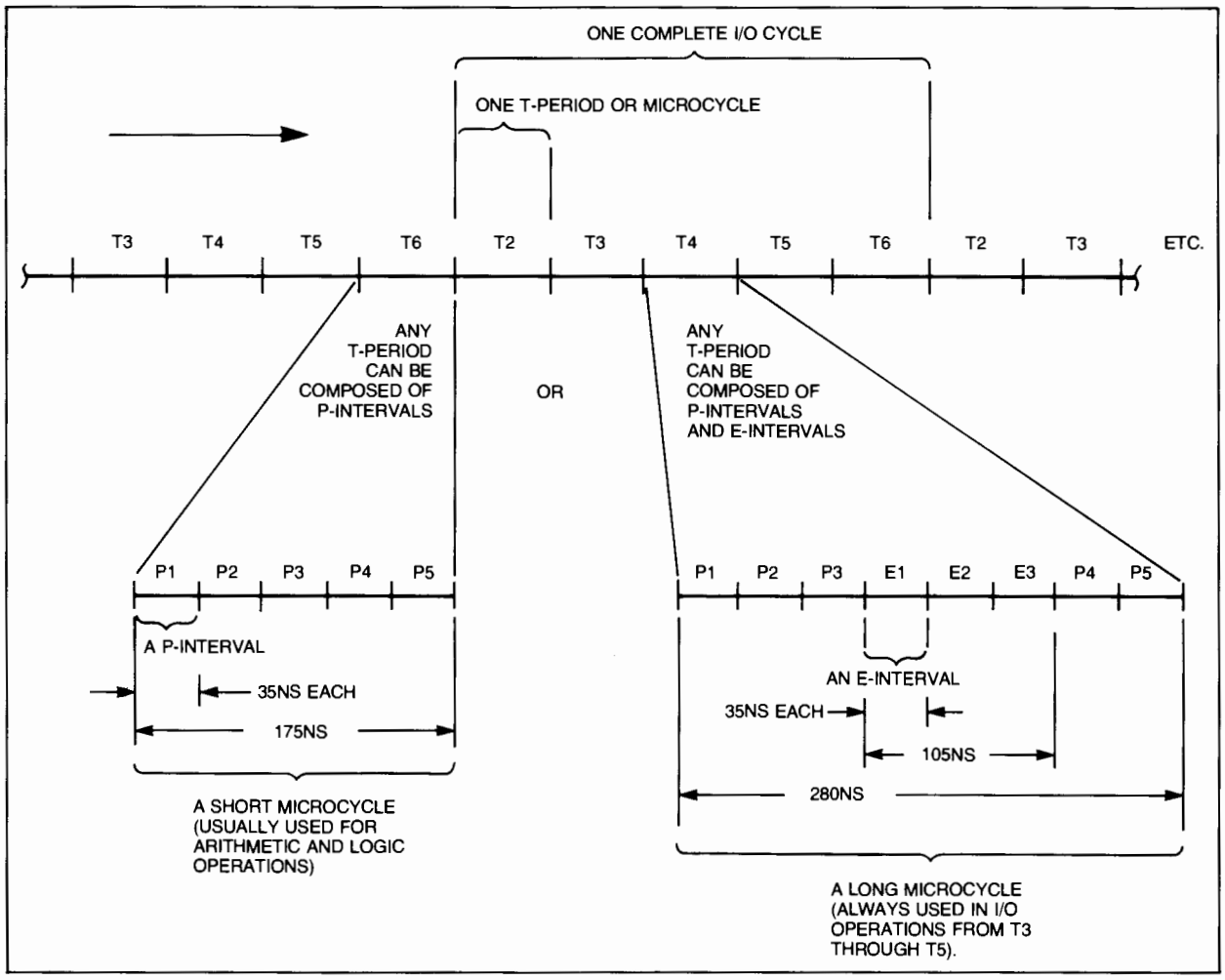
Each of these "functions" essentially operates asynchronously until they are required to communicate in order to perform a "unit" task such as a main memory read or write operation, or some I/O operation.

In normal operation, the Control Processor and Arithmetic Logic section can operate at the fastest rate of any of the functions in the computer. Main memory is the next slowest and the I/O section (understandably) requires the longest cycle time.

Some operations involving main memory take some additional time if certain accessories (DMS or DCPC) are installed. The timing factor for DMS will be discussed in this section but, for the microprogramming application, DCPC operation can only be estimated as taking a percentage of overall microprogram execution time. Section 13 provides some guidelines on calculations when considering DCPC. There is an internal main memory operation (refresh) that can be calculated by taking a percentage of overall microprogram execution time; this is also discussed in section 13. In the timing calculations in this section, these "unpredictable" factors (DCPC and memory refresh) will be considered transparent for user microprogramming applications.

5-2. REVIEW AND EXPANSION OF TIMING DEFINITIONS AND TERMS

Recall from the section 2 timing definitions that the Control Processor executes one microinstruction during one microcycle. The microcycle (also designated a T-period) is the time required to completely execute the microinstruction (which is composed of up to five micro-orders). In order to sequentially execute the micro-orders in the various fields of any particular microinstruction, it can be seen that another timing interval is needed. In figure 5-1 you will see that each microcycle is partitioned into a number of intervals designated P1 through P5 and also, for reasons which will be discussed shortly,



7115-13

Figure 5-1. Basic Timing Definitions

that intervals designated E1 through E3 also exist. Each E- or P-interval is always 35 nanoseconds long. One exception, which will be discussed shortly, is when a pause condition exists. A crystal-controlled (28.5 MHz) oscillator and timing circuits generate the 35-nanosecond intervals which are the basic "building blocks" for making up the microcycles.

Figure 5-1 also shows that any Input/Output (I/O) timing cycle is composed of five microcycles (T-periods T2 through T6). An I/O cycle is the time required to generate all the I/O signals necessary to execute any particular I/O instruction. All I/O signals and their respective generation times are described in the *HP 21MX/21MX E-Series Computer I/O Interfacing Guide,* part no. 02109-90006.

T-periods are initiated at the start of a P1 interval. Note in figure 5-1 that the length of a microcycle can vary. That is, a T-period can be either 175 nanoseconds long, or E-intervals can be inserted to extend the T-period to 280 nanoseconds. These variations and some other variable timing factors are discussed in the next paragraph.

5-3. TIMING VARIABLES

There are essentially three variable factors to consider in computer timing. They are the:

- Short or long microcycle.
- Pause.
- Timing freeze.

Each of these factors is discussed in the following paragraphs.

5-4. SHORT/LONG MICROCYCLES

As seen in figure 5-1, a short microcycle consists of five 35-nanosecond intervals that run in sequence from P1 through P5. The long microcycle consists of eight 35-nanosecond intervals that always run in the sequence P1, P2, P3, E1, E2, E3, P4, and P5. The Arithmetic/Logic section in the computer is designed to operate with a 175-nanosecond microcycle. There are three reasons for the Control Processor timing circuits to switch to long (eight 35-nanosecond intervals) microcycles:

- Certain I/O interfaces may not be able to accommodate a T-period of less than 196 nanoseconds during execution of an I/O instruction. Therefore, if an I/O operation is indicated, long microcycles are always generated from T3 through T5.

- The Memory Expansion Module (MEM), which is part of the DMS, is unable to gate data onto the S-bus fast enough when a 175-nanosecond microcycle is used. Therefore, if an MEU micro-order is in the S-bus field of a microinstruction, a long microcycle will be generated.

- The Microinstruction Register (MIR) is clocked at the beginning of each microcycle (P1) and the Control Memory Address Register (CMAR) is conditionally loaded at P3 of each microcycle. If a microbranch microinstruction is to be executed, only two P intervals, P4 and P5 (70 nanoseconds), would be left in a short microcycle to access control memory (CM) and reload the CMAR with the address of the new microinstruction then carry out the tasks normally associated with P4 and P5.

This would not be enough time to correctly reload the CMAR and access CM since CM has a worst-case access time of approximately 140 nanoseconds.* Therefore, if a microbranch is to be made, long microcycles are generated and the three extra 35-nanosecond times are added after P3 to allow enough time to complete the microbranch. A conditional microbranch microinstruction with the branch condition not met, will leave the Control Processor in the short microcycle mode.

Most microcycles will be short but a change to long microcycle timing could occur, based on prevailing conditions, during P3 of every microcycle. That is, the conditions that determine a switch to long microcycles are monitored at every P3. So, as could be expected, a great deal of microprogrammed condition testing, I/O, or DMS activity involving the S-bus will make the computer run slower.

5-5. PAUSE

As mentioned in a general way in paragraph 5-1, main memory and the Control Processor operate asynchronously until they must communicate (in a "handshaking" manner) to accomplish read or write operations. The "pause" in microcycle timing is used to interact with an asynchronous memory interface. This feature permits greater performance with existing systems and compatibility with various speed memories.

A pause operates in the following way. A read or write operation can be started with the appropriate micro-order in any microcycle. Memory is then engaged in completing the operation under its own timing (asynchronously). If the Control Processor, through another microinstruction, requests another memory operation while memory is completing the first (or another) task, a conflict in timing occurs. This possible conflict is monitored by the Control Processor at P3 of every microcycle before the Control Processor actually makes the request for the use of main memory. If a conflict is detected (i.e., there is an attempt to use memory while it is busy), the Control Processor will go into the pause state (suspend all timing clocks) until main memory is no longer busy.

A pause is accomplished by *effectively* having the timing circuits "latch-back" into P3 so that P3 is repeated for the appropriate number of times until the pending request can be processed. Pause time, therefore, will always be an integer multiple of 35 nanoseconds. At the end of the pause, the Control Processor timing will progress to either P4 or E1 (the long microcycle) depending upon the short/long microcycle conditions as discussed in paragraph 5-4.

When a memory operation has been started and memory is still busy, the conditions that can cause a pause in a microcycle are:

- An attempt to begin another read or write operation; that is, having a READ or WRTE in the OP field, or an RJ30 in the Special field of a microinstruction.
- An attempt to enable the T-register for storage from the S-bus (TAB in the Store field) or for reading the contents of the T-register onto the S-bus (TAB in the S-bus field; e.g., to obtain the results of a read operation).
- DCPC cycle in process or memory refresh operations but, as stated in paragraph 5-1, this will be transparent for microprogramming.

---

*Base set CM access time is approximately 90 nanoseconds; Writeable Control Store (WCS) CM access is about 132 nanoseconds; and Firmware Accessory Board (FAB) CM access takes the longest time (approximately 140 nanoseconds).

Figure 5-2 shows four typical examples of microcycles with a pause. Figures 5-2A and 5-2B are both short microcycles. Figures 5-2C and 5-2D are examples of long microcycles. Given specific state information (memory cycle time, memory operation being performed, etc.), the length of the extended P3 interval can be determined. Figure 5-2 shows these typical length pauses under both read and write conditions. Paragraph 5-8 specifically covers these calculations.

5-6. FREEZE

The Control Processor and I/O section operate asynchronously until an I/O instruction begins execution and communication is needed. That is, although T-periods run sequentially from T2 through T6, and each T-period is initiated by P1 of any microcycle, I/O microinstructions must begin at the appropriate part of an I/O cycle. The freeze condition therefore suspends microinstruction execution (but continues T-period generation) until the "appropriate" T-period starts.

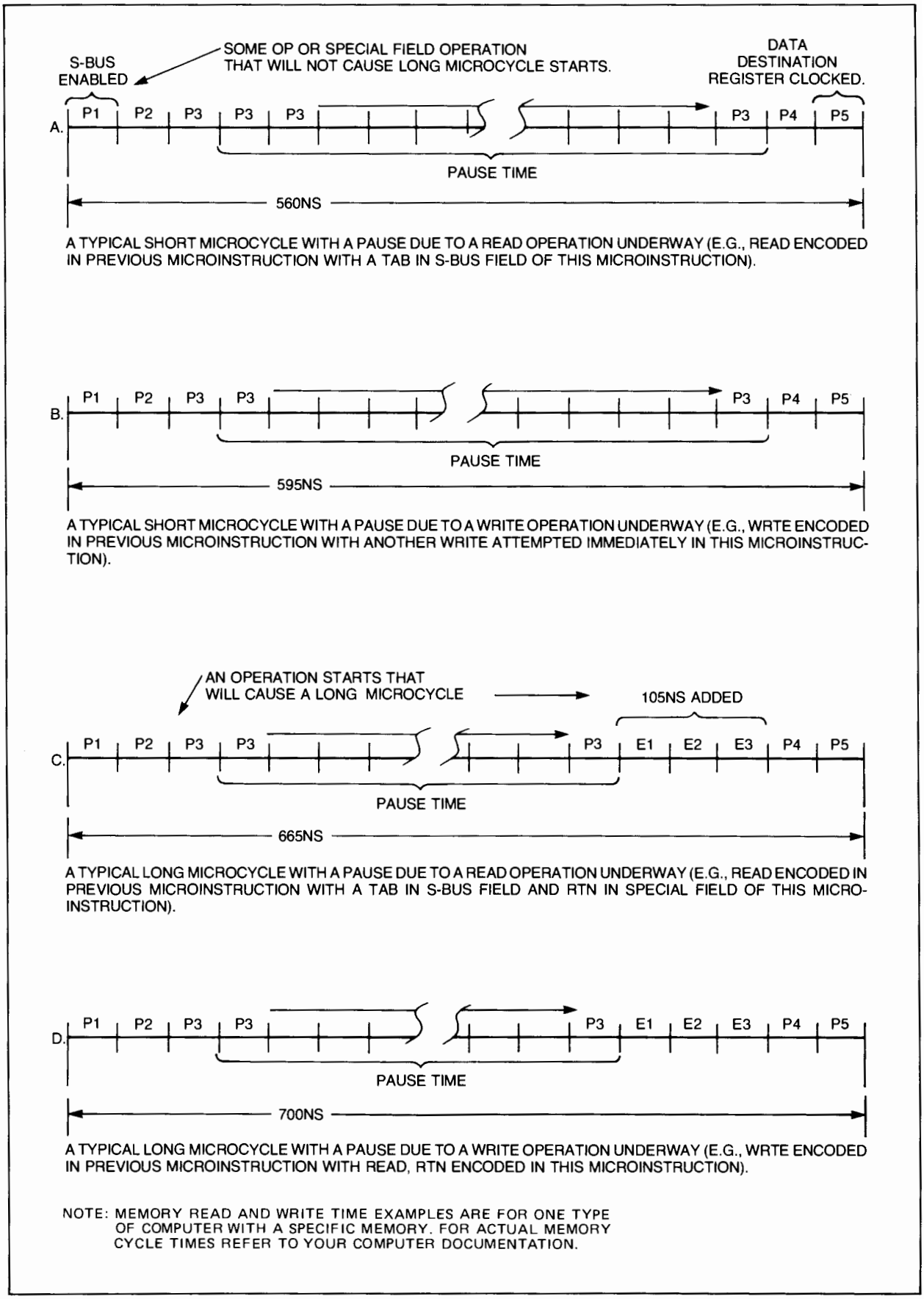
As far as microprogramming is concerned, a freeze exists to synchronize microinstruction execution with T2 or T6. Again it should be noted that DCPC activity and some memory operations may also cause freeze conditions, but these will not be considered here. For microprogramming purposes, the two factors causing a freeze condition are:

- An I/O operation is to be performed (an IOG micro-order in the Special field of a microinstruction). This will suspend all microinstruction execution until T2 starts. I/O type microinstructions can then be executed properly in the appropriate T-periods (i.e., during T3 through the end of T5).

- An interrupt acknowledge operation is to be performed (an IAK micro-order in the Special field of a microinstruction). This will suspend all microinstruction execution until T6 starts. During T6 the CIR is loaded and an IAK is generated.

The timing freeze can begin at the end of any microcycle. When I/O instructions are to be executed, long microcycles will always exist from T3 through T5 (as mentioned in paragraph 5-4).

In summary, it should be noted that the two freeze conditions mentioned above are mutually exclusive. Only one freeze can be initiated per microcycle, but a freeze condition may exist for several microcycles. In other words, if the Control Processor is not at the *beginning* of a T2 when an IOG micro-order is decoded, there will be a freeze until the start of the next T2; if the Control Processor is not at the *beginning* of a T6 when an IAK micro-order is decoded, there will be a freeze until the start of the next T6.

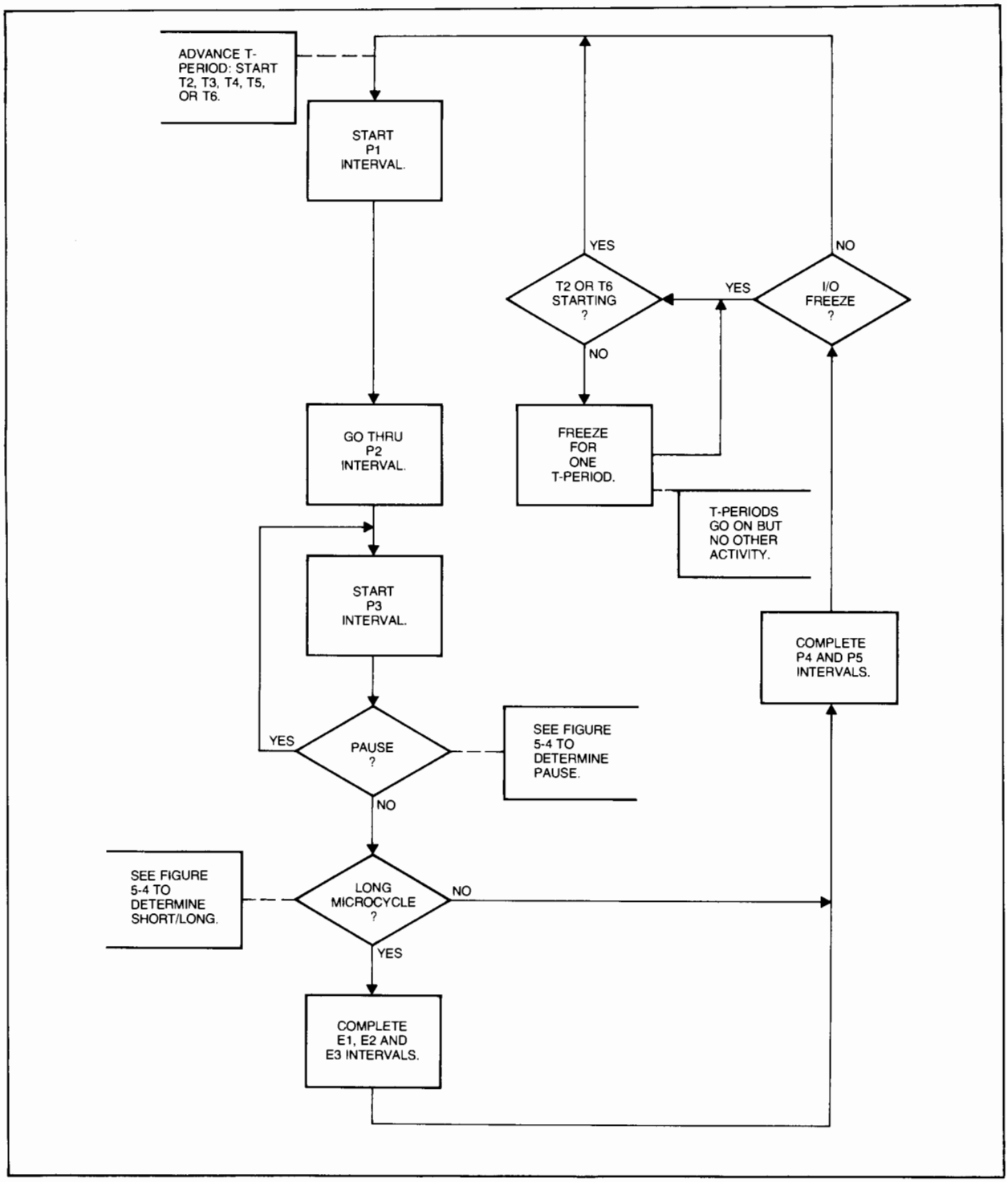


7115-14

Figure 5-2. Variable Microcycles with Pause Conditions

5-7. OVERALL TIMING

Figure 5-3 shows the sequence of timing events occurring in any given microcycle, which always starts at P1. The decision of whether or not to freeze is made at the end of the microcycle. The decision to pause or not to pause and whether or not to go to long microcycles is made in P3. It can be seen that if all three variable timing conditions are to be considered, the pause comes before the effect of long/short microcycles and a freeze will occur after the effect of either a pause or long/short microcycle.



7115-15

Figure 5-3. Overall Microcycle Timing Flowchart

Freeze or pause conditions prevail whenever communication is required between the Control Processor and the I/O section or the Main Memory section. That is, a freeze occurs to synchronize the Control Processor with the I/O section (an IOG or IAK Special field micro-order decoded). A pause occurs to suspend Control Processor operations and wait for main memory if an attempt is made to use main memory while it is still busy. If you do not attempt to use main memory while it is busy (i.e., use a READ, WRTE, RJ30, or TAB micro-order in any microinstruction), you may continue Control Processor operation. In other words, you can continue to execute microinstructions between memory operations if the above-mentioned micro-orders are not executed.

Long microcycles prevail whenever additional time is required to complete a task in a microcycle, such as for I/O operations. Also, long microcycles prevail whenever control memory branches are to be made.

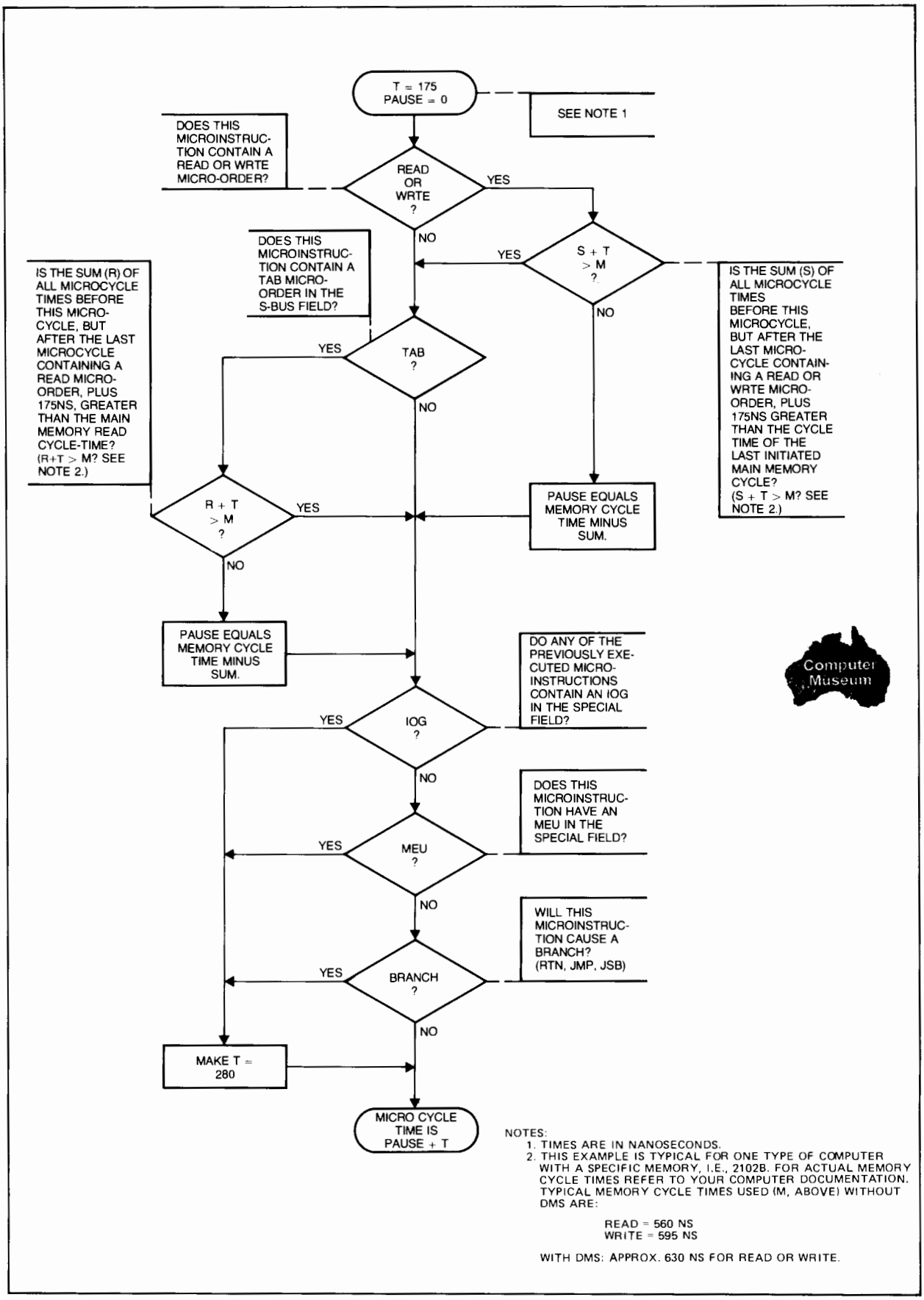
Figure 5-4 may be used in conjunction with figure 5-3 as a quick reference for estimating the time taken to complete a microcycle. Detailed calculations for typical microinstruction and microprogram execution times are discussed in paragraph 5-8.

When one or both DCPC channels are busy, the Control Processor is effectively in a freeze condition. This is why DCPC operations are considered transparent to the microprogrammer. Careful analysis of the processes you wish to accomplish with microprogramming, with the timing factors kept in mind, will provide maximum performance gain.

5-8. TIMING CALCULATIONS

The flowchart illustrated in figure 5-5 can be used to calculate the execution time for individual microcycles and also for estimating overall microprogram execution time. The flowchart is to be read from left to right once for each microcycle. To estimate the execution time for a microroutine, repetitive cycles through the flowchart must be made, noting times and remembering conditions encountered during earlier microcycles.

All conditions that change timing (for user microprograms) during any microcycle are shown in figure 5-5 along with times (in nanoseconds) that should be summed while proceeding through the microcycle. Specific micro-orders determine timing changes. Therefore, all calculations described in this section are made by comparing micro-orders against the chart. The examples that follow consider events as they occur through a microcycle with increasing complexity of timing calculations.



7115-16

Figure 5-4. Consolidated Microcycle Estimating Flowchart

5-9. ARITHMETIC/LOGIC SECTION OPERATIONS

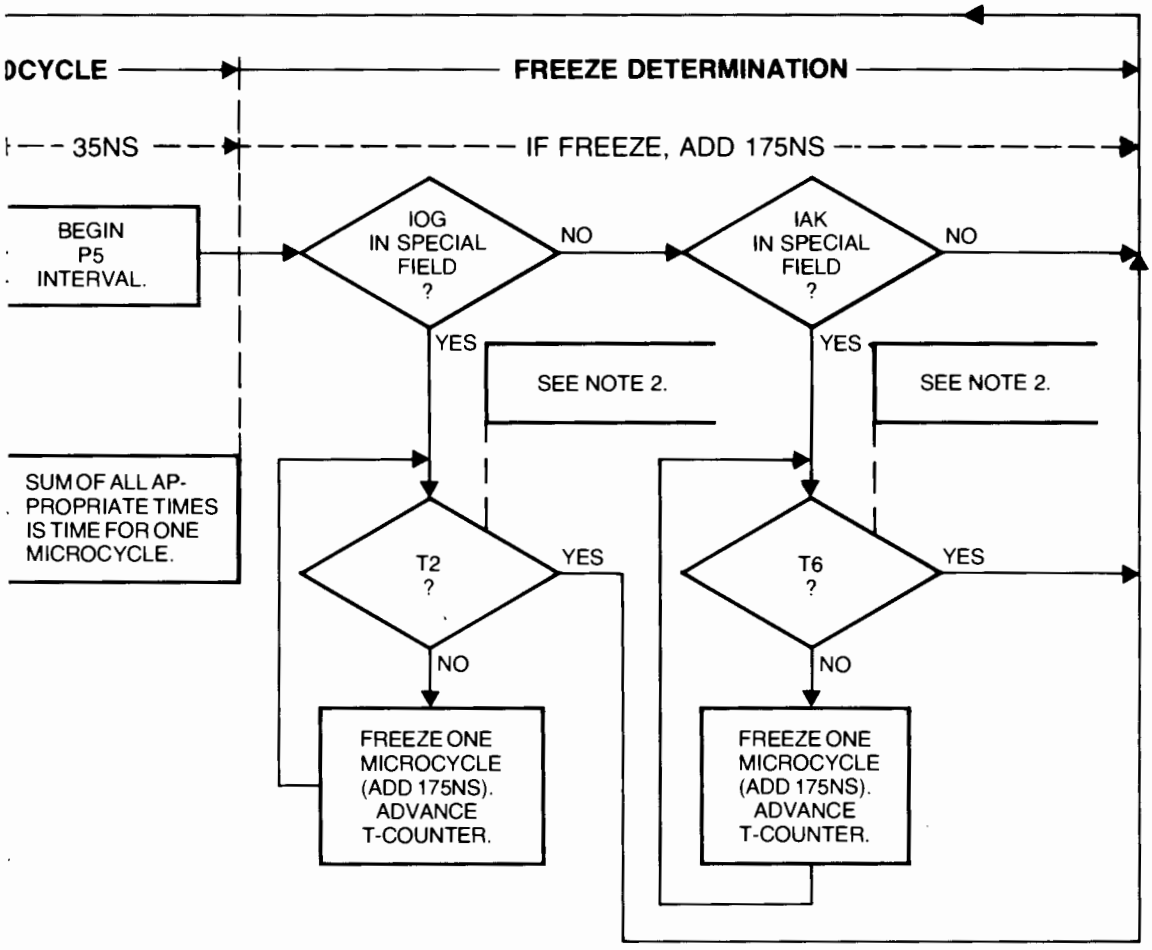
The fastest microcycle timing is found when microprogrammed operations deal with the Arithmetic/ Logic section registers. For example, suppose the timing for the following portion of a microroutine is to be estimated:

```
                    ALU/
          OP/       MOD/          S-BUS/
LABEL     BRCH  SPCL COND  STR    ADDRESS    COMMENTS

                     .
                     .
                     .
FIRST           STFL CMPS  B      B
SECOND               CMPS  A      A
THIRD                INC   A      A
                     .
                     .
                     .
                     (ETC.)
```

Read figure 5-5 from left to right with the first microinstruction in mind. The total time for the first two intervals (P1 + P2) is 70 nanoseconds. The Special field in the first microinstruction does not contain an RJ30 and the OP field does not contain a READ or WRTE. Also, the S-bus field does not contain TAB. Thus, in following the timing line into P3, note that no pause condition exists.

Continuing in P3, since an I/O operation is not being performed, you will not be concerned about the T-period in existence. The answer here will follow the decision line labeled "unknown" and assume here no IOG in the Special field within the last three microinstructions. Also, a long microcycle will not occur since there is no MEU in the S-bus field of this microinstruction and no JSB, JMP, or RTN micro-orders coded. With conditions as they are, the Control Processor timing circuits will not switch to a long microcycle. Following the timing line in figure 5-5 through the end of P3, time in this microcycle thus far is 105 nanoseconds. Intervals P4 through P5 are executed immediately making the total time for execution of the microinstruction labeled FIRST = 175 nanoseconds. Recall that it was assumed that no freeze conditions are in effect for this example, thus the timing line can be followed back to the beginning of P1.

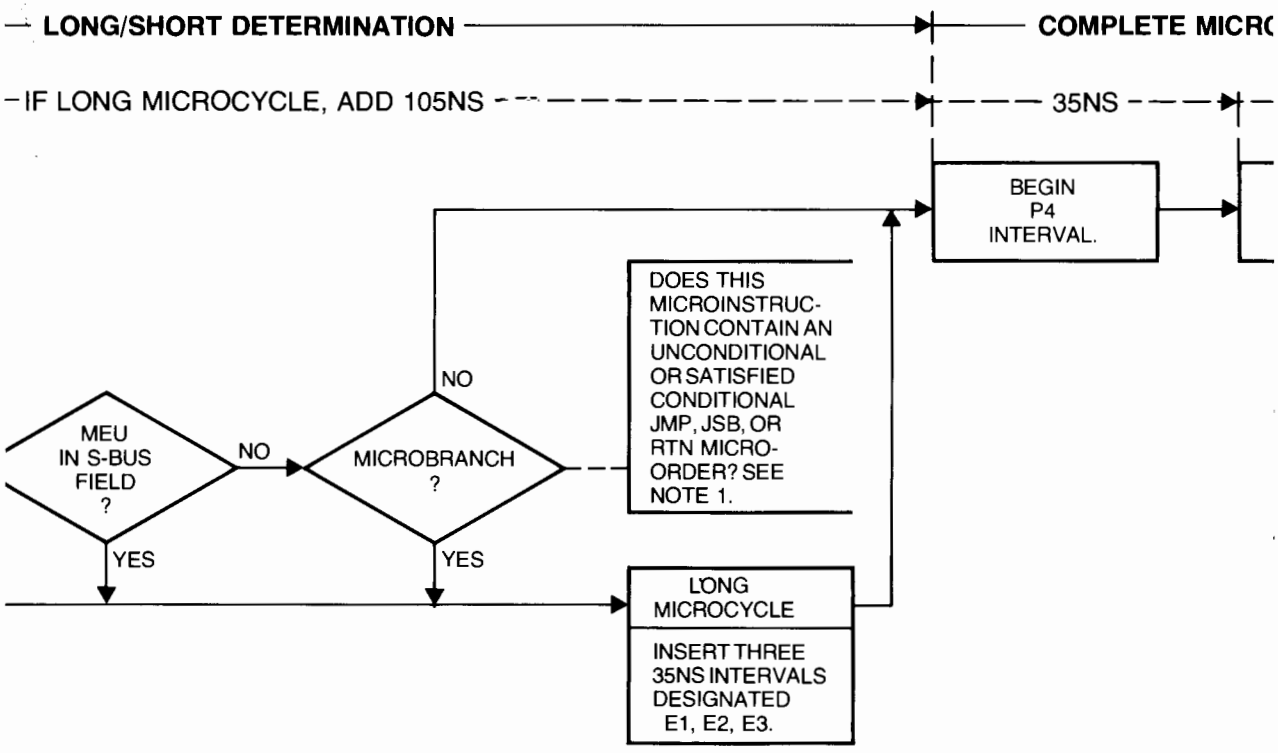
Microinstructions SECOND and THIRD are executed in a similar manner (check the microroutine using the flowchart). The total time for this microroutine is 525 nanoseconds.



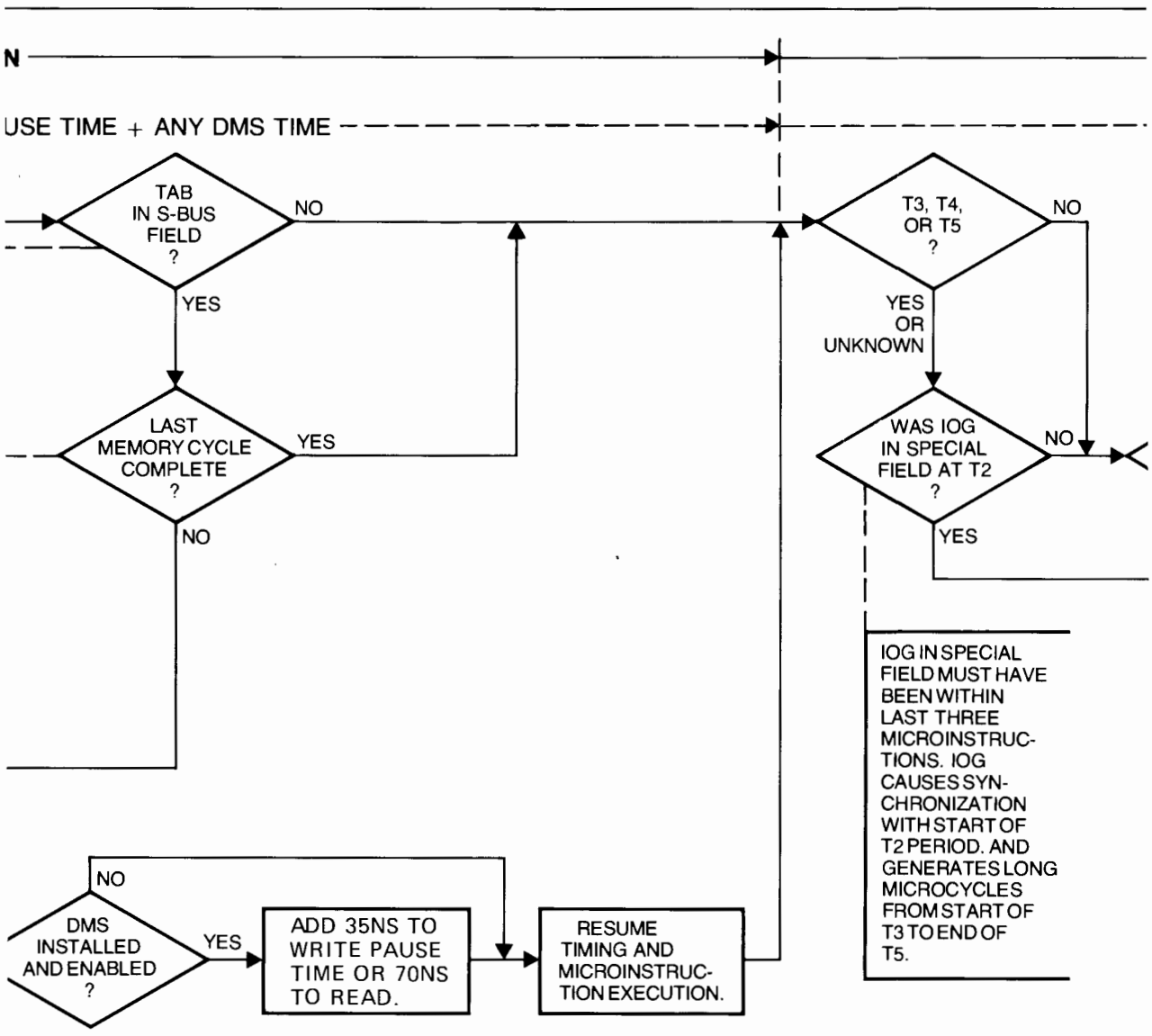
DETERMINE. ASSUME A PERCENTAGE OF

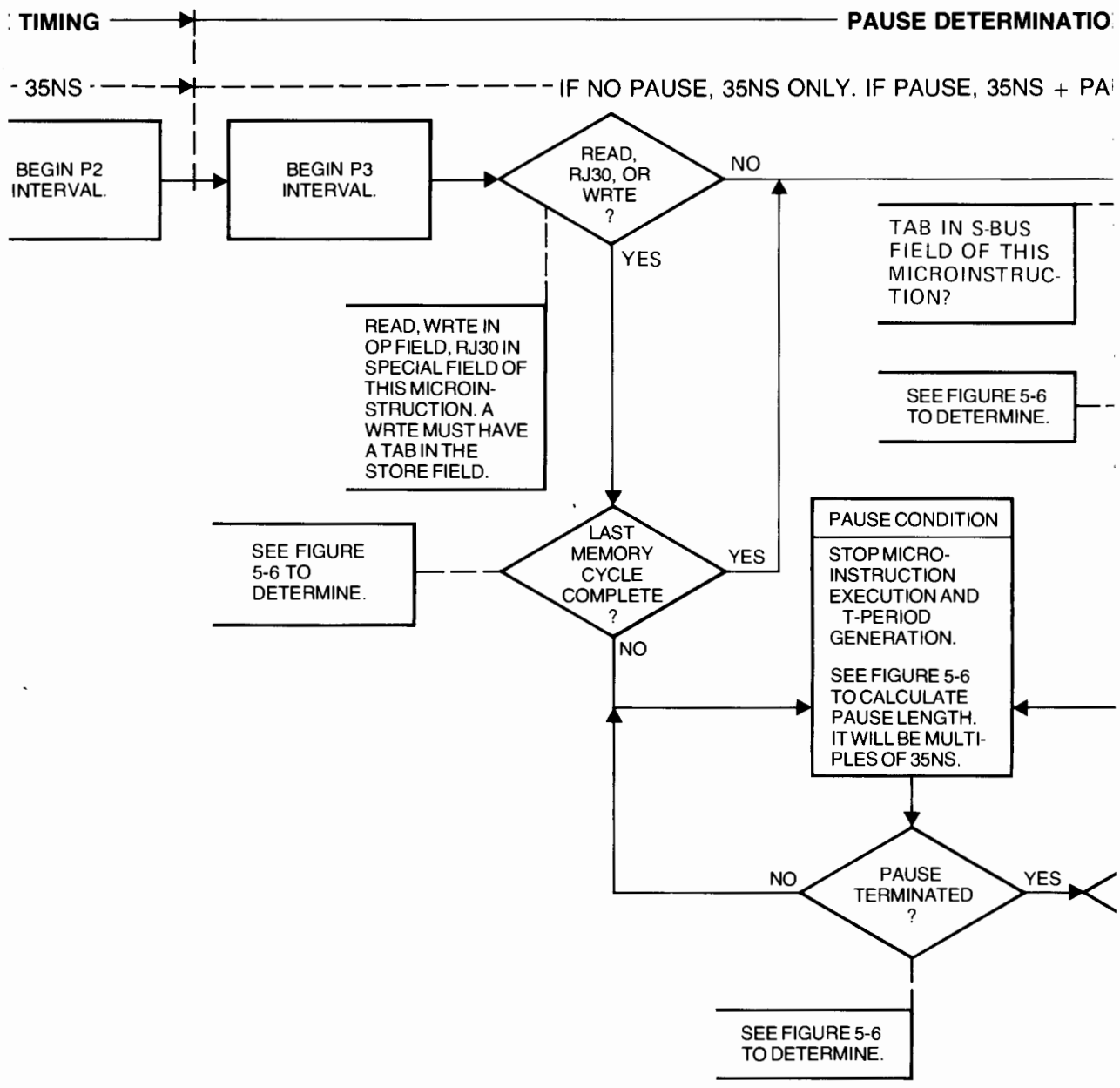
AN I/O CYCLE TREAT THE ESTIMATE AS

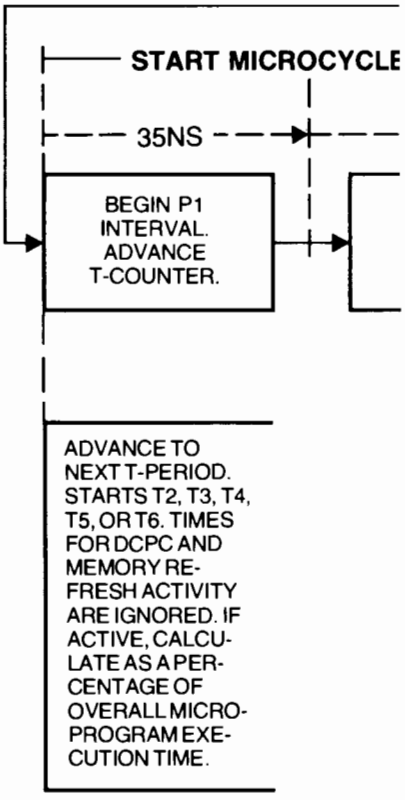
Figure 5-5. Detailed Microcycle Time Determination Flowchart



NOTES:
1. CONDITIONAL MICROBRANCHES NOT MET MAY BE DIFFICULT TO D
   BRANCHES MET BASED ON YOUR APPLICATION.
2. TO DETERMINE WHICH T-PERIOD IS PRESENT WHEN BEGINNING
   RANDOM.


N
USE TIME + ANY DMS TIME
TAB IN S-BUS FIELD ?
NO
YES
LAST MEMORY CYCLE COMPLETE ?
YES
NO
T3, T4, OR T5 ?
NO
YES OR UNKNOWN
WAS IOG IN SPECIAL FIELD AT T2 ?
NO
YES
IOG IN SPECIAL FIELD MUST HAVE BEEN WITHIN LAST THREE MICROINSTRUCTIONS. IOG CAUSES SYNCHRONIZATION WITH START OF T2 PERIOD. AND GENERATES LONG MICROCYCLES FROM START OF T3 TO END OF T5.
NO
DMS INSTALLED AND ENABLED ?
YES
ADD 35NS TO WRITE PAUSE TIME OR 70NS TO READ.
RESUME TIMING AND MICROINSTRUCTION EXECUTION.


TIMING
PAUSE DETERMINATIO
35NS
IF NO PAUSE, 35NS ONLY. IF PAUSE, 35NS + PA
BEGIN P2 INTERVAL.
BEGIN P3 INTERVAL.
READ, RJ30, OR WRTE ?
NO
YES
TAB IN S-BUS FIELD OF THIS MICROINSTRUCTION?
READ, WRTE IN OP FIELD, RJ30 IN SPECIAL FIELD OF THIS MICROINSTRUCTION. A WRTE MUST HAVE A TAB IN THE STORE FIELD.
SEE FIGURE 5-6 TO DETERMINE.
SEE FIGURE 5-6 TO DETERMINE.
LAST MEMORY CYCLE COMPLETE ?
YES
NO
PAUSE CONDITION
STOP MICRO-INSTRUCTION EXECUTION AND T-PERIOD GENERATION.
SEE FIGURE 5-6 TO CALCULATE PAUSE LENGTH. IT WILL BE MULTIPLES OF 35NS.
PAUSE TERMINATED ?
NO
YES
SEE FIGURE 5-6 TO DETERMINE.


START MICROCYCLE
35NS
BEGIN P1 INTERVAL. ADVANCE T-COUNTER.
ADVANCE TO NEXT T-PERIOD. STARTS T2, T3, T4, T5, OR T6. TIMES FOR DCPC AND MEMORY RE-FRESH ACTIVITY ARE IGNORED. IF ACTIVE, CALCU-LATE AS A PER-CENTAGE OF OVERALL MICRO-PROGRAM EXE-CUTION TIME.

5-10. CONTROL MEMORY BRANCHES

The switch to long microcycles is made in P3 when any of the three conditions shown in figure 5-5 can be answered affirmatively. For example, consider a control memory branch condition shown in the following portion of a microroutine. In this example the microcycle times are included in the right-hand column.

```
                    ALU/
          OP/       MOD/          S-BUS/
LABEL     BRCH SPCL COND  STR     ADDRESS   COMMENTS

*                                           TIME (NS)
*                                           (IF BRANCH MET) (IF NOT MET)
                    .
                    .
                    .
START               ADD   L       S3        175             175
ONE       JSB  CNDX L15           CLEAR     280             175
TWO                 INC   S3      L                         175
THREE     RTN  CLFL       A       S3                        280
CLEAR     IMM  RTN  CMHI  L       377B      280
                                            ---             ---
                                            735 NS          805 NS
                    .
                    .
                    .
                    (ETC.)
```

By using figure 5-5 and checking the microroutine, it can be seen that the JSB and RTN micro-orders in the microinstructions labeled ONE, THREE, and CLEAR can cause long microcycles.

5-11. I/O OPERATIONS

Suppose the T-period is T4 and the Control Processor has just placed the first microinstruction of your microroutine in the MIR. Suppose further that part of the microroutine is as follows (note the time column):

```
                    ALU/
          OP/       MOD/          S-BUS/
LABEL     BRCH SPCL COND  STR     ADDRESS   COMMENTS

*                                           TIME (NS)
                    .
                    .
                    .
XXX               IOG           IRCM      S4        T4 175
*                  ↓              ↓         ↓        T5 175
*                 (SUSPENDED EXECUTION UNTIL T2)     T6 175
*                    (NOW EXECUTION CONTINUES)       T2 175
                  NOP                               T3 280
                  NOP                               T4 280
                                S5        IOI       T5 280
                        INC     S8        S3        T6 175
                                                    T2
                    .
                    .
                    .
                    (ETC.)
```

The microinstruction at label XXX includes micro-orders in the S-bus and Store fields as well as the IOG micro-order in the Special field. As P1 and P2 occur, the S-bus and Store field micro-orders will be executed but the effect of the IOG in the Special field is not felt until the end of the microcycle. Also, (in following the timing line in figure 5-5) note that the freeze condition is not in effect until the microinstruction labeled XXX completes execution. At the end of the microcycle, the IOG micro-order causes all microinstruction execution to be suspended until T2 completes. The total waiting time in the freeze condition in this case is 525 nanoseconds. Note that with a freeze condition present, T-periods will be short microcycles until synchronization occurs. Time T3 starts the I/O cycle and each microinstruction is executed in the appropriate *long* microcycle (T-period). If T6 is short (as shown in the example), the total time for the I/O cycle will be 1.120 microseconds. If T6 had been long (e.g., a RTN coded), the total time for the I/O cycle would be 1.225 microseconds. This example microroutine is used only to illustrate the freeze until T2 starts. Section 7 provides appropriate microprogramming considerations. An IAK micro-order in the Special field can cause a freeze until the start of T6. That is, (follow the timing line in figure 5-5) at the end of the microcycle where an IAK Special field micro-order has been included in the microinstruction just executed, a freeze will occur until the end of T6. During the T6 period microcycle, the appropriate functions for the IAK micro-order will be executed.

5-12. MAIN MEMORY OPERATIONS

Typical main memory cycle times for reading and writing differ. Therefore, calculations for read and write operations are discussed separately. The example read and write times are for an HP 2102B Memory.

5-13. READING FROM MEMORY. First consider a read from main memory with a TAB micro-order in the S-bus field two microinstructions after the microinstruction containing the READ micro-order. In the example microroutine below, assume no memory operation is in progress as the microroutine begins at label START (assume you do not have the DMS installed). The letters shown in the timing comments are keyed to the text explanation that follows this microroutine.

```
                ALU/
        OP/             MOD/            S-BUS/
LABEL   BRCH    SPCL    COND    STR     ADDRESS     COMMENTS

*                                                   TIME (NS)
                        .
                        .
                        .
START                   PASS    S1      P           175        A↘
FIRST   READ            PASS    DSPL    S11         175 ______________
SECOND                  INC     PNM     P           175↘
THIRD                   DEC     X       X           175↔B  ≥560←C
DATA                    PASS    S2      TAB         210←D     ↓
END             RTN             IRCM    S2          280←E
                        .
                        .
                        .
                        (ETC.)
```

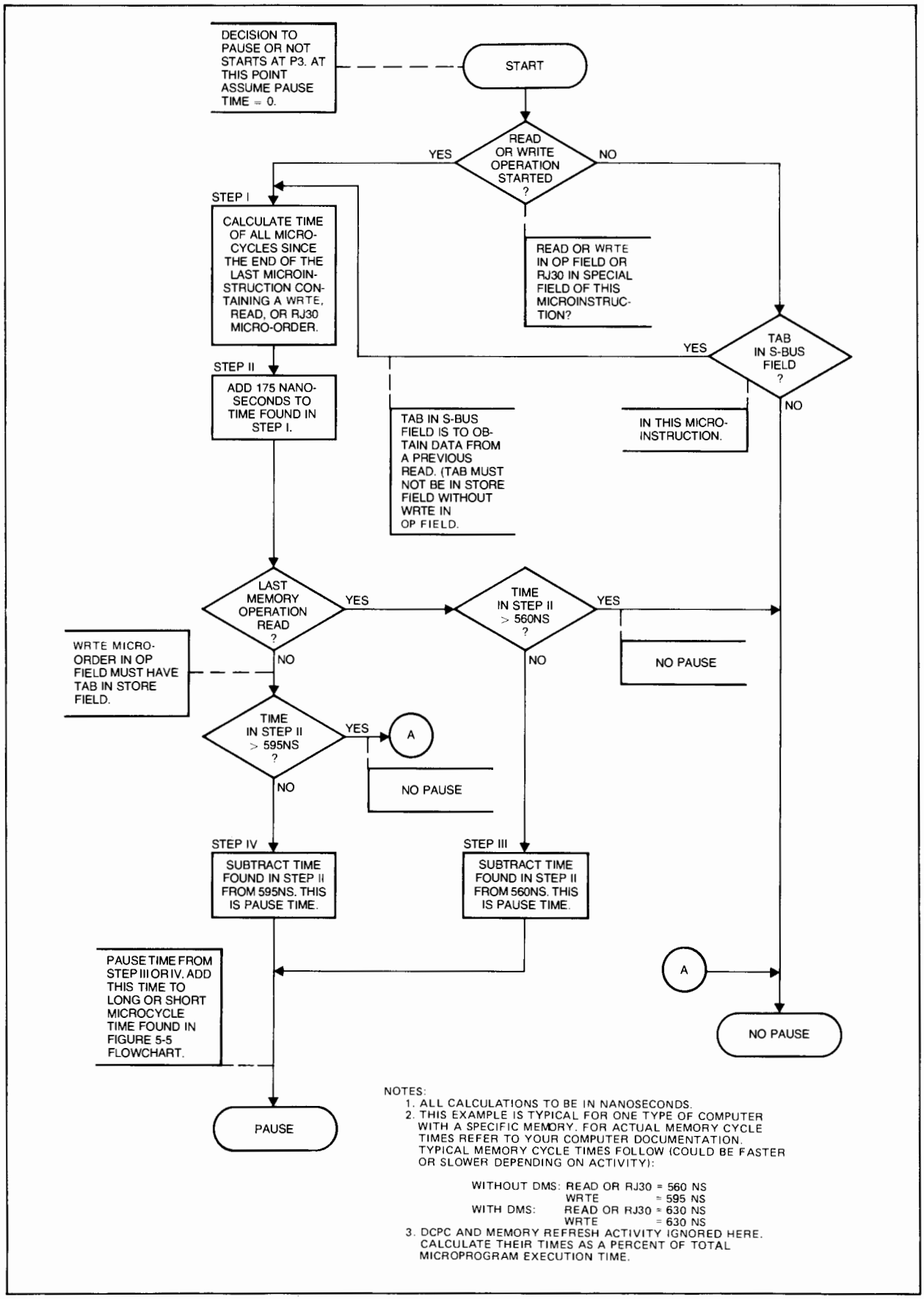
Using figure 5-5 note that START executes in 175 nanoseconds. In FIRST (using figure 5-5), note that although there is a READ in the OP field of this microinstruction (which begins a memory operation) there is not a memory operation *already* in progress; thus, FIRST also executes in 175 nanoseconds. Point A shows where the main memory read cycle timing starts (the request for memory is made at the end of the microcycle). No delays occur for execution of the microinstructions labeled SECOND and THIRD; they each execute in 175 nanoseconds as shown at point B, while main memory is still busy executing the read request. (Note that these two microinstructions do not contain micro-orders that would cause a freeze.)

Now the microinstruction labeled DATA begins to execute. Figure 5-5 shows that if there is a TAB in the S-bus field while memory is busy, there will be a pause time added to the microcycle. Figure 5-6 can be used to calculate the time as follows. At the first decision point in the flowchart, no READ, or WRTE, or RJ30 micro-order is encoded in *this* microinstruction. Entry is made at step I (figure 5-6 because there is a TAB micro-order encoded in the S-bus of the microinstruction under consideration.

In step I add the execution times for microinstructions labeled SECOND and THIRD which = 350 nanoseconds (point B). In step II the result = 525 nanoseconds. Since the last operation (in the microinstruction labeled FIRST) was a READ, the flowchart in figure 5-6 directs you to step III which when completed provides pause time = 35 nanoseconds in this case. Returning to figure 5-5, the result through P3 = 4 x 35 nanoseconds = 140 nanoseconds. Since microinstruction DATA will be short, P4 and P5 are entered immediately with a resulting total time for this microinstruction = 210 nanoseconds (point D). Microinstruction END will be long (point E) because of the CM branch. You may look at the partial microroutine just illustrated and consider that you can simply subtract the time for all microinstructions executed (before the microinstruction labeled DATA but after the one labeled FIRST) from the memory cycle time and in this case obtain 210 nanoseconds; however, this procedure will not always yield correct results. The next microprogram example illustrates why this is so.

5-14. WRITING TO MEMORY. Consider a write operation to main memory using the following microroutine. For this example, assume the DMS is installed. Also, consider conditions for the microbranch (in microinstruction CHECK) not met and no memory operation in progress as entry is made. Again note that the microroutine in these examples is used only to show timing relationships. Consult section 7 for microprogramming considerations in write operations.

```
                    ALU/
          OP/              MOD/                S-BUS/
LABEL     BRCH      SPCL   COND     STR        ADDRESS     COMMENTS

*                                                           TIME (NS)
                         .
                         .
                         .
ENTER                    INC      X          X           175 <-A          B
WRITE     WRTE      MPCK  PASS     TAB        X           175
CHECK     JMP       CNDX  ALZ      RJS        *+2         175 <-C  ≥630 (SEE TEXT)
GO        READ      RTN   INC      PNM        P           560 <-E
                         .
                         .
                         .
                         (ETC.)
```



7115-18

Figure 5-6. Detailed Pause Time Calculation Flowchart (Using an HP 2102B Memory as an Example)

Microinstructions labeled ENTER and WRITE (point A) both execute in 175 nanoseconds each and the main memory write cycle timing begins at point B. Microinstruction CHECK executes in 175 ns (point C) since branch conditions are not met, then a read from main memory is next attempted. Using the flowcharts in figures 5-5 and 5-6 it can be seen that the calculation for the time shown at point E is made for microinstruction GO as shown below. (The write time at point D is 630 nanoseconds because of the DMS factor.)

105 nanoseconds	time for P1,P2,P3 (from figure 5-5)
245 nanoseconds	add pause time (calculated in figure 5-6)
35 nanoseconds	add for DMS
105 nanoseconds	add for E1,E2,E3 (RTN in SPCL field)
70 nanoseconds	add for P4,P5
560 nanoseconds	total time spent in microinstruction GO.

5-15. SUMMARY

Table 5-1 is a summary of some times used in this section that may be helpful if you are making execution time estimates. With the information presented in this section you should now be able to verify that the following microroutine executes in the noted time. Assume no memory cycle in progress as the microroutine is entered and no DMS activity occurring:

```
                    ALU/
          OP/                          S-BUS/
LABEL     BRCH   SPCL  MOD/   STR      ADDRESS    COMMENTS
                       COND
*                                                  TIME (NS)
                       .
                       .
                       .
START     READ   CLFL  PASS   M        S1         175
                       PASS   L        S2         175
          ENVE         ADD    S3       TAB        385
          READ         PASS   M        S3         175
          IMM          CMLO   L        374B       175
                       ADD    L        S3         175
          ENVE         ADD    S3       TAB        210
          RTN    CNDX  OVFL                       280/175
          RTN    SOV                              280
                       .
                       .
                       .
                       (ETC.)
```

If no overflow, the total time is 1.750 microseconds. If an overflow, the total time is: 1.925 microseconds.

Table 5-1. Summary of Timing Factors

ITEM	TIME	
P period	35	nanoseconds
P4 plus P5	70	nanoseconds
E1 through E3	105	nanoseconds
Short microcycle	175	nanoseconds
Long microcycle	280	nanoseconds
Typical main memory read cycle	560	nanoseconds
Typical main memory write cycle	595	nanoseconds
DMS factor (WRTE)	35	nanoseconds
DMS factor (READ)	70	nanoseconds

Section 6
MAPPING TO THE USER'S MICROPROGRAMMING AREA

MAPPING TO THE USER'S MICROPROGRAMMING AREA

SECTION 6

In order to have operational flexibility using your Computer Series microprogramming facilities you must have an understanding of the methods used to branch from main memory to control memory and then back to your program in main memory when your microprogrammed operation is complete. This section provides information that will enable you to:

- Understand the control memory mapping scheme.
- Link to the user's microprogramming area from your Assembly language (or FORTRAN) program.
- Pass parameters to your microprogram.
- Understand control memory branch address modification (using some of the available micro-orders).
- Return from control memory (making a "normal" exit).
- Pass parameters back to your main memory program.

For this discussion on mapping it will be assumed that your microprograms have already been prepared (using the microassembler and probably the Microdebug Editor) and placed in some facility of control memory (e.g., WCS, FAB, or UCS). Section 8 describes how to assign starting addresses to your microprograms. Various microassembler pseudo-microinstructions, which also exist and are capable of modifying control memory addresses while preparing microprograms, are described in section 8. Section 7 provides information on how to check for and handle interrupts when you are in your microprograms.

Part III in this manual describes methods used to get microprograms into control memory. The methods include creating and installing permanent microprograms and using the "dynamic" microprogramming method (the WCS facility). By using WCS and the WCS related microprogramming support software (DVR36, WLOAD, and the Microdebug Editor), microprograms can be loaded into control memory (WCS) and swapped (or overlayed) with other microprograms.

As is obvious from the above discussion, the information related to passing control in your program from main memory to control memory and back is considerably interrelated. It is important that the concepts of main memory/control memory links be firmly established first. Then, with an understanding of the mapping, parameter passing, and branching techniques described in this section; the interrupt handling and control memory address assignment methods described in sections 7 and 8; and the microprogramming support software used to control WCS; you will have complete microprogram address manipulation and transfer capability.

6-1. CONTROL MEMORY MAPPING METHOD

As mentioned in section 2, the Control Processor is always in control of the computer and the base set microroutines cause the read operations to occur for all instructions (and data) from main memory. In this manner, all 16-bit instructions are placed in the Instruction Register (IR) and decoded. (Data can be considered as "parameters" which can be loaded into the desired and appropriate registers by your microprogram to later perform certain operations; parameter passing will be discussed later in this section). For instructions, the process of decoding the Instruction Register bits determines which control memory address (which microprogram) is called by the instruction received from main memory. The decoding process (mapping method) discussion in this paragraph is at the level you will need for "normal" user microprogramming and the instruction codes you may use to map to particular control memory entry points are defined. If you are planning an extensive microprogramming effort, however, you may be interested in the details of the mapping process contained in appendix C.

6-2. SOFTWARE ENTRY POINTS

Recall that the control memory map in figures 2-3 and 2-4 shows all modules of control memory, their module boundary addresses, and whether or not the module has available "software entry points". The software entry points are the bit patterns which, when placed in the Instruction Register (from your main memory program), will cause the Control Memory Address Register to be finally loaded (through mapping) with a desired control memory module *entry* address.

The hardware/firmware combination in the Control Processor is the facility that imposes restrictions on control memory software entry points. By using the proper instruction codes you may (with discretion) map to any *obtainable* location. However, as mentioned in section 2, certain areas of control memory may be used for HP microprograms and/or microprogrammed computer enhancements. Thus, the use of descretion in accessing control memory. It is recommended that you restrict your use of the software entry point instruction codes to those set aside for entrance into the user's microprogramming area. The instruction codes for most software entry points (excluding modules 0 and 1 of the base set) will be defined shortly and the instruction codes for entrance into the user's area (the primary concern of this section) will be identified.

Once in a control memory module, you may have microinstructions that branch to any control memory location. Again, the use of discretion is implied since the areas shown in figure 2-3 reserved for HP microprograms and/or microprogrammed accessories may be filled with microprograms. But you could, for example, branch and use a microroutine of the base set then return to your own microprogram if you prepare your microprogram correctly.

6-3. THE USER INSTRUCTION GROUP

For the purposes of mapping to the "user" areas, the Computer base set has a reserved block of binary codes called the User Instruction Group (UIG). These codes (UIG instructions) permit you to link Assembly language routines to your microprograms. The key to the UIG is the upper byte (most significant bits) of the calling code which must have the format:

105xxx (bit 11 of the IR = 1)

or:

101xxx (bit 11 of the IR = 0).

where:

xxx equals values to be defined in the following paragraphs.

Control memory module selection is determined by the value of bits 8 through 4 in the Instruction Register (still part of the coded UIG instruction). In general, a secondary index (composed of bits 3 through 0) directly determines which address in the first 16 locations of the selected module will be used for entry.

Bit 11 in the third octal digit (105xxx or 101xxx) of the UIG instruction in the IR can be used as an indicator (for your microprograms) by micro-orders which test the Instruction Register data. For example, the Store field and S-bus field micro-order CAB tests IR bit 11 to select either the A- or B-register.

The value of bits 8 through 4 of the UIG instruction in the IR is not directly translatable into a control memory module number but these bits help determine the address of branches in the control memory base set Primary Mapping Table, which in turn direct a branch to the desired module.

6-4. HP RESERVED UIG CODES. As mentioned in paragraph 6-2, modules of control memory have software entry points assigned, but modules 0 and 1 of the base set must be disregarded in this discussion since codes for access to those modules do not fall within the UIG. All modules of control memory that are accessible through the UIG instructions are shown in table 6-1. This table is arranged in UIG instruction (binary code) order. The modules these codes map to are shown along with the control memory entry addresses.

As can be seen from table 6-1, all modules below module 46 accessible with UIG instructions have been reserved for HP use and are not recommended for normal user microprogramming. Also, as noted in the table, modules 2, 3, and 32 have a mapping situation that is slightly different than the one used for modules with a single UIG module selection code (one combination of bits 8 through 4). This multiple entry point mapping is used only for modules reserved for HP use (base set or HP accessories) and it will not be discussed in this manual. The module selection codes (bits 8 through 4) briefly mentioned in paragraph 6-3 are further discussed in appendix C. Refer to the appendix if you require more information about the module selection codes or the HP reserved area.

To avoid access to the HP reserved area do not use the following UIG instruction (binary codes) for main memory to control memory linking:

105000 through 105137

or

101 (or 105) { 200 through 437
460 through 477
700 through 777

Table 6-1. Control Memory User Instruction Group Software Entry Point Assignments

RANGE OF UIG INSTRUCTION (MAIN MEMORY) VALUES USED (OCTAL)	MODULE MAPPED TO	CONTROL MEMORY ENTRY POINTS (RANGE OF ADDRESSES) (OCTAL) (NOTE 2)	USE
105000-105137	3	01xxx (NOTE 1)	Floating Point
105140-105157	60	36000-36017	User area
105160-105177	62	37000-37017	User area
101 (or 105) 200-217	34	21000-21017	FFP
101 (or 105) 220-237	35	21400-21417	FFP
101 (or 105) 240-257	36	22000-22017	EMA
101 (or 105) 260-277	8	04000-04017	HP Reserved
101 (or 105) 300-317	38	23000-23017	DS/1000
101 (or 105) 320-337	40	24000-24017	SIS (NOTE 3)
101 (or 105) 340-357	16	10000-10017	HP Reserved
101 (or 105) 360-377	42	25000-25017	SIS (NOTE 3)
101 (or 105) 400-417	4	02000-02017	HP Reserved
101 (or 105) 420-437	6	03000-03017	HP Reserved
101 (or 105) 440-457	46	27000-27017	User area
101 (or 105) 460-477	12	06000-06017	HP Reserved
101 (or 105) 500-517	47	27400-27417	User area
101 (or 105) 520-537	48	30000-30017	User area
101 (or 105) 540-557	49	30400-30417	User area
101 (or 105) 560-577	50	31000-31017	User area
101 (or 105) 600-617	56	34000-34017	User area
101 (or 105) 620-637	57	34400-34417	User area
101 (or 105) 640-657	58	35000-35017	User area
101 (or 105) 660-677	59	35400-35417	User area
101 (or 105) 700-737	32	20xxx (NOTE 1)	DMS
101 (or 105) 740-777	2	01xxx (NOTE 1)	EIG

NOTES:
1. xxx signifies last three digits for the entry address. See appendix C for details.
2. All modules except 2, 3, and 32 have 16 entry points. See appendix C.
3. Available in F-Series only.

6-5. USER AREA UIG CODES. Modules 46 through 63 comprise the primary user's microprogramming area. (Modules 4 through 31 for E-Series and 27 through 31 for F-Series are also addressable once in control memory.) The modules in the user's area that have UIG module selection codes assigned are designated as user area modules in table 6-1. As apparent from the table, 11 of the 18 modules in the range 46 through 63 are directly accessible. Entry to other control memory modules will require an extra branch after reaching control memory.

As can also be seen in table 6-1, each module has 16 possible control memory software entry points provided by the UIG instruction secondary index (UIG instruction bit 3 through 0 combination). The secondary index directly determines which control memory address (of the first 16 locations in the selected module) will be loaded into the Control Memory Address Register. The ranges of values for UIG instructions you should use to access the respective control memory addresses are summarized below. Since each module may be entered at 16 different locations, 176 direct entry points into the recommended user's microprogramming area are available.

Summary of UIG instructions (binary codes) you can use:

105140 through 105177

and

101 or 105 { 440 through 457
500 through 677

6-6. USER'S AREA MAPPING EXAMPLE

A typical example of mapping to the user's microprogramming area through the base set using a recommended UIG instruction is discussed below. Information about the proper procedure to use in main memory and for returning to main memory is also included. The depth of the discussion should be sufficient for your normal microprogramming needs.

6-7. MAIN MEMORY/CONTROL MEMORY LINKAGE. Suppose that your main memory program has a UIG instruction 105602 (octal) written into a particular location designated "I". The UIG instruction may or may not have address pointers and/or operands in main memory locations I + 1, I + 2, etc. For example:

MAIN MEMORY

Location	Contents
I	105602
I + 1	.
I + 2	.
.	.
.	.
.	.

During execution, UIG instruction 105602 maps to control memory location 34002 as follows. The base set Fetch microroutine completes the read and IR store operation (as described in paragraph 2-16) for your 105602 UIG instruction and begins the mapping procedure by executing these microinstructions:

CONTROL MEMORY
(Fetch Microinstructions, start at CM location 00000)

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			.			
FETCH	READ	FTCH	PASS	IRCM	TAB	IR = 105602, L = 0
		JTAB	INC	PNM	P	M = I + 1, P = I + 2
			.			
			.			
			.			

The JTAB micro-order indexes the upper eight bits of the 105602 UIG instruction (in the IR) through the Control Processor Jump Tables to the following microinstruction in the base set's microroutines:

```
                ALU/
        OP/     MOD/            S-BUS/
LABEL   BRCH    COND    STR     ADDRESS     COMMENTS

                .
                .
                .
MAC1    JMP     J74             MACTABL1    BEGIN MAPPING TO USER AREA
                .
                .
                .
```

As can be seen from this example, this microinstruction branches to the control memory address at label "MACTABL1" (still in the base set) but the J74 Special field micro-order indexes the branch, making a branch address modification, by replacing bits in this microinstruction branch address field with bits from the Instruction Register (refer to table 4-1 for the explanation of J74). This index actually serves as the UIG module selection code, described in paragraphs 6-3 and 6-4, and causes entry at a particular address in the base set's Primary Mapping Table. At the indicated address in the Primary Mapping Table, another control memory branch is directed. This branch is made to the desired module (in this case CM address 34000) by the appropriate microinstruction as follows:

```
                ALU/
        OP/     MOD/            S-BUS/
LABEL   BRCH    COND    STR     ADDRESS     COMMENTS

                .
                .
                .
MACTABL1 JMP                    23420B          . . .
                .
                .
                .
        JMP     RJ30            34000B      COMPLETE MAPPING TO USER AREA
                .
                .
                .
```

Note that the branch to control memory address 34000 is modified by an RJ30 Special field micro-order. The RJ30 implements the secondary index and causes the Control Memory Address Register to be loaded with the final module entry point address (one of the first 16 locations). In this case, since the UIG instruction is 105602, the microinstruction's branch address field bits are replaced with the Instruction Register bits that will cause entry to be made at control memory address 34002. (Refer to table 4-1 for the explanation of RJ30). The RJ30 micro-order simultaneously starts a read operation from main memory location I + 1. (See the Fetch microroutine previously described.)

Upon reaching the user microprogramming area (at address 34002) the following situation exists:

IR = 105602,
L = 0, (FTCH cleared the L-register)
P = I + 2,
M = I + 1, and a READ of main memory location I + 1 is in progress.

Microinstructions at your control memory entry points should usually have been previously prepared to cause an additional branch to the control memory address where the desired microroutine begins. Typically the first 16 locations in a user module are set up with unconditional branches (word type IV) to the actual microroutines as follows (module 56 used in this example):

```
                                ALU/
                   OP/          MOD/          S-BUS/
LOCATION   LABEL   BRCH   SPCL  COND   STR    ADDRESS    COMMENTS

                                  .
                                  .
                                  .
34000              JMP                       INST00MC   ENTRY POINT 1
34001              JMP                       INST01MC   ENTRY POINT 2
34002              JMP                       INST02MC   ENTRY POINT 3
34003              JMP                       INST03MC   ENTRY POINT 4
                                  .
                                  .
                                  .
34007              JMP                       INST07MC   ENTRY POINT 8
34010              JMP                       INST08MC   ENTRY POINT 9
                                  .
                                  .
                                  .
34017              JMP                       INST15MC   ENTRY POINT 16
34020      INST02MC                     S3   TAB        BEGIN MICROROUTINES
                                  .
                                  .
                                  .
                   READ   RTN     INC   PNM   P          EXIT
```

In this example the microinstruction at the entry address causes a branch to control memory location 34020 where the actual microroutine begins.

The TAB micro-order (location 34020) is used to obtain the results of the RJ30 initiated main memory read operation that occurred while in the base set Primary Mapping Table. In this example the data is stored in S3. This data could be a parameter address passed from your main memory program. The data obtained by this RJ30 initiated read operation must be taken from the T-register while at the first microinstruction in your microroutine, or at the latest, during execution of the next microinstruction (refer to table 4-1 for the explanation of a READ micro-order). If desired, the results of the RJ30 initiated read operation may be ignored.

6-8. ASSEMBLER PROCEDURE. An Assembly language procedure for invoking a microprogram and passing parameters is discussed below. Paragraph 6-11 provides some additional information. The basic concepts of invoking microprograms and passing parameters should be evident from the information presented here.

Basically, the microprogram is invoked and parameters are passed using an Assembly language procedure such as follows:

```
ASMB,L
        NAM  TEST,7
        ENT  TEST,MACRO
        EXT  ISC,NMBR,IBUF
TEST    NOP
MACRO   OCT  105603     MICROPROGRAM OP CODE
        DEF  *+4        RETURN ADDRESS, ALSO FTN COMPATIBILITY
        DEF  ISC(,I)    SELECT CODE
        DEF  NMBR(,I)   DATA COUNT
        DEF  IBUF(,I)   DATA BUFFER
        JMP  TEST,I
        END
```

As can be seen from the above, a UIG instruction (as described in preceding paragraphs) appears in an OCT statement. This is used at the point in the Assembly language source program where the branch is to occur. The value to be inserted should be OCT 101xxx (or 105xxx) (where xxx is in the range shown in table 6-1) to properly map to the desired control memory module address. If parameters are to be passed, they are usually defined as constants (via DEF or OCT statements) immediately following the OCT statement as seen in the example above. The microprogram procedures for accessing parameters are presented in the following paragraph.

6-9. PARAMETER PASSING. The following two examples of microprograms show how to access parameters in main memory and resolve indirect main memory references. The initialization portion of each microprogram (microassembler control commands and pseudo-instructions) will be described in later sections. The primary thing you should observe in these examples is the method used to handle parameters. Pay particular attention to the P- and M-register adjustments. Remarks and explanatory notes are included in the microprograms. Note that any line beginning with an asterisk is a comment. The interrupt handling methods shown in these microprograms will be described in section 7.

EXAMPLE 1: ACCESSING A PARAMETER LIST FROM A MICROPROGRAM

```
PAGE 0002 RTE MICRO-ASSEMBLER REV.A 760805

                          MICMXE,L                                  21MX E-SERIES
                          $CODE=MPOBJ,REPLACE                       OBJECT TO DISC
                          INDIRECT EQU  34355B                      USER WRITTEN
                          *                                         INDIRECT
                          *                                         MICROPROGRAM
                          *                                         (SEE EXAMPLE 2)
                                   ORG  34003B                      105603 => 34003
 34003  327 001407                 JMP                 INST03MC     SAVE ENTRY
                          *                                           POINTS
                          * THIS MICROPROGRAM IS AN EXAMPLE OF HOW TO
                          * RETRIEVE MAIN MEMORY PARAMETERS AND ADDRESSES
                          *
                          * A USER WRITTEN MICROSUBROUTINE (SEE EXAMPLE 2)
                          * WILL BE USED TO RESOLVE INDIRECT ADDRESSES
                          *
                          * INITIALIZE THE CNTR
                          *   THE USER WRITTEN INDIRECT MICROPROGRAM (EXAMPLE 2),
                          *   IF INTERRUPTED, USES THE CNTR TO ADJUST P (I.E.
                          *   SET P TO MAIN MEMORY ADDRESS + 1 OF THE
                          *   MICROPROGRAM OP CODE)
                                   ORG  34030B
 34030  343 176547        INST03MC IMM       LOW  CNTR 377B       CNTR = -1
                          *
                          * GET PARAMETERS:
                          *   SELECT CODE, DATA COUNT, BUFFER ADDRESS
 34031  227 174725                 READ DCNT INC  PNM  P          GET SELECT CODE
 34032  307 016647                 JSB                 INDIRECT   RESOLVE ADDR
 34033  010 000507                              L    TAB        L = SELECT CODE
                          *
 34034  227 174725                 READ DCNT INC  PNM  P          GET DATA COUNT
 34035  307 016647                 JSB                 INDIRECT   RESOLVE ADDR
 34036  353 007123                 IMM  L4    CMLO S3   303B     (SEE NOTE 1)
 34037  010 001147                              S4   TAB        S4 = DATA COUNT
                          *
 34040  227 174725                 READ DCNT INC  PNM  P          GET BUFFER ADDR
 34041  010 145107                           IOR  S3   S3        (SEE NOTE 1)
 34042  307 016647                 JSB                 INDIRECT   RESOLVE ADDR
 34043  010 033207                              S5   M          S5 = BUFFER ADDR
                          *
                          * NOTE 1. ONE NON-FREEZABLE MICROINSTRUCTION MAY
                          *          PRECEDE AND/OR FOLLOW THE JSB INDIRECT'S
                          *
 34044  227 174700                 READ RTN  INC  PNM  P          START FETCH FOR
                          *                                       NEXT MAIN MEMORY
                          *                                       INSTRUCTION
                                   END

END OF PASS 2: NO ERRORS
```

EXAMPLE 2: RESOLVING INDIRECT MAIN MEMORY REFERENCES

```
PAGE 0002 RTE MICRO-ASSEMBLER REV.A 760805

0001                        MICMXE.L                                    21MX E-SERIES
0002                        $CODE=INDOBJ,REPLACE                        OBJECT TO DISC
0003                        HORI      EQU  6                            BASE SET HALT-
0004                        *                                           OR-INTERRUPT
0005                        *                                           MICROROUTINE
0006                                  ORG  34355B
0007                        *
0008                        * THIS IS AN EXAMPLE OF A USER WRITTEN MICROSUBROUTINE
0009                        * THAT RESOLVES INDIRECT MAIN MEMORY REFERENCES
0010                        *
0011                        * EACH INDIRECT LEVEL REQUIRES AN ADDITIONAL MEMORY
0012                        * CYCLE
0013                        * AT ENTRY,
0014                        *   THE CALLING PROGRAM MUST HAVE INITIALIZED THE CNTR
0015                        *   (SEE EXAMPLE 1) SO THAT THIS MICROSUBROUTINE, IF
0016                        *   INTERRUPTED, WILL CORRECTLY ADJUST P (I.E SET P TO
0017                        *   MAIN MEMORY ADDRESS + 1 OF THE MICROPROGRAM OP
0018                        *   CODE) BEFORE JUMPING TO HORI. THE BASE SET
0019                        *   HALT-OR-INTERRUPT MICROROUTINE
0020                        *
0021                        * AT EXIT,
0022                        *   THE FINAL (DIRECT) MAIN MEMORY ADDRESS WILL HAVE
0023                        *   BEEN DETERMINED, AND A READ OF THE FINAL ADDRESS
0024                        *   WILL BE IN PROGRESS
0025                        *
0026                        * FOR THE FIRST THREE INDIRECT LEVELS, INTERRUPTS
0027                        * ARE NOT CHECKED
0028                        *
0029                        * AFTER THE THIRD, OR ANY SUCCESSIVE, INDIRECT LEVEL
0030                        * INTERRUPTS ARE CHECKED FOR AND SERVICED
0031                        *
0032 34355  230 000647  INDIRECT READ            M    TAB        INDIRECT ?
0033 34356  367 140002          RTN  CNDX AL15 RJS                NO.RTN
0034                        *
0035 34357  230 000647          READ             M    TAB        INDIRECT ?
0036 34360  367 140002          RTN  CNDX AL15 RJS                NO, RTN
0037                        *
0038 34361  230 000643  NEXT    READ ION         M    TAB        ION. INDIRECT ?
0039 34362  367 140002          RTN  CNDX AL15 RJS                     NO, RTN
0040 34363  323 157042          JMP  CNDX HOI  RJS  NEXT        INTERRUPT OR
0041 34364  336 057042          JMP  CNDX NSNG RJS  NEXT        INSTR STEP?
0042                        *                                     NO, NEXT ADDR
0043 34365  010 026507                           L    CNTR       YES, ADJUST P
0044 34366  320 000307          JMP                  HORI       EXIT TO HORI
0045                            END

END OF PASS 2: NO ERRORS
```

Parameters may be passed back to your main memory programs by writing the values (loaded into the T-register) into the desired locations (address loaded into the M-register) since you have direct control of the registers while you are executing microinstructions in control memory.

6-10. CONTROL MEMORY/MAIN MEMORY LINKAGE. It is the microprogrammers responsibility to have stored and/or adjusted the values in the P, M, and other applicable registers (using the appropriate micro-orders) when entering a microprogram so that the respective registers may be restored with the desired values before returning control to main memory. When preparing to exit a microprogram and return to the base set Fetch microroutine, the following must be accomplished to properly interface with the next main memory instruction. Assume that a main memory location designated "J" contains the next instruction. Upon microprogram completion you must ensure:

P = J + 1
M = J, and a read operation of location J starts within three microinstructions before microprogram exit.

Note that the last example in paragraph 6-7 and the last part of microprogram EXAMPLE 1, both end in the manner stated above.

6-11. SOME MAIN MEMORY PROGRAM PROCEDURES

Information on another Assembly language instruction and a FORTRAN procedure that can be used to invoke microprograms is included in the following paragraphs. Further information on Assembly language procedures can be found in the *RTE Assembler Reference Manual,* part no. 92060-90005 or the *RTE IV Assembler Reference Manual,* part no. 92067-90003. Examples of FORTRAN procedures are included in parts III and IV of these manuals. Also refer to the *RTE FORTRAN IV Reference Manual,* part no. 92060-90023. For information on other languages, refer to the appropriate manuals listed in the Table 3-3 in the preface of this manual.

6-12. THE MIC PSEUDO-INSTRUCTION

An Assembly language program can also call a microprogram with a mnemonic code which has been assigned earlier in the program. That is, with a MIC pseudo-instruction, you can define a source language instruction which passes control and a series of parameter *addresses* to a microprogram. In this use of the MIC instruction, a UIG instruction (binary code) is assigned to a mnemonic so that whenever the mnemonic appears, the code is written into that location in the assembled program. The number of parameters is also specified in the following format for the MIC pseudo-instruction:

MIC *opcode, fcode, pnum* *comments*

where:

opcode = any three-character alphabetic mnemonic

fcode = a UIG instruction (octal) from table 6-1

pnum = the number of associated parameter addresses (zero to seven) (may be an expression which generates an absolute result).

NOTE

All three operands (*opcode, fcode,* and *pnum*) must be supplied in the MIC pseudo-instruction in order for the specified instruction to be defined. If *pnum* is zero, it must be expressly declared as such (*not* omitted).

This Assembly language pseudo-instruction provides you with the ability to define your UIG instructions with mnemonics, but the MIC declaration must appear before the three-character alphabetic mnemonic is used. When the "newly" assigned user-defined instruction is used later in your Assembly language source program, the specified number of parameter addresses (*pnum*) are supplied in the operand field separated from one another by spaces. These parameter addresses can be any addressable values, relocatable and/or indirect. If it is desired to pass additional parameters to a microprogram beyond those pointed to by the user-defined instruction, they must be defined as constants (via OCT or DEF statements) immediately following each use of the user-defined instruction.

6-13. PARAMETER ASSIGNMENT EXAMPLE. Assume that a total of three parameters are to be passed to a microprogram. Suppose the values of the first two parameters are in main memory locations designated ISC and NMBR and that the value for the third parameter is in a memory location *pointed to* by IBUF. A UIG instruction for your microprogram could be 105602. In this case the Assembly language source language statement would be written:

```
MIC     MIO,105602B,3
```

After this above statement in the source, you may use the MIO statement in your source program whenever it is necessary to pass control to a particular microprogram with the entry point at control memory address 34002 by using the following:

```
MIO     ISC NMBR IBUF,I
```

An example of a short but complete Assembly language program illustrating some of the procedures outlined thus far appears in the next paragraph.

6-14. EXAMPLE MIC PSEUDO-INSTRUCTION USE. The Assembly language use principles are summarized in the following example. Note that the two MIC instructions are declared first. One has no parameter addresses to pass, the other has four. SRT could be a sort microroutine and MIO a microprogrammed I/O operation. In source statement sequence number 0014, designation *+5 is used to limit the list and make the program FORTRAN callable. ISC is the select code, NMBR the count, and IBUF a reserved data buffer (5 locations).

EXAMPLE 3: MIC PSEUDO-INSTRUCTION USE

```
PAGE 0002 # 01

0001                    ASMB,L
0002  00000                   NAM MIC PSEUDO INSTRUCTION USAGE
0003*
0004                          MIC SRT,105600B,0
0005*
0006                          MIC MIO,105602B,4
0007*
0008  00000 000000  START NOP
0009*
0010*
0011  00001 105600  SORT  SRT
0012*
0013*
0014  00002 105602  MCIO  MIO *+5 ISC NMBR IBUF
      00003 000007R
      00004 000013R
      00005 000014R
      00006 000015R
0015*
0016*
0017                          EXT EXEC
0018  00007 016001X         JSB EXEC
0019  00010 000012R         DEF *+2
0020  00011 000012R         DEF RC
0021  00012 000006  RC    DEC 6
0022*
0023  00013 000016  ISC   OCT 16
0024  00014 000005  NMBR  DEC 5
0025  00015 000000  IBUF  BSS 5
0026                          END START
**  NO ERRORS*
```

6-15. CALLING MICROPROGRAMS FROM FORTRAN

Treating a microprogram as an external subroutine is a typical way to invoke a microprogram from FORTRAN. The process (using the example MIO microprogram) is shown below followed by explanations.

```
FTN4,L,M
        SUBROUTINE FTNMP (ISC, NMBR, IBUF)
        DIMENSION IBUF (1)
                  .
                  .
                  .
        CALL MIO (ISC, NMBR, IBUF)
                  .
                  .
                  .
        END
        END$
```

The M in the compiler control statement provides mixed mode operation and expansion to Assembly language. The CALL MIO statement expands to a JSB MIO followed by a series of parameter addresses as follows:

```
JSB   MIO
DEF   *+4
DEF   00000,I
DEF   00001,I
DEF   00002,I
```

The load time JSB replace routine would appear as follows:

```
ASMB,L
        NAM RPLCE
MIO     RPL 105602
        END
```

The MIO RPL 105602 statement above alerts the RTE relocating loader that all external references to MIO are to be replaced with 105602 and, if loaded with the program shown first in this paragraph, causes the RTE relocating loader to substitute the required microprogram UIG instruction (105602), for the JSB MIO. In this way, the FORTRAN program accesses the microprogram directly at execution time.

6-16. SUMMARY

Equipped with knowledge gained through information in this section, you should have no trouble planning where you want your microprograms placed in control memory. You should have a good understanding of linking between main memory and control memory. The concept of control memory branching has been presented so that, if necessary, you may also use the J74 and RJ30 micro-orders for CM branch address modification in your microroutines. The concepts of parameter passing should also be clear.

Section 7
MICROPROGRAMMING CONSIDERATIONS

MICROPROGRAMMING CONSIDERATIONS

SECTION 7

Some key points that you will want to be aware of when writing microprograms are presented in this section. The assumption is that you will refer to section 4 for complete descriptions of micro-orders, but the additional considerations in this section include:

- The techniques to use for microprogrammed read, write, and arithmetic operations.
- Microprogramming with the Memory Protect or Dual Channel Port Controller (DCPC) installed.
- Microprogrammed Input/Output operations.
- Microprogramming with the Dynamic Mapping System installed.

Some guidelines for writing IBL loaders are also included.

7-1. READ AND WRITE CONSIDERATIONS

Microprogrammed main memory read and write operations are easily implemented and will be successful when the guidelines outlined below are followed. Conditionally valid and invalid methods of using the READ and WRTE micro-orders are also discussed in paragraph 7-5.

7-2. TYPICAL READ OPERATIONS

Load the M-register before or during microinstructions containing READ in the OP field. Do not modify the M-register until at least two microinstructions after the READ (See the information in this paragraph on reading the A- and B-registers with a TAB micro-order.). A simple READ with the M > 1 is performed as follows:

```
                ALU/
        OP/             MOD/                S-BUS/
LABEL   BRCH    SPCL    COND    STR         ADDRESS     COMMENTS

                        .
                        .
                        .
        READ                    M           S3          175 NS
                                S4          TAB         560 NS
                        .
                        .
                        .
```

The T-register contents must be placed on the S-bus no later than two microinstructions after a READ is specified, because the T-register is disabled by the Main Memory Section after the second microinstruction is executed. Microinstructions may be used between READ and TAB. When using one microinstruction between READ and TAB, the microroutine may appear as follows:

```
                    ALU/
          OP/              MOD/          S-BUS/
LABEL     BRCH      SPCL   COND   STR    ADDRESS    COMMENTS

                           .
                           .
                           .
          READ                    M      S3         175 NS
                           INC    S3     S3         175 NS
                                  S4     TAB        560 - 175 = 385 NS
                           .
                           .
                           .
```

Note that if a DCPC is active, freezable microinstructions (e.g., IOG) may not be used between READ and TAB. Also, no more than two microinstructions may be executed between READ and TAB. If there is no DCPC activity, neither restriction applies. When using two microinstructions, the microroutine may appear as follows.

```
                    ALU/
          OP/              MOD/          S-BUS/
LABEL     BRCH      SPCL   COND   STR    ADDRESS    COMMENTS

                           .
                           .
                           .
          READ                    M      S3         175 NS
                           INC    S3     S3         175 NS
          IMM              LOW    L      0          175 NS
                           AND    S4     TAB        560 - (175 x 2) = 210 NS
                           .
                           .
                           .
```

For utilizing main memory address 00 as the A-register, use the following microinstructions:

```
                    ALU/
          OP/              MOD/          S-BUS/
LABEL     BRCH      SPCL   COND   STR    ADDRESS    COMMENTS

                           .
                           .
                           .
                           ZERO   S3
          READ                    M      S3         175 NS,AAF=1, READ INHIBITED
                                  S4     TAB        175 NS, S4 =A-REGISTER
                           .
                           .
                           .
```

For utilizing main memory address 01 as the B-register, use the following microinstructions:

```
                    ALU/
        OP/         MOD/        S-BUS/
LABEL   BRCH  SPCL  COND  STR   ADDRESS   COMMENTS

                    .
                    .
                    .
        IMM         CMLO  S3    376B      S3 = 1
        READ              M     S3        175 NS, BAF = 1,READ INHIBITED
                          S4    TAB       175 NS, S4 = B-REGISTER
                    .
                    .
                    .
```

If reading main memory location 00:

```
                    ALU/
        OP/         MOD/        S-BUS/
LABEL   BRCH  SPCL  COND  STR   ADDRESS   COMMENTS

                    .
                    .
                    .
                    ZERO  S3
        READ  PRST        M     S3        175 NS, PRST CLEARS AAF
                          S4    TAB       560 NS, S4 = CONTENTS OF MAIN
*                                             MEMORY LOCATION 0
                    .
                    .
                    .
```

If reading main memory location 01:

```
                    ALU/
        OP/         MOD/        S-BUS/
LABEL   BRCH  SPCL  COND  STR   ADDRESS   COMMENTS

                    .
                    .
                    .
        IMM         CMLO  S3    376B      S3 = 1
        READ  PRST        M     S3        175 NS, PRST CLEARS BAF
                          S4    TAB       560 NS, S4 = CONTENTS OF MAIN
*                                             MEMORY LOCATION 1
                    .
                    .
                    .
```

Memory address 00 and 01 may be written into (refer to paragraph 7-3 by using the Special field micro-order PRST one microinstruction before the TAB micro-order is used. In read or writes the main rule is that PRST precede the TAB micro-order by one microinstruction. Note that (see the last two microroutines) main memory locations 00 and 01 may be used for Hewlett-Packard generated microroutines; therefore, the use of main memory locations 00 and 01 is not recommended.

Microprogrammed successive READ's may appear as follows but note that if two READ's are coded without an intervening TAB, the result of the first READ is lost.

```
                    ALU/
          OP/       MOD/          S-BUS/
LABEL     BRCH SPCL COND  STR     ADDRESS   COMMENTS

                    .
                    .
                    .
          READ              M     S3        175 NS
          READ              M     TAB       560 NS
                            M     TAB       560 NS
                    .
                    .
                    .
```

If the M-register is modified between READ and TAB, the decision between the A-register, B-register, and main memory may be made incorrectly. For example:

```
                    ALU/
          OP/       MOD/          S-BUS/
LABEL     BRCH SPCL COND  STR     ADDRESS   COMMENTS

                    .
                    .
                    .
          IMM       CMLO  S4      376B      S4 = 1
                    ZERO  S3
          READ              M     S3        READ A-REGISTER, AAF = 1
                            M     S4        M = 1, BAF = 1, AAF = 0
                            S5    TAB       S5 = B-REGISTER, NOT A-REGISTER
                    .
                    .
                    .
```

7-3. TYPICAL WRITE OPERATIONS

Load the T-register with data to be written to main memory in the same microinstruction that contains the WRTE micro-order or the DCPC could alter the T-register before the WRTE is executed. Do not alter the T-register unless initiating WRTE, since the T-register is internal to the Main Memory section and is used by both the CPU and the Dual Channel Port Controller (DCPC). The T-register is not intended to be used as a general purpose register, but to be used only in referencing main memory. A simple write operation with M > 1 is accomplished as follows:

```
                    ALU/
          OP/       MOD/          S-BUS/
LABEL     BRCH SPCL COND  STR     ADDRESS   COMMENTS

                    .
                    .
                    .
                            M     S3
          WRTE MPCK         TAB   S4        175 NS
                    .
                    .
                    .
```

For interpreting main memory address 00 as the A-register, use the following microinstructions:

```
                    ALU/
        OP/         MOD/        S-BUS/
LABEL   BRCH  SPCL  COND  STR   ADDRESS   COMMENTS

                    .
                    .
                    .
                    ZERO  S3
                          M     S3        M = 0, AAF = 1
        WRTE  MPCK        TAB   S4        175 NS, A-REGISTER = S4, MAIN
*                                             MEMORY LOCATION 0 UNALTERED
                    .
                    .
                    .
```

For interpreting main memory address 01 as the B-register, use the following microinstructions:

```
                    ALU/
        OP/         MOD/        S-BUS/
LABEL   BRCH  SPCL  COND  STR   ADDRESS   COMMENTS

                    .
                    .
                    .
        IMM         CMLO  S3    376B      S3 = 1
                          M     S3
        WRTE  MPCK        TAB   S4        175 NS, B-REGISTER = S4, MAIN
*                                             MEMORY LOCATION 0 UNALTERED
                    .
                    .
                    .
```

Writing into main memory location 00 is accomplished as follows:

```
                    ALU/
        OP/         MOD/        S-BUS/
LABEL   BRCH  SPCL  COND  STR   ADDRESS   COMMENTS

                    .
                    .
                    .
                    ZERO  S3
              PRST        M     S3        PRST CLEARS AAF
        WRTE  MPCK        TAB   S4        175 NS, MEMORY LOCATION 0 = S4,
*                                             A-REGISTER UNALTERED
                    .
                    .
                    .
```

Writing into main memory location 01 is accomplished as follows:

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS    COMMENTS

                        .
                        .
                        .
          IMM           CMLO   S3     376B       S3 = 1
                 PRST           M      S3         PRST CLEARS BAF
          WRTE   MPCK           TAB    S4         175 NS, MAIN MEMORY LOCATION 1
*                                                   = S4, B-REGISTER UNALTERED
                        .
                        .
                        .
```

Note that (see the last two microroutines) main memory locations 00 and 01 may be used for Hewlett-Packard generated microroutines; therefore, using main memory locations zero and one is not recommended.

Microprogrammed successive WRTE's may appear as follows:

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS    COMMENTS

                        .
                        .
                        .
                                M      S3
          WRTE   MPCK           TAB    S4         175 NS
                                M      S5         175 NS
          WRTE   MPCK           TAB    S4         595-175 = 420 NS
                        .
                        .
                        .
```

In all the WRTE examples above, MPCK checks the M-register, which must be loaded in a microinstruction preceding (not necessarily immediately) the MPCK. To write into protected main memory, omit MPCK.

CAUTION

Writing into protected main memory must be done with caution because of the possibility of crashing the system environment.

After the execution of a microinstruction containing a WRTE, the 595 nanoseconds needed to write into main memory does not extend succeeding microinstructions unless they attempt to access main memory before 595 nanoseconds has elapsed.

7-4. USE OF MPCK

In an active DCPC environment, the use of the MPCK micro-order in a microinstruction containing a WRTE micro-order ensures that the Memory Protect check will be made correctly. The Store field of a microinstruction with READ and MPCK micro-orders must not contain M, PNM, or IRCM because this will result in an erroneous Memory Protect check. A correct sequence of microinstructions might appear as follows:

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS    COMMENTS

                        .
                        .
                        .
                               M      S3         M = ADDRESS TO BE WRITTEN INTO.
          WRTE   MPCK          TAB    S4         MPCK AS USED HERE WILL CORRECTLY
*                                                    CHECK FOR A MEMORY PROTECT
*                                                    VIOLATION.
          READ                 M      S5         MPCK AS USED HERE WILL CORRECTLY
                 MPCK                               CHECK FOR A MEMORY PROTECT
*                                                    VIOLATION.
                        .
                        .
                        .
```

7-5. CONDITIONAL AND INVALID OPERATIONS

The READ/WRTE sequence shown below is conditionally valid. That is, if there is no DCPC activity the sequence will work.

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS    COMMENTS

                        .
                        .
                        .
          READ                 M      S3         175 NS
          WRTE                 TAB    TAB        595 NS
                        .
                        .
                        .
```

The following READ is conditionally valid:

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS    COMMENTS

                        .
                        .
                        .
          READ                 M      S3         175 NS
                        INC    S3     S3         175 NS
          IMM           LOW    L      0          175 NS
                        ZERO   S4                 175 NS
                               S5     TAB        175 NS
                        .
                        .
                        .
```

Note that both examples will fail frequently in an environment in which there is DCPC activity. Any number of microinstructions may separate a READ and TAB if there is no DCPC activity.

The microroutine sequences shown below are examples of invalid use of READ and WRTE:

```
                    ALU/
        OP/         MOD/          S-BUS/
LABEL   BRCH   SPCL COND   STR    ADDRESS    COMMENTS

                    .
                    .
                    .
        READ                M      S3        READ WILL COMPLETE, BUT
        WRTE                                 THE WRTE IS INHIBITED
*
        READ                M      S3
        WRTE                TAB              177777 WRITTEN INTO MEMORY.
                    .
                    .
                    .
```

When an I/O cycle is in progress, a READ or WRTE must not be initiated before T6 in the cycle under either of the following conditions:

- An input or output routine is in progress. (Refer to paragraph 7-22 for microprogrammed I/O considerations.)
- A skip flag test of the I/O system is taking place.

7-6. SOME MICROPROGRAMMING TECHNIQUES

Techniques for using the alter-skip related micro-orders and for performing microprogrammed arithmetic operations are included in the following paragraphs.

7-7. THE USE OF SRG1 AND SRG2

Micro-order SRG2 is sensitive to the contents of the Instruction Register (IR). In particular, bits 4, 2, 1, and 0 control a variety of shift/rotate actions. However, SRG2 causes the top of the Save Stack to be loaded into the CMAR unless an SRG2 skip condition is met. This pseudo-RTN is usually undesirable in a user microprogram. The simplest way to prevent the undesired loading of the CMAR is to satisfy an SRG2 skip condition by setting bit 3 of the IR and having bit 0 of the T-bus be clear. IR bit 3 = 1 is the equivalent of an Assembler SL*. By ensuring that T-bus bit 0 = 0 as execution of the SRG2 begins, the SRG2 skip test is satisfied and the CMAR is not loaded from the Save Stack. The lines at labels SRG2.1, and SRG2.2, and SRG2.3 in the following microroutine illustrate the above technique.

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS   COMMENTS

                        .
                        .
                        .
SRG2.1    IMM           LOW    CNTR   37B       IR(4-0) = 11111 = SL*, *LF.
SRG2.2                  ZERO                    T-BUS (0) = 0.
SRG2.3          SRG2           S4     S3        S4 = CONTENTS OF S3 ROTATED LEFT 4.
                        .
                        .
                        .
```

As shown in line SRG2.1, the CNTR micro-order may be used in place of IRCM if only IR bits 7 through 0 are significant. Storing into the counter does not alter IR bits 15 through 8. In regard to IRCM, note that if IR bit 10 = 0, the upper five bits of the M-register will be automatically cleared (zeroed) as bits 9 through 0 of the IR are stored into the M-register. If IR bit 10 = 1, bits 14 through 10 of the IR are stored into the M-register (in addition to IR bits 9 through 0) to form an operand address.

Micro-order SRG1 is also sensitive to the contents of the IR, but does not cause loading of the CMAR from the Save Stack; therefore, the use of SRG1 is straightforward as shown in lines SRG1.1 and SRG1.2 below.

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS   COMMENTS

                        .
                        .
                        .
SRG1.1    IMM           HIGH   IRCM   3         IR(9-5) = 11111 = *LF, CLE.
                        .
                        .
                        .
SRG1.2          SRG1           S6     S5        S6 = CONTENTS OF S5 ROTATED LEFT 4,
*                                                  AND E-REGISTER = 0.
                        .
                        .
                        .
```

7-8. USING THE ASG MICRO-ORDER

Micro-order ASG is sensitive to the contents of the IR. In particular, IR bits 7 and 6 may be used to clear, complement, or set the E-register. However, ASG causes the top of the Save Stack to be loaded into the CMAR unless an ASG skip condition is met. This pseudo-RTN is usually undesirable in a user microprogram. The simplest way to prevent the undesired loading of the CMAR is to satisfy an ASG skip condition by setting bit 0 of the IR. For an ASG, IR bit 0 = 1 is the equivalent of an Assembler RSS, i.e., a satisfied ASG skip condition. ASG is also sensitive to IR bit 2, if IR bit 2 = 0 the micro-order in the ALU field is ignored and a PASS is executed. To execute anything but a PASS in the ALU field, set the IR bit 2 to a 1. With the use of the microinstructions shown below, the E-register will be set, S4 incremented and stored into S4, and the microinstruction following the ASG will be executed next:

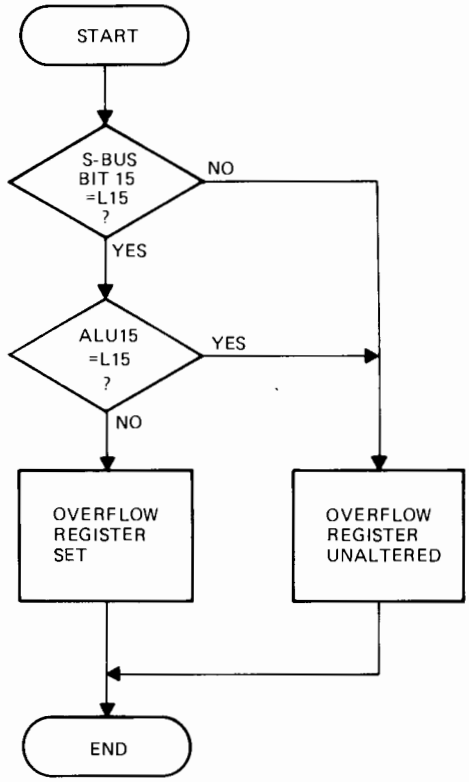
```
                    ALU/
        OP/         MOD/          S-BUS/
LABEL   BRCH  SPCL  COND  STR     ADDRESS     COMMENTS
                    .
                    .
                    .
        IMM         LOW   IRCM    305B        IR(7,6,2,0) = 1,1,1,1, = CCE,RSS.
              ASG   INC   S4      S4          CCE,S4 = S4+1
                    .
                    .
                    .
```

7-9. SETTING AND CLEARING OVERFLOW

Some guidelines for programmatically setting and clearing the Overflow register are shown below. The use of the SOV, COV, ENVE micro-orders are involved.

```
                    ALU/
        OP/         MOD/          S-BUS/
LABEL   BRCH  SPCL  COND  STR     ADDRESS     COMMENTS

                    .
                    .
                    .
* EXPLICITLY SETTING & CLEARING OVERFLOW
              SOV                             EXPLICITLY SETS OVERFLOW
              COV                             EXPLICITLY CLEARS OVERFLOW
*
* SETTING OVERFLOW WITH SHIFT OPERATION
        ARS   L1            B     B           IF B15 NOT = B14 PRIOR TO L1,
*                                                OVERFLOW WILL BE SET AFTER ARS
*                                                EXECUTES
* SETTING OVERFLOW ARITHMETICALLY
        IMM   COV   HIGH    L     200B        L = 040377 = LARGE + NUMBER
        IMM         HIGH    S3    200B        S3 = 040377 = LARGE + NUMBER
        ENVE        ADD     S3    S3          OVERFLOW WILL BE SET
*
        IMM   COV   HIGH    L     0           L15 = 0
        IMM         HIGH    S3    177B        S3 = 077777
        ENVE        INC     S3    S3          OVERFLOW WILL BE SET
*
* THE FOLLOWING WILL NOT SET OVERFLOW CORRECTLY
        IMM   COV   HIGH    L     200B        L = 040377 = LARGE + NUMBER
        IMM         CMHI    S3    200B        S3 = 137000 = LARGE - NUMBER
        ENVE        SUB     S3    S3          OVERFLOW WILL NOT BE SET
                    .
                    .
                    .
```

The rule for setting the Overflow register arithmetically is summarized in figure 7-1.



7115-23

Figure 7-1. Overflow Register Control

7-10. THE USE OF PNM

For time-critical loops, the PNM micro-order can be used as shown in the microroutine below to reduce loop execution times. The microinstruction at label LOOP uses PNM to initialize M for the current READ and to update P for the next READ. Since these functions usually require two micro-instructions, loop execution time reduces by one microinstruction. Saving P and initializing P with the buffer address (assumed to be in B) uses two control memory locations. Microprogram specifications determine whether the control memory/execution time tradeoff is worth while. Note that the restoration of P is "buried" in preparing to exit the microprogram, as in line MPEND:

```
                ALU/
        OP/     MOD/            S-BUS/
LABEL   BRCH    COND    STR     ADDRESS     COMMENTS

                .
                .
                .
                        S3      P           SAVE P
                        P       B           P = BUFFER ADDRESS
                .
                .
                .
LOOP    READ    INC     PNM     P           READ BUFFER, UPDATE BUFFER
LOOPEND                                       ADDRESS.
                .
                .
                .
MPEND   READ    INC     PNM     S3          FIX, P, START FETCH FOR NEXT
*                                             INSTRUCTION.
                .
                .
```

7-11. THE CNTR MICRO-ORDER

If a loop requires 256 or fewer repetitions, and the IR contents are not required, the CNTR micro-order can be used as shown in the microroutine below to reduce loop execution time. Incrementing or decrementing the CNTR is "buried" in line LOOP. Since loop count updating using a scratch register, (or general purpose register) would require a separate microinstruction, loop execution time is reduced by one micro-instruction using this method. Initializing the CNTR with the loop count uses one control memory location. Microprogram specifications determine whether the control memory/execution time tradeoff is worth while. Note that, INCT or DCNT does not use the ALU; therefore, arithmetic operations may be performed in the same microinstruction. Note that ICNT cannot immediately precede a conditional jump which has a CNT4 or CNT8 as the condition.

```
                ALU/
        OP/     MOD/            S-BUS/
LABEL   BRCH    COND    STR     ADDRESS     COMMENTS

                .
                .
                .
                        CNTR    A            CNTR = LOOP COUNT.
                .
                .
                .
LOOP    READ    INC     PNM     P           READ BUFFER, UPDATE BUFFER
*                                             ADDRESS AND LOOP COUNT.
LOOPEND JMP     CNT8    RJS     LOOP        COUNT = 0? NO, CONTINUE.
                .
                .
                .
```

7-12. MAGNITUDE TESTS

If the magnitude of the difference between two operands is less than 32768, the limited test shown in the microroutine that follows may be used to determine whether one of the elements to be compared is arithmetically less than, equal to, or greater than the other element. To understand the limitation of the test, consider integers of −1 (element 1) and +32767 (element 2). Subtracting −1 from +32767 yields +32768, which is a number that cannot be correctly represented by a 16-bit signed integer. The result of the subtraction is ALU bit 15 set, and bits 14 through 0 clear. The AL15 conditional test selects the C1.GT.C2 microinstruction. Clearly, element 2 (+32767) is greater than element 1 (−1), and the test has failed.

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS     COMMENTS

                        .
                        .
                        .
* LIMITED LESS THAN, EQUAL TO, GREATER THAN TEST.
*                                L      S3          L = C1 (FIRST ELEMENT).
SUBTRACT                SUB            S4          ALU = C2 - C1.
          JMP    CNDX   ALZ            EQUAL       ALU = 0? YES, C1 = C2.
          JMP    CNDX   AL15           C1.GT.C2    AL15 = 0? YES, C1 GREATER THAN C2,
C1.LT.C2                                             NO, C1 LESS THAN C2.
                        .
EQUAL                   .
                        .
C1.GT.C2                .
                        .
                        .
```

The test in the microroutine that follows holds for all 16-bit signed integers. Consider how integers of −1 and +32767 are now analyzed. Based on the XOR of the two elements, the ALZ test for equality fails, the AL15 RJS test for equal signs fails, and the L15 test for element 1 less than element 2 succeeds which causes the C1.LT.C2 microinstruction to be selected correctly.

Note that when the signs of the elements being compared are opposite, subtraction is unnecessary since the negatively signed element must be smaller. Note also that when the signs of the element signs are the same, subtraction always yields a result which causes correct microinstruction selection.

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS     COMMENTS

                        .
                        .
                        .
* GENERAL LESS THAN, EQUAL TO, GREATER THAN TEST.
                                L      S3          L = C1 (FIRST ELEMENT).
                        XOR            S4          ALU = C2 XOR C1.
          JMP    CNDX   ALZ            EQUAL       ALU = 0? YES, C1 = C2.
          JMP    CNDX   AL15   RJS     SUBTRACT    SIGNS = ? YES, SUBTRACT.
          JMP    CNDX   L15            C1.LT.C2    L15 = 1? YES, C1 LT C2.
          JMP                          C1.GT.C2    NO, C1 GT C2.
SUBTRACT                SUB            S4          ALU = C2 - C1.
          JMP    CNDX   AL15           C1.GT.C2    AL15 = 1? YES, C1 GT C2.
C1.LT.C2                                             NO,C1 LT C2.
                        .
EQUAL                   .
                        .
C1.GT.C2                .
                        .
                        .
```

7-13. MEMORY PROTECT CONSIDERATIONS

If the HP 12892B Memory Protect (MP) accessory is used with the Computer, there is a relationship between certain micro-orders and Memory Protect that should be understood.

The Main Memory section and I/O section are involved in the Memory Protect functions. You will also want to refer to the read/write and microprogrammed I/O considerations in this section (in addition to the discussion of MP related micro-orders presented in the following paragraphs) for a complete understanding of the microprogramming/Memory Protect relationship.

Memory Protect can only be enabled or disabled through use of the I/O system; there are no micro-orders that directly perform these operations. When an STC 05 instruction enables MP, main memory access cannot occur below the value set in a Fence register and no I/O operations (except those referencing select code 01) can occur. The Memory Protect functions are disabled by any interrupt, interrupting to a non-I/O type instruction in a trap cell. Refer to the discussion of the Memory Protect accessory in your *Computer Series Operating and Reference Manual* and have an understanding of MP details before microprogramming with this accessory installed. The key points to remember when studying the following descriptions of MP related micro-orders (also refer to table 4-1) are that MP effectively does not allow any I/O and that at the microprogramming level you are not necessarily under the "protective umbrella" of MP when performing main memory operations. These factors impose upon you the responsibility of being acutely aware of the effect of your microprogram.

Memory Protect must be turned off to generate some MEM signals and execute I/O instructions. The following example demonstrates how to turn off Memory Protect. To turn off Memory Protect, execute an I/O instruction to any select code other than 1. This will violate Memory Protect, disabling it and cause assertion of FLG5, on the Memory-Protect PCA which is the interrupt signal to the CPU for select code 5. An IAK following the IOG will eliminate the interrupt request from select code 5. However, the Memory Protect hardware will not allow execution of any I/O instructions until a FTCH micro-order has been executed. FTCH performs special operations on the CM addressing logic, therefore a RTN micro-order can not be used successfully. This implies that the routine that turns off Memory Protect is in the zero level of subroutines, and the microinstruction JMP 0B must be used to return to CM location 0. However if subsequent subroutine calls are required before returning to FETCH then the CM addressing logic must be initialized, refer to example 2. This function is performed by the JTAB micro-order in conjunction with the INCI and remaining micro-orders to prevent the JTAB branch from occurring.

```
Example 1
LABEL OPER SPEC ALU STORE S-BUS        COMMENTS
              .
              .
              .                    VIOLATE MEMORY PROTECT
            IOG                    SELECT CODE ≠ 1
            IAK                    CLEAR MEMORY PROTECT INTERRUPT
            FTCH                   ALLOW I/O INSTRUCTIONS
              .
              .
              .
      JMP                  0B      RETURN TO FETCH
```

```
Example 2
LABEL OPER SPEC ALU STORE S-BUS        COMMENTS
            .
            .
            .                        VIOLATE MEMORY PROTECT
          IOG                        SELECT CODE ≠ 1
          IAK                        CLEAR MEMORY PROTECT INTERRUPT
          FTCH                       ALLOW I/O INSTRUCTIONS
          INCI ZERO                  PREVENT JTAB BRANCH
          JTAB DBLS                  INITIALIZE CM LOGIC
            .
            .
            .
      JMP                  0B      RETURN TO FETCH
```

7-14. THE FTCH MICRO-ORDER

The FTCH micro-order stores the present contents of the M-register into the MP Violation register, clears the MP Violation Flag flip-flop, and resets the MP Indirect Counter (indirect address levels). The FTCH micro-order also performs operations on CM addressing logic and is therefore to be used only in the base set. Refer to table 4-1.

7-15. IRCM

The IRCM micro-order causes MP hardware to record the type of instruction being stored in the IR and whether or not IR bits 5 through 0 equal 01. When MP is enabled (by an STC 05 instruction):

- Only I/O instructions with a select code of 01 may be executed.
- The IR must be loaded prior to initiating an I/O cycle with the IOG to ensure that the signal decoding logic is enabled.

When MP is not enabled:

- No restriction is placed on select codes that are otherwise valid.

7-16. INCI

The INCI micro-order should be used whenever another level of indirect addressing is to be implemented by a microprogram. After three counts of the MP Indirect Counter, the MP hardware *effectively* performs an ION micro-order (i.e., a pseudo ION), thus enabling recognition of I/O interrupts by branch conditional type microinstructions. INCI has special considerations involved if used just before a microinstruction containing the JTAB micro-order. Refer to table 4-1 and appendix C for INCI and JTAB use. Also see interrupt handling techniques in this section.

7-17. MPCK

The MPCK micro-order should be used (particularly in main memory write operations) to ensure that a microprogram will not alter memory below the protective address "fence" set in MP. When this micro-order is used and a MP violation is detected:

- All subsequent READ microinstructions end with invalid data in the T-register.
- No WRTE micro-order will be executed.
- All I/O signals from the computer are inhibited until after the next FTCH or IAK micro-order is executed.

Refer to the read and write considerations outlined in paragraph 7-4 for using MPCK and to table 4-1 for restrictions when using MPCK.

7-18. THE IOG MICRO-ORDER

If Memory Protect is enabled, the use of the IOG micro-order causes a check of the select code and the MP Violation Flag flip-flop is set if the select code (IR bits 5 through 0) is not equal to 01. If an MP violation is detected, the actions described for the MPCK, micro-order (above) take place.

7-19. IAK

When an IAK micro-order is executed, the MP Indirect Counter is cleared. The IAK micro-order also causes the computer to "freeze" (i.e., stop executing microinstructions) until I/O period T6 occurs and then issue an IAK signal, acknowledging receipt of an interrupt request, to the requesting device. If the interrupt device select code is 05, the PARITY indicator on the Operator Panel is cleared and the MP Violation Flag flip-flop is cleared. Whenever IAK executes, logic in the MP hardware determines whether or not the MP should be disabled (clear the control bit). This hardware determination is made six microinstructions after the IAK. MP is disabled if no I/O instruction (IOG) micro-instruction is executed or if a halt is executed. To re-enable Memory Protect, an STC 05 instruction is required. The execution of IAK causes the MEM hardware to address the system map which will alter the memory address.

7-20. THE IOFF MICRO-ORDER

The IOFF micro-order turns off recognition of I/O interrupts but does not disable Memory Protect. The Memory Parity function shares the same interrupt location as MP and the *Operating and Reference Manual* provides information for determining the source of an interrupt. The DMS accessory also works in conjunction with MP for certain functions which are also described in the *Operating and Reference Manual*.

7-21. DUAL CHANNEL PORT CONTROLLER CONSIDERATIONS

The HP 12897B Dual Channel Port Controller (DCPC) "steals" full I/O cycles to perform direct transfers between peripheral devices and main memory. The DCPC functions are essentially transparent to microprogramming. When DCPC takes a sequence of consecutive I/O cycles for input transfers, any attempted IOG, READ, or WRTE micro-orders will freeze the Control Processor until DCPC is finished. When using DCPC with MBIO and MPP refer to Section 13 for special considerations.

Both DCPC channels may operate concurrently but Channel 1 has priority over Channel 2 when simultaneous cycles are requested. A channel stealing consecutive I/O cycle may operate at up to 890,000 words per second during output data transfers,* and 1,000,000 words per second during input data transfers. Under maximum bandwidth conditions the Control Processor is essentially locked out. For further information on DCPC refer to the applicable manuals.

7-22. MICROPROGRAMMED I/O

Microprogramming input and output (I/O) functions requires more care than any other type of microprogramming because there are strict timing dependencies. To maintain the integrity of the I/O system, each I/O device control signal is generated in a specific time period (T-period). Section 5 in this manual defines and describes the timing for the computer. Summary information on timing is presented in subsequent paragraphs but you should be familiar with the concepts presented in section 5 before attempting microprogrammed I/O.

Also provided in subsequent paragraphs are applicable information on signal generation by the I/O section; I/O control, and data transfer guidelines for microprogramming; and interrupt handling rules. In addition to the information in paragraph 7-13, Memory Protect in relation to I/O is discussed briefly. Guidelines for forming and executing microprogrammed I/O instructions are included and some special I/O techniques are covered. These special techniques are referenced from section 13.

7-23. SYNCHRONIZING WITH THE I/O SECTION

The I/O cycle consists of five T-periods designated T2 through T6. Specific I/O activity is restricted to certain T-periods in order to synchronize data flag setting, data latching, and resolving multiple interrupt requests. (Section 14 provides an example of I/O microprogramming that you can reference while studying the following information.) Microinstructions in T-periods generally execute in 280 nanoseconds for each T-period (see section 5 on timing variations).

A microprogram becomes synchronized with the I/O system when the Control Processor detects an IOG micro-order. When this occurs, the Control Processor "freezes" (i.e., stops executing microinstructions) until period T2. Any other micro-orders in the microinstruction containing IOG are executed without delay but the IOG is not executed until T2. The next microinstruction is executed during period T3, the next during T4, and so on. IOG may be used in any microinstruction that does not require some other Special or Modifier micro-order.

*Refer to your *Computer Series Operating and Reference Manual* specifications for DCPC latency.

As can be realized, the relationship between microinstruction execution and the I/O T-periods places certain restrictions on the use of some registers and micro-orders. In order for your microprograms to execute properly, you must observe the following rules:

- Do not start an I/O cycle (using IOG) before data is transferred from the T-register following a READ operation. The reason is that if the IOG causes a freeze, the data in the T-register will be invalid. For example, a microinstruction sequence similar to the following must *not* be programmed:

```
                ALU/
        OP/     MOD/        S-BUS/
LABEL   BRCH    COND   STR  ADDRESS   COMMENTS

                .
                .
                .
        READ    INC    PNM  P
        IOG     PASS   S4   TAB
                .
                .
                .
```

- Load the Instruction Register before issuing an IOG unless there is no chance that Memory Protect is enabled. (See paragraph 7-31 on special techniques.)

The following conditions will always cause the Control Processor to freeze in order to synchronize with the I/O section:

- An IOG is in the Special field and either the cycle period is not T2 or the DCPC is operating.

- An IAK micro-order is in the Special field and either the I/O cycle period is not T6 or the DCPC is operating.

It should be noted that the Computer main memory read and write operations may cause microinstruction execution delays that are defined as "pauses". This is not the same as "freezing" to synchronize with the I/O section. Refer to section 5 for details.

7-24. I/O SECTION SIGNAL GENERATION

When the IOG micro-order is executed, the I/O system sends I/O backplane signals to the I/O devices starting at period T3 according to the contents of the Instruction Register (IR). These signals are different and separate from micro-orders. For example, on a data output transfer, the IOG micro-order causes the I/O section to generate the IOO signal during T3 and T4 (caused by IR bits 8,7, and 6 = 1,0,0). But the micro-order IOO (which only serves to connect the S-bus and I/O bus) must be microprogrammed to be present during T4 and T5. If the proper microprogramming sequence is not followed there will be (in this case) a race condition between the backplane IOO signal and the effect of the IOO micro-order.

Table 7-1. Backplane I/O Signal Generation Determined by IR Bits 11 through 6

IR* 11	10	9	8	7	6	BACKPLANE I/O SIGNAL	BACKPLANE I/O SIGNAL TIME	GENERAL USE
x	x	y	0	0	0	none	T3	Clear the Run flip-glop on the CPU (HLT).
x	x	0	0	0	1	STF	T3	Set device flag (STF).
x	x	1	x	x	1	CLF	T4	Clear device flag (CLF).
x	x	y	0	1	0	SFC	T3-T5	SKPF condition is true if and only if the device flag is clear (SFC).
x	x	y	0	1	1	SFS	T3-T5	SKPF condition is true if and only if the device flag is set (SFS).
x	x	y	1	0	x	IOI	T4	If the corresponding select code is not between 1 and 7 (during T4 only), transfer the input data latch on the device onto the I/O bus (MIA/B, LIA/B).
							T5	Transfer the input data latch on the device onto the I/O-bus.
x	x	y	1	1	x	IOO	T3-T4	Store the I/O bus into the input data latch on the device (OTA/B).
0	x	y	1	1	1	STC	T4	Set device control flag (STC).
1	x	y	1	1	1	CLC	T4	Clear device control flag (CLC).

NOTE:
*Bit entries with x are not significant for the I/O signal specified. If bit 9 is set the device flag is cleared; if bit 9 is clear the device flag is not altered. Bit 9 entries with y indicate the option available to hold or clear the device flag in these instructions. Bits 5 through 0 (not shown) indicate the select code for the device. (Assembler instructions STO, CLO, SOC, and SOS all referring to the Overflow register always have bits 5 through 0 = 01 (octal).

In order for your microprogram to perform an I/O operation, IR bits 5 through 0 must contain the select code (SC) of the device that is to respond to the I/O signals. As shown in table 7-1, IR bits 11 through 6 determine which I/O signals are sent. The IR must be loaded prior to or during occurrence of the IOG to ensure that the correct signals are sent to the desired SC (refer to paragraph 7-23). If Memory Protect is enabled, the IR must be loaded prior to issuing IOG (refer to paragraphs 7-13 and 7-28). With certain exceptions, I/O can not be done with MP enabled (refer to paragraph 7-31).

Select codes 00,01,02,03,04, and 05 are usually used by the interrupt system, the Operator Panel, Dual Channel Port Controller (DCPC), power fail, and Main Protect/parity interfaces and accessories. For a description of the effect of I/O signals on these select codes, refer to your *Computer Series Operating and Reference Manual.*

7-25. I/O CONTROL

A microprogram can generate I/O control signals for the select code of an I/O device without I/O data transfer. As previously described, IR bits 5 through 0 must contain the SC of the device and bits 11 through 6 may specify any of the following control signals:

STF CLF SFC SFS STC CLC HLT

Note that CLF can be generated in conjunction with any other signal simply by setting IR bit 9 to 1 as shown in table 7-1. For example, the Assembly language instruction combination STC,C can be simulated by setting IR bits 11 through 6 to 0x1111 (where x means "don't care"). (Refer to table 7-1.) An I/O control routine with the IR specifying STC and select code 05 can be used to re-enable Memory Protect.

For SFS and SFC, the state of the device flag may be tested by a conditional branch microinstruction (word type III) having micro-order SKPF in the Condition field. Micro-order SKPF is true only when the SFS I/O signal is present and the flag is set, or when SFC is present and the flag is clear. The SKPF test should be microprogrammed to occur during I/O period T4 or T5 (i.e., two or three microinstructions after the IOG). Any operation desired may be performed as a result of this test; for example, incrementing the contents of the P-register causes a skip in the main memory program. Refer to paragraph 7-30 for examples of forming and executing I/O control microinstructions.

7-26. I/O OUTPUT

An I/O output routine must use both the IOG and IOO micro-orders. (Special exceptions are discussed in section 13). The IR must contain the bits that specify the IOO signal and the SC of the IOO device. The same bit pattern for STC.C also specifies the IOO signal. The IOO micro-order connects the S-bus to the I/O bus. Do not confuse this with the IOO backplane I/O signal (refer to paragraph 7-24). The microprogram must put the proper data on the S-bus, then direct it onto the I/O bus. The IOO backplane signal latches the I/O bus data into the I/O device interface card. Detailed timing requirements are:

- During I/O period T3, the S-bus must be driven by the register containing the output data to prepare for the transfer to the I/O bus.
- During T4 and T5, the S-bus must be driven by the same register and the IOO micro-order must be in the Store field. This ensures valid data on the I/O bus.

For example, an OTA/B instruction can be simulated by the following sequence of microinstructions:

```
                ALU/
        OP/             MOD/                S-BUS/
LABEL   BRCH    SPCL    COND    STR         ADDRESS     COMMENTS

                        .
                        .
                        .
GO              IOG                                     T2
                                            CAB         T3
                                    IOO     CAB         T4
                RTN                 IOO     CAB         T5
                        .
                        .
                        .
```

7-27. I/O INPUT

An I/O input routine must use both the IOG and IOI micro-orders, and the IR must contain the bits that specify the IOI signal and the SC of the I/O device. Special exceptions are discussed in section 13.) The IOI signal transfers data from the I/O device interface card to the I/O bus and the IOI micro-order connects the I/O bus to the S-bus to allow data to be present for latching into a register. The IOI micro-order is used in the I/O cycle during T5 to input data from the I/O bus onto the S-bus. Do not confuse this with the IOI backplane I/O signal present during T4 and T5. (Refer to paragraph 7-24.) For example, an LIA/B instruction can be simulated by the following microinstruction sequence:

```
                ALU/
        OP/     MOD/            S-BUS/
LABEL   BRCH    COND    STR     ADDRESS     COMMENTS
SPCL

                .
                .
                .
INPUT   IOG                                 T2
        NOP                                 T3
        NOP                                 T4
        RTN             CAB     IOI         T5
                .
                .
                .
```

You can see from the above that parts of some I/O microroutines may have unused microinstruction periods. Caution is required when using these periods. Until all I/O-related microinstructions have been executed for an I/O cycle, do not use microinstructions that may cause the CPU to freeze. (Refer to paragraph 7-23.) In the above I/O input example, if the T3 and T4 NOP's were replaced by READ and TAB micro-orders (in T3 and T4 respectively), the CPU would pause in the middle of T4 and IOI would not be executed until too late to correctly handle the data transfer. On the other hand, during an I/O control routine that is not generating SFS or SFC signals, many kinds of microinstructions can be used after the IOG.

7-28. MEMORY PROTECTION IN RELATION TO I/O

When an instruction is loaded into the Instruction Register, Memory Protect (MP) records information about the instruction. When an IOG micro-order is detected, MP checks the select code (IR bits 5 through 0). If the SC is not equal to 01, MP inhibits any I/O signals and prevents the Control Processor from altering main memory or the P- or S- registers, and generates an interrupt request. (A microprogram cannot prevent this if MP is enabled.) Thus, MP protects a portion of memory and maintains compatibility with HP software operating systems for I/O operations even in the microprogramming environment. Refer to your *Computer Series Operating and Reference Manual* and to paragraph 7-13 of this manual for further details on Memory Protect.

7-29. INTERRUPT HANDLING

Once a microprogram starts executing, it has complete control over the computer until it terminates. It can not be interrupted, suspended, or terminated unless the microprogram itself checks for interrupts. It is not desirable to hold off interrupts for very long and you must decide how long your microprograms can be allowed to execute before testing for an interrupt. In making this decision, consider the impact that a long non-interruptible microprogram can have in the RTE environment.

When a microprogram detects an interrupt, it should execute a JSB to a microroutine that saves whatever is necessary to allow the microprogram to continue after the interrupt is serviced or to provide for complete restart of the microprogram. (Refer to microprogram examples in section 14 for an illustration.) The P-register must be set to point to an address one location beyond the main memory instruction that invokes the microprogram (the instruction that was interrupted). The M-register will be adjusted to point to the address of the main memory instruction that will handle the interrupt. It will be readjusted later so no special conditions are placed on M. For example, suppose your main memory instruction invoking a microprogram resides in the location designated I. Then, if your microprogram tests for and detects an interrupt you must:

- Ensure P = I + 1.
- Execute a RTN (or JMP to control memory location 6 if in a microsubroutine). This is described in more detail below.

If parameters are saved, the microprogram must be written to begin with a test that determines the starting point of the microprogram based on whether or not the microprogram was interrupted.

Generally, to initiate interrupt service, your microprograms must branch (JMP) or return (RTN) to control memory location 6 where the base set microprogram takes the trap cell address from the Central Interrupt Register and gives control to a main memory routine which services the interrupt. When the main memory interrupt routine which services the interrupt terminates, the interrupted microprogram is restarted (assuming the P-register was properly set upon interrupt detection). A check must be made to see if the interrupt system is turned on.

The presence of a pending interrupt or halt request can be detected by a microprogram in two ways:

- Executing a conditional test microinstruction (JMP CNDX) having HOI or NINT in the Condition field.
- Executing a JMP or RTN to CM location 0; a pending interrupt or halt will cause control memory addrss 6 to be loaded into the CMAR to handle the interrupt.

Using a RTN to pass control to control memory location 6, as shown in the microroutine below, line EXIT1, will not work if the microroutine being exited was entered with a JSB. Using a JMP to location 6, as in line JUMP (in the microroutine below) will always work. NINT may also be used to check for interrupts. Note that NINT is not sensitive to halts.

```
                ALU/
        OP/             MOD/            S-BUS/
LABEL   BRCH    SPCL    COND    STR     ADDRESS     COMMENTS

                        .
                        .
                        .
        JMP     CNDX    HOI             EXIT1       INTERRUPT? YES, EXIT
                        .
                        .
                        .
EXIT1           RTN     DEC     P       P           FIX P, RTN (??).
                        .
                        .
                        .
        JMP     CNDX    HOI             EXIT2       INTERRUPT? YES, EXIT.
                        .
                        .
                        .
EXIT2                   DEC     P       P           FIX P, EXIT TO HALT-OR-
JUMP    JMP                             6           INTERRUPT MICROROUTINE.
                        .
                        .
                        .
```

When the Halt-Or-Interrupt microroutine is reached, the P-register is decremented and a test is made to see if the Operator Panel was used to cause a halt. If not, an IAK micro-order freezes the Control Processor until I/O period T6, then causes the I/O system to send an IAK signal to the interrupting device. A CIR micro-order causes the interrupting device's SC (trap cell address) to be placed on the S-bus, then this is stored into the lower-order 6 bits of the M-register (high order bits = 0). A read from the address in the M-register obtains the first instruction of the main memory interrupt handling program.

Suppose a microprogram is to be interruptible, but only by emergency interrupts (i.e., halt, parity error, DMS, Memory Protect). An HOI conditional test detects emergency interrupts, but also detects I/O interrupts. However, issuing an IOFF prior to the HOI test prevents detection of I/O interrupts. Issuing an ION after the HOI test reenables detection of I/O interrupts. The microroutine below illustrates this process. Note that IOFF and ION control only the detectability of power fail and I/O interrupts, and do not turn off or turn on the interrupt system. Note also that I/O interrupts held off by an IOFF condition remain pending (i.e., are not lost), and are detectable when the ION condition is re-established:

```
                ALU/
        OP/             MOD/            S-BUS/
LABEL   BRCH    SPCL    COND    STR     ADDRESS     COMMENTS

                        .
                        .
                        .
                IOFF                                PREVENT DETECTION OF I/O
*                                                     INTERRUPTS
        JMP     CNDX    HOI             INTRPT      TEST FOR DETECTABLE INTERRUPTS,
*                                                     I.E., HALT, PARITY ERROR,
*                                                     DMS, MEMORY PROTECT.
*
                ION                                 REENABLE DETECTION OF I/O
*                                                     INTERRUPTS.
                        .
                        .
                        .
```

7-30. FORMING AND EXECUTING MICROPROGRAMMED I/O INSTRUCTIONS

The following continuous example microroutines show how to accomplish formation and execution of some microprogrammed I/O instructions. These examples are offered as models for you to write microprograms that perform I/O functions. Note that putting the select code (SC) in the L-register is prerequisite to using the IOR in the STC line. MPP and block I/O transfers require somewhat different I/O instruction formats. MPP and block I/O transfers are discussed in section 13.

```
                        ALU/
          OP/           MOD/          S-BUS/
LABEL     BRCH   SPCL   COND   STR    ADDRESS    COMMENTS

                        .
                        .
                        .
*
* READ CIR (CENTRAL INTERRUPT REGISTER)
CIR                             L      CIR        L= SC (SELECT CODE).
*
* FORM AND EXECUTE STC SC, C.
STC       IMM    L4     CMLO   S8     303B       S8 = 001700 = STC 0,C.
                        IOR    S8     S8         FORM STC SC,C.
                               IRCM   S8
                 IOG                              T2 EXECUTE STC, SC,C.
*
* FORM AND EXECUTE LI* SC.
LI*       IMM           CMHI   S4     376B       S4 = 000400 = LI* 0.
                        IOR    S4     S4         FORM LI* SC.
                               IRCM   S4
                 IOG                              T2 EXECUTES LI* SC.
                 NOP                              T3 SEE NOTE 1.
                 NOP                              T4 SEE NOTE 1.
                               S5     IOI        T5 S5 = DATA.
*
* FORM AND EXECUTE OT* SC,
OT*       IMM    L1     CMLO   S9     77B        S9 = 000600 = OT* 0.
                        IOR    S9     S9         FORM OT* SC.
                               IRCM   S9
                 IOG                              T2 EXECUTE OT* SC.
                                      S5         T3 SEE NOTE 4.
                               IOO    S5         T4 DATA CLOCKED OUT AT,
                               IOO    S5         T5 T4/T5 INTERFACE.
*
* FORM AND EXECUTE SFS SC.
SFS       IMM           CMLO   S10    77B        S10 = 000300 = SFS 0,
                        IOR    S10    S10        FORM SFS SC.
                               IRCM   S10
WAIT             IOG                              T2 EXECUTE SFS SC.
                 NOP                              T3 SEE NOTES 1, AND 2.
          JMP    CNDX   SKPF   RJS    WAIT       T4 SEE NOTE 3.
*
* LOAD CIR, ACKNOWLEDGE INTERRUPT
IAK              IAK                              T6
*
*NOTES:
* 1. ANY NON-FREEZABLE MICROINSTRUCTIONS MAY BE USED IN PLACE OF THE NOP.
* 2. THE FLAG CAN BE SENSED NO EARLIER THAN T4.
* 3. EACH ATTEMPT TO SENSE THE FLAG REQUIRES AN IOG: THEREFORE, THE JMP TARGET FOR
*    UNSUCCESSFUL SENSING OF THE FLAG MUST BE WAIT NOT ``*''.
* 4. SEE PARAGRAPH 7-24, SIGNAL GENERATION (I.E., THE IOO SIGNAL AND IOO MICRO-ORDER ARE NOT
*    ONE IN THE SAME).
                        .
                        .
                        .
```

7-31. SPECIAL I/O TECHNIQUES

The following microroutine shows how to perform microprogrammed I/O with both the interrupt system and Memory Protect enabled. This is desirable when writing I/O data into main memory in a DMS environment, and/or Memory Protect checks are required. The microroutine shown assumes that S3 and S5 have previously been initialized with the device select code and current buffer address, respectively. An input function, LI*, will be performed: "*" indicates that the microroutine selects the input data register.

Lines FAKESC and REALSC work together to enable execution of an I/O instruction with Memory Protect enabled. Micro-order IOG, in addition to initiating an I/O operation, checks the I/O operation select code (i.e., IR bits 5 through 0). If the select code is 01, the I/O operation proceeds. Attempting to use any other select code inhibits the I/O operation and generates a Memory Protect interrupt. However, the Memory Protect Hardware checks the select code when the store into the IR occurs in line FAKESC. The store into the CNTR does not cause a check of the IR by the Memory Protect Hardware; therefore, the I/O operation proceeds without a Memory Protect interrupt generated.

If the write to main memory generates a DMS or Memory Protect interrupt, the HOI conditional test detects the interrupt and terminates the microprogram. The IOFF micro-order prevents detection of I/O interrupts permitting "privileged" I/O as required for the MPP or block I/O transfer. Section 13 contains examples of MPP and block I/O microprograms.

```
                ALU/
        OP/             MOD/            S-BUS/
LABEL   BRCH    SPCL    COND    STR     ADDRESS     COMMENTS

                        .
                        .
                        .
        IMM             CMHI    L       376B        L=LI* 0
        IMM             CMLO    S4      376B        S4=1
                IOFF    IOR     S4      S4          S4=LI* 1
FAKESC                          IRCM    S4          IR=LI* 1
                        IOR     S4      S3          S4=LI* SC
REALSC                          CNTR    S4          IR=LI* SC
                IOG             M       S5          START I/O OPERATION
                                                    M=BUFFER ADDRESS
                                S6      IOI         S6=DATA
        WRTE    MPCK            TAB     S6          WRTE DATA, DO MPCK
*
        JMP     CNDX    HOI             INTRPT      TEST FOR HALT,
*                                                   PARITY ERROR, DMS, OR
*                                                   MEMORY PROTECT INTERRUPTS
                        .
                        .
                        .
```

7-32. I/O MICRO-ORDER SUMMARY

All micro-orders that are generally used in I/O microprogramming are summarized in table 7-2 for your reference.

Table 7-2. I/O Micro-Order Summary

MICRO-ORDER	WORD TYPE	FIELD	CONDENSED MEANING
IAK	I, II	Spec.	At T6, load the CIR and issue the IAK signal.
IOFF*	I, II	Spec.	Disable normal interrupt recognition.
IOG**	I, II	Spec.	Freeze action until T2 then do what is in the IR.
ION**	I, II	Spec.	Re-enable normal interrupt recognition.
IOO	I, II	Store	Connect the S-bus to the I/O bus (for output); used after an IOG micro-order.
CIR	I	S-bus	Put the CIR content on the S-bus.
IOI	I	S-bus	Connect the I/O bus to the S-bus.
HOI	III	Cond.	If there is a halt or an interrupt pending, branch to the CM address in this microinstruction address field.
NINT	III	Cond.	If there is no interrupt pending, branch to the CM address in this microinstruction address field.
SKPF	III	Cond.	Check to see if I/O signal SFS is present (T3 to T5) *and* the addressed I/O device's flag is set. If the above conditions are true, branch to the CM address shown in this microinstructions address field. — OR — Check to see if SFC signal is present (T3 to T5) and the I/O device's flag is clear.

NOTES:
*This micro-order can also be used in the Special field of a word type IV (unconditional branch microinstruction).
**This can be used in the Special field of word type IV microinstructions. The branch microaddress is modified by bits in the IR. See table 4-1 explanations.

7-33. DYNAMIC MAPPING SYSTEM CONSIDERATIONS

If you have the HP 13305A Dynamic Mapping System (DMS) installed there are a number of Assembly language instructions that may be used to program the accessory. These Assembly language instructions invoke HP written microroutines in the HP reserved area of CM to operate DMS according to HP's design specifications. The micro-orders used in HP's microinstructions and microroutines for controlling DMS are also available for your microprogramming use.

It is beyond the scope of this manual to discuss HP's method of operating DMS or describing operation of the DMS hardware. However, a discussion of the three micro-orders (referenced from table 4-1) you may use and the DMS signals generated is within the scope of user microprogramming. (For more information on HP 13305A DMS operation and the applicable HP Assembler language instructions refer to your *Computer Series Operating and Reference Manual*). A prerequisite to using the DMS micro-orders described below is that you be thoroughly familiar with the DMS and its operation.

With DMS installed, the Memory Expansion Module (MEM), residing (logically) in front of the main memory controller, forms a 20-bit address from the 15-bit main memory address received on the M-bus. DMS always "looks at" the M-bus address and MEM creates the 20-bit address for DMS according to control signals received from the Control Processor. The control signals, of course, are generated because of the Control Processor's decoding of microinstructions from CM. The three micro-orders; MESP (in the Special field), MEU (in the Store field), and MEU (in the S-bus field) that can be used in microinstructions involving DMS, must be used in tandem. That is, a signal sent to the DMS is generated from the "decoding" of a specific combination of the three micro-orders.

There are three signals generated directly from control memory that are used to control the MEM. In the Special field, "MESP" generates MESP. In the Store field, "MEU" generates the MEST signal. In the S-bus field "MEU" generates MEEN.Other signals which directly affect the MEM are MPCK, READ, TEN, IAK (CIREN). Table 7-3 indicates what 'control line' signal is generated by each combination of the micro-orders. The three micro-orders are used in a one-of-eight command structure. Because a combination of all three micro-orders must be used (Special field, Store field, S-bus field) only word type I microinstructions are used for DMS. Table 7-4 lists all the functions performed by each of the control signals referenced by table 7-3. The DMS functions are performed only in the microcycle during which they are asserted (with the exception of Q_4, port 1).

Table 7-3. MEM Signals Invoked by Micro-Orders

LABEL	OP	SPEC	ALU	STORE	S-BUS	MEM SIGNAL	RULES (SEE NOTES)
@	@	MESP	@	MEU	MEU	Q_0	1, 4
@	@	MESP	@	MEU	$	Q_1	1, 4
@	@	MESP	@	$	MEU	Q_2	1, 4
@	@	MESP	@	$	$	Q_3	2, 4
@	@	*	@	MEU	MEU	Q_4	4
@	@	*	@	MEU	$	Q_5	3, 4
@	@	*	@	$	MEU	Q_6	—
@	@	*	@	$	$	Q_7	—

@ = Any legal code
* = Any legal code except MESP
$ = Any legal code except MEU

RULES GOVERNING MEM SIGNALS:

1. Must have a READ or RJ30 or WRTE *exactly two* microinstructions before use of the micro-order, and a READ, RJ30 or WRITE instruction may not be repeated until execution is complete.
2. Must have a READ, RJ30 or WRTE either 1 or 2 microinstructions before use of the micro-order.
3. Must be a READ or RJ30 or WRTE either 1, 2 or 3 microinstructions before use of the micro-order.
4. Must not occur in the same microinstruction as READ or RJ30 or WRTE.

Q5 control information:

- **When issuing a Q_5 command, further information is needed to indicate the utility register into which you wish to store information. Since the information has been presented on the S-bus and none of the registers require more than 11 bits of information themselves, several of the S-bus bits are reserved for determination of which register is activated.**

- **Bit 14 indicates that the MEM State Registers are to be loaded (i.e., enable/disable MEM; select system/user map). Bits 9 and 8 contain the status information.**

- **Bit 13 indicates that the Address Register is to be loaded. Bits 7 through 0 contain the address information.**

- **If a Q_4 signal has preceded this step by exactly one microcycle (i.e., Q_4, Q_5 in a row), then bit 14 will indicate that the Fence Register will be loaded. Bits 10 through 0 contain the fence information.**

NOTE

Any modification of the fence register will also effect base page addressing for DCPC, as DCPC uses logical to physical address translation rules in the base page similar to those of the user map.

- **Bit 15 is used to override the Protected Mode, thus allowing these registers (specifically the State Registers) to be altered under microprogram control at any time.**

Table 7-4. DMS Micro-Order Control Signals

SIGNAL	FUNCTION
Q_0	1. Enable SYS/USR map to S-bus per MEAR bit 5:0 = SYS, 1 = USR. 2. Store S-bus into PORTA/PORTB map per MEAR bit 7:0 = PORTA, 1 = PORTB. 3. Relative map address specified by MEAR bits 4 through 0. 4. Increment MEAR.
Q_1	1. Store S-bus into maps per MEAR bits 6 and 5:00 = SYS, 01 = USR, 10 = PORTA, 11 = PORTB. 2. Relative map address specified by MEAR bits 4 through 0. 3. Increment MEAR.
Q_2	1. Enable maps to S-bus per MEAR bits 6 and 5:00 = SYS, 01 = USR, 10 = PORTA, 11 = PORTB. 2. S-bus bits 13 through 10 are always low. 3. Relative map address specified by MEAR bits 4 through 0. 4. Increment MEAR.
Q_3	1. Select opposite program map (does not change currently selected map per Q_5). 2. Can generate DMAFRZ to CPU.
Q_4	1. Set "Status Command" flag through next Control Processor cycle (defines Q_6 operation). 2. Reset to currently selected program map (nullifies Q_3). 3. Set "Enable Base Page Fence" Flag through next Control Processor cycle (partly defines Q_5 operation).
Q_5	1. Store S-bus into MEM (other than maps) a. MEM State Register (2 bits) = S-bus bits 9,8: If S-bus bit 9 = 0, disable MEM; = 1, enable MEM. If S-bus bit 8 = 0, select SYS maps; = 1, select USR maps. b. MEM Base Page Fence Register (11 bits) = S-bus bits 10 through 0. c. MEM address Register (7 bits) = S-bus bits 6 through 0. 2. Register selected by S-bus bits 15 through 13: If S-bus bits 15 through 13 = 000 = Base Page Fence Register if preceded by Q_4; 001 = Address Register; 010 = State Register. 3. If S-bus bit 15 = 1 then Memory Protect is disabled for the current microinstruction.
Q_6	1. Enable MEM data (other than maps) onto S-bus. a. Normally enables MEM Violation Register. b. If preceded by Q_4 signal microinstruction, Status Register enabled.
Q_7	1. No MEM (DMS) microinstruction specified (NOP state for MEM).

Notes:
1. MEAR is the MEM Address Register.
2. MAP bits 9-0 are transferred to/from S-bus bits 9-0.
3. MAP bits 11, 10 are transferred to/from S-bus bits 15, 14.
4. USR = User.
5. SYS = System.

7-34. GUIDELINES FOR WRITING LOADERS

Table 4-1 describes the HP IBL loader microprogram techniques, bit patterns for the Operator Panel registers, and information on the Remote Program Load Configuration Switches. Normally the HP supplied IBL microprograms will suffice for all user needs. If, however, you desire to write your own loader the guidelines outlined below may be of assistance. In addition, refer to the base set listing in appendix G (the IBL and Operator Panel microroutines) for examples of a workable loader and information on the use of the DES, LDR, DSPI, and DSPL micro-orders.

If you write your loader, it should be prepared *exactly* in the way you wish it to execute. The base set will configure the select code according to the information entered into the Operator Panel. One method that may work for you is to write the loader first in Assembly language then convert it to "machine code," then to a microprogram and finally, fuse the pROM's. If you have a double select code (i.e., magnetic tape or disc, SC10 and SC11, for example) the data channel select code should come first, then the command channel. In addition, follow these guides:

- There should be 64 (main memory) words or less designed to start at x7700, where x = 0, 1, 2, ... 7.

- All select codes in the loader I/O instructions will be configured at IBL time as follows:

  - S-register bits 11 through 6 will be taken as the configuring select code, 10 (octal) will be subtracted from the configuring select code and the result added to the select code part of all loader I/O instructions except: if the select code in a loader I/O instruction is less than 10 (octal), the select code will not be modified.

  - Note that loader constants having bit 15 on, bits 14 through 12 off, bit 10 on, and bits 8 through 6 anything but 000 (this prevents halts from being configured), will be interpreted as I/O instructions and will be configured as per the information just presented above.

- At IBL time:

  - Word 64 of the loader will be forced to the starting address of the loader in two's complement form.

  - Word 63 of the loader will be unconditionally configured as described above (i.e., S-register bits 11 through 6 will be taken as the configuring select code, etc.). The standard HP loaders use word 63 as DCPC Control Word 1.

7-35. SUMMARY

In using any of the guidelines and microroutine examples presented in this section you must make the final judgement as to "usability" and "workability" of the microprograms you create because of the wide range of applications for microprograms. The base set (appendix G) should be referred to as an example of "correct" microprogramming. Also, section 14 provides examples of microprograms you may be able to use.

With the completion of your study of this section you are prepared to write microprograms for use in the HP 21MX E-Series Computers. The use of microprogramming support software is also necessary and the following sections of the manual provide all the rest of the information you need.

Section 8
PREPRATION WITH THE MICROASSEMBLER

PREPARATION WITH THE MICROASSEMBLER

SECTION 8

With the information in this final section of part II you will be able to prepare your microprograms so that they will be accepted by the RTE Microassembler. If properly prepared, your microprogram will be processed (using information in section 9) to generate micro-object code which is ready to load into WCS for execution in the computer. The section provides:

- A suggested method for preparing your microprograms.
- A description of the microassembler character set, fields, and other rules for preparation.
- Microassembler control methods.
- Methods of making microprogram starting address assignments and making other modifications using the pseudo-microinstructions.

The information in this section requires as a prerequisite, a study of the preceding sections (particularly sections 4 and 6).

8-1. PLANNING AND PREPARATION

Using the information on the microassembler (starting in paragraph 8-6) you can prepare your microprogram for input to the microassembler on punched cards, paper tape, or magnetic tape cartridges. It is suggested, however, that it may be easier to prepare the microprogram on a disc file. To prepare a file containing a microprogram, use the RTE system Interactive Editor as outlined below.

8-2. PLANNING

Plan the microprogram essentially the same way as for an Assembly language program but base the objective on the concepts discussed in section 1. Steps that must be taken to achieve the objective should be clear and the logical sequence for the microprogram perhaps prepared in flowchart form.

To prepare a microprogram taking full advantage of your system's RTE Interactive Editor program (EDITR), all that is needed is pencil, paper, and the system console. The instructions given here are intended for use at the system console in a single-user environment. If you are operating in a Multi-Terminal Monitor (MTM) environment, it is assumed that you have taken the HP RTE training course or have the assistance of a person familiar with the MTM.

The EDITR program provides the tool for generating the source code, and the RTE FMGR program provides a means for storing microprogram sources as files. The files can be accessed later for editing and microassembling. Complete instructions for using these RTE system programs are beyond the scope of this manual which only provides guidelines for use to prepare and edit microprograms. Complete information on the EDITR and FMGR is provided in other documentation supplied with your RTE system.

8-3. PRELIMINARY INFORMATION. When preparing your microprograms using the EDITR, the first two lines of your microprogram should be the microassembler control instructions MICMXE and $CODE; the last line should be the psuedo-microinstruction END. Paragraph 8-6 provides all the details on the microassembler you will need. You should read through these or refer to them before actually going on-line. After the microprogram is written, press any key on the system console to get an RTE prompt character (*). Then type RU,FMGR and press the RETURN key. The system responds by outputting a FMGR prompt character (:). Type LS and press RETURN, the system outputs another FMGR prompt. Type RU,EDITOR and press RETURN; the system outputs SOURCE FILE? followed by the EDITR prompt character (/). Enter a space (blank) character and press RETURN; the system outputs EOF. At this point the system console should show the following:

```
*RU,FMGR
:LS
:RU,EDITR
SOURCE FILE?
/^
EOF
/
```

where:

^ means a space (blank) character.

Typing errors can be corrected by backspacing (or use a CONTROL H) then retyping the correct entry. After completing the above, make subsequent corrections using the EDITR as described in the EDITR documentation.

8-4. FIELD TEMPLATE

It should be noted at this point that if desired, you can prepare complete short microprograms using the Microdebug Editor. The starting column for each field in microinstructions is taken care of for you by the MDE in this case. Examples in section 14 use this method to illustrate and familiarize you with the microprogramming support software. Details on the Microdebug Editor are included in section 10.

The method you can use to identify the starting columns for microinstruction fields when preparing microprograms for input to the microassembler with the RTE Interactive Editor (as described in paragraph 8-3) is to use the Editor Tab function. So, at this point, to create a "pseudo-coding form" that will locate the starting point of each field (assuming you have followed the instructions in paragraph 8-3); enter the following after the EDITR prompt showing on the console:

```
T;10,15,20,25,30,40
```

Press RETURN and the system will output another EDITR prompt. You may now enter your microprogram as described in the next paragraph. Remember to enter a space after each prompt (/) to reach column one of your "coding form". Use the semicolon (;) key as a tab key to reach desired microinstruction fields.

8-5. MICROPROGRAM ENTRY

When you have a template (pseudo-coding form), enter your microprogram (prepared according to the rules to follow). Enter a space after each prompt (/) to reach column one of your "pseudo-coding form" (usually the EDITR "Tab" function) and terminate each line by pressing the RETURN key. You can list any line in your microprogram by entering the number of the desired line. After entering your complete microprogram, go back to line 1 and list the entire program by entering L*nn* (where *nn* is the number of lines in the program file) immediately following the EDITR prompt. Check the program for errors and make any corrections as necessary. Now assign the file a new name by entering EC*new* (where *new* is a new file name) immediately after the prompt. For example:

/ECJOE1

The system outputs the message END OF EDIT followed by a FMGR prompt. At this point you will have created a file that contains your first microprogram. If your system console is a teleprinter (TTY), you have a hard copy of your microprogram; if your console is a CRT terminal, obtain a hard copy on the system list device by using the FMGR LIst command (LI,*JOE1*). Check the copy and correct any errors.

8-6. THE MICROASSEMBLER

The RTE Microassembler translates symbolic microprograms into binary object code. The object code is produced in either a standard format recognized by the RTE Microdebug Editor and the WLOAD subroutine or a special format to be used as input to the HP ROM Simulator. The source may be entered from an input device or the RTE system LS tracks. (Microassembler execution will be described in section 9.) Object code may be generated to an output device as well as to a disc file. The microassembler can also produce a symbol table map, listing of source records and generated code, and a cross-reference symbol table which will all be described in section 9. The rules for preparation with the microassembler are described in this section. The hardware/software environment for the microassembler is described in section 3.

8-7. MICROASSEMBLER RULES

The RTE Microassembler accepts 72-character fixed-field source records (from the devices mentioned in paragraph 8-6). The 72-column format allows sequencing of card decks if you choose to prepare your source records on that type of medium. Each source record falls into one of the following categories:

- Comment
- Control command
- Microinstruction
- Psuedo-microinstruction

An asterisk in column one of a source record indicates that the entire microassembler source is a comment. Control commands are described in paragraph 8-8. The microinstruction source records that may be used are described in detail in section 4 (in particular see figures 4-3 and 4-4) but general requirements for microassembler use are discussed in this section. The psuedo-microinstructions are fully described in this section.

Where there are deviations from specifications for a particular type of source record (or field as described below) the difference will be so noted. Any ASCII character may appear in the comments source record (i.e., asterisk in column one). Most characters are legal in labels except as noted in paragraph 8-15. A space may only begin a field if no micro-order is specified in that field.

8-8. CONTROL COMMANDS

Control command source records affect external characteristics of the microassembly (e.g., listing and object code formats). The control command must start in the first column. Blanks are permitted only preceding and within comments following the control command. Control commands may be intersperced with other source records to specify control over the microassembly process. Certain control commands must be used (as mentioned in paragraph 8-3) in specific places in your microprograms. To wit: the first source record of your microprogram must be a "MIC" control command. There are options that may be used with some of the control commands and they are so noted in the description of each command that follows. There should be only one control command per source record. All control commands except MIC begin with a "$" (Dollar character) in column 1. No intervening spaces are allowed in any control statement other than as specified.

8-9. MIC ASSEMBLY COMMAND. For the E-Series or F-Series Computer, a MICMXE control command must be the first line in the source file. This command indicates whether the source is a M-Series or E/F-Series Computer microprogram, respectively, and specifies certain microassembly options. The form of the command for this computer is:

MICMXE,*p1,p2, . . .*

where:

"*p1, p2, . . .*" indicates a list of parameters. The parameters are optional and may appear in any order. The microassembly options are:

B = Output object code to the punch device.

R = Produce standard format object code.

S = Produce special format object code for the HP ROM Simulator.

L = List source and generated code on list device.

T = List a symbol table map on the list device.

C = Generate a cross-reference on the list device.

If "B" is not specified, no punched output is produced (this option does not affect the $CODE output). The "R" and "S" optional parameters are mutually exclusive; if neither is specified, the microassembler defaults to the format specified for the "R" parameter. The "R" and "S" parameters affect both the punched and $CODE (control command) output. (Note that the "B, R," and "S" parameters operate in a manner similar to Assembler conventions.) The "S" option is a special 32-microinstruction object code format. This special HP ROM Simulator format is reserved for system maintenance. Appendix E describes the format.

If the "L" option is not specified, only error and pass-completion messages will be written on the list device. $LIST commands will be ignored. The "T" option provides a listing of label names and the corresponding octal address used in the microprogram. The "C" option, and all the options for microassembler output are described in section 9.

An example of the use of the MIC control command (starting in column 1) would appear as shown below:

MICMXE,L,T

Here, note that the microassembler will default to the standard format object code.

8-10. THE $CODE COMMAND. The $CODE command directs object code to be written to the specified file. The command has the following form:

$CODE=*FNAME* [:*security* [:*crlabel*]] [,REPLACE]

The *"FNAME"* parameter specifies the name of the file to be created. For the "R" parameter, a type 5 file is created for the object code to permit a checksum of the records. A type 3 file is created for "S" format object code (to prevent a checksum of the records, which would be invalid due to the different format) blanks are not permitted between subparameters (as indicated in paragraph 8-8). The "%" notation for octal values generally accepted in the microassembler is treated as an alphanumeric character string here (to be consistent with RTE). If a file with the same name already exists and the REPLACE option is specified, the existing file is purged. Otherwise, object code is generated only to the punch device. The *"security"* and *"crlabel"* parameters indicate the file security code and disc cartridge label respectively; these sub-parameters are optional.

Object code generated to the $CODE file depends on the "R" or "S" option specified in the MICMXE command. For the suggested method of preparing your microprogram this control command should appear immediately after the MIC command.

8-11. $PAGE COMMAND. The $PAGE command causes a page eject and, optionally, replaces the heading during the listing of the microprogram. The forms of the command are:

$PAGE
$PAGE=*title*

The first form simply causes a page eject; the current heading is not altered. The second form, additionally, replaces the heading with the character string following the equal sign. The heading (*title*) is truncated after 60 characters. The $PAGE command is ignored when listing is disabled.

8-12. THE $LIST AND $NOLIST COMMANDS. The $LIST and $NOLIST commands have no parameters. The two commands control the source listing in the second pass of the microassembly. The $NOLIST command disables the listing of the source records and generated code until a subsequent $LIST command is encountered. These commands are ignored if the "L" option is omitted in the MIC assembly command.

8-13. $PUNCH AND $NOPUNCH. The $PUNCH and $NOPUNCH commands have no parameters. The effect that $NOPUNCH/$PUNCH have on the output depends on the object code format and the device. For "R" MIC command parameter format, disjoint code groups always cause a new (DBL) record to be written to the device of $CODE file. For "S", if the "missing" portion of code (between two disjoint code groups) does not extend beyond the buffer, the space is simply filled with microwords containing all 1 bits. Otherwise, leader or an end-of-file separates disjoint code groups on a punch device or $CODE file respectively (after padding the remainder of the buffer as before).

8-14. HP 1000 E-SERIES AND F-SERIES MICROINSTRUCTIONS

The format of the four microinstruction word types and all the micro-orders that can be used in the various fields are described in section 4 (in particular, figures 4-3 and 4-4). These source records can contain up to 72 characters with the legal field entries. To summarize section 4 information, the general uses for the four word types are defined below:

- Word type I executes:
  - Data transfers between main memory, the I/O section, and the Arithmetic/Logic section.
  - Logical and arithmetic functions on data.
- Word type II specifies data to be transferred to a specific register.
- Word type III executes a conditional branch based on flags or data values. When the OP field micro-order is "RTN", the address field (field 6) must be empty: comments must not appear before column 31. Field numbers are reviewed next.
- Word type IV executes an unconditional branch or microsubroutine branch.

Microinstruction source records and psuedo-microinstruction source records (to be described in paragraph 8-19) have similar fixed-field formats and are distinguished by the mnemonic in the OP field. Each microinstruction source record contains seven fields with the starting column of each field as follows:

FIELD	COLUMN	MEANING
1	1	Label
2	10	OP/Branch
3	15	Special, or Branch modifier
4	20	ALU, Branch Condition, or IMM modifier
5	25	Store, or Branch Sense
6	30	S-bus, Branch Address or, IMM operands
7	40	Comments (see allowable exception below)

A mnemonic in any field must begin in the first column of that field. The seventh, (Comment) field must be separated from the last field by at least one blank column. For word type I microinstructions, the Comment field must *not* appear before column 35.

As shown in figure 4-4, the fields are fixed for microassembly language source records. A few things to remember about the fields are:

- Field 1 can contain a label that is no longer than eight characters.
- Field 2 contains a micro-order no longer than four characters. This field can also contain a psuedo-microinstruction (refer to paragraph 8-19 for the explanation of psuedo-microinstruction mnemonics).
- Field 3 contains a micro-order no longer than four characters.
- Field 4 contains a micro-order no longer than four characters.
- Field 5 contains a micro-order no longer than four characters.
- Field 6 contains a micro-order no longer than four characters (word type I,) or an operand (word type II,) or an address (word types III and IV).
- Field 7 contains comments only. Field 7 ends in column 72.

Some additional comments on the fields follow.

8-15. THE LABEL FIELD. As mentioned above, a label (field 1) may be comprised of up to eight characters. The label may contain any ASCII character except a plus (+) or a minus (−). The first character must not be numeric or an asterisk (*), dollar sign ($), or a percent sign (%). Each label should be unique within the microprogram and cannot contain spaces within the label. Names which appear in EQU psuedo-microinstructions (refer to paragraph 8-19) may not be used as source record labels in the same microprogram.

8-16. MICRO-ORDERS. Fields two through six may contain any of the legal micro-orders used in word types I through IV. Refer to figure 4-4 for a list of the legal micro-orders. Word type II contains an operand in field 6 which must conform to the constrains listed in table 4-1.

8-17. ADDRESS FIELDS. Word types III and IV have address expressions in field 6. The address expressions may have one of the following forms:

number
label
label+*number*
label−*number*
*
*+*number*
*−*number*

The asterisk means "current address". If "*number*" is preceded by a percent sign (%) or followed by a "B", the string represents an octal quantity. For EQU psuedo-microinstructions, any "*label*" must have appeared previously in a Label field. Refer to the table 4-1 explanations of the Address fields for further information.

8-18. COMMENT FIELD. This optional field can be any string of characters up to the limit of the source record (column 72). If you have comments that are long you may use an asterisk source record in the next line.

8-19. PSEUDO-MICROINSTRUCTIONS

Psuedo-microinstructions have a direct affect on the object code generated; however, they are not composed of micro-orders as defined by the Control Processor. The format of pseudo-microinstructions differs slightly from that of the microinstructions. The fields are as follows:

FIELD	COLUMN(S)	MEANING
1	1-9	Label
2	10	OP
3	30-39	Operand

The Operand field may start in any column between 30 and 39 inclusive. A Comment field may start in any column, separated by at least one blank column from the last field. The pseudo-microinstructions that can be used include ORG, ALGN, END, EQU, DEF, ONES, and ZERO. The function and constraints for the use of each pseudo-microinstruction are included below. Note the CM address assignment and modification pseudo-microinstructions include ORG and ALGN. EQU and DEF are also used in conjunction with CM addressing.

8-20. THE ORG PSEUDO-MICROINSTRUCTION. The starting address of each microprogram must be assigned by an ORG pseudo-microinstruction. The form of the ORG pseudo-microinstruction source record is:

LABEL	OP	OPERAND
—	ORG	*expression*

The ORG pseudo-microinstruction specifies the control memory address of the subsequent microinstructions. An ORG must precede the first generated microinstruction. Subsequent ORG pseudo-microinstructions are permitted: however, the specified CM address must not be less than the address of the next microinstruction. If the first ORG is not included the microassembler will default to set the CM address of subsequent microinstructions to CM location 27000 (octal). The Operand field may be any expression. Any label must have appeared previously in a Label field.

Section 6 on mapping and section 2 provide information on CM locations and CM software entry points of which you should be aware before using the ORG in a microprogram. Since it is unlikely that any of your microprograms will use an entire module, you should organize (or "map") each of your modules to accommodate several microprograms. This is done by placing branch microinstructions in some (or all) of the module starting addresses that can be accessed by OCT main memory instructions. Each of these branch microinstructions should point to a microprogram located within the module. For example:

```
                                                ALU/
                        OP/     MOD/            MOD/                S-BUS/
LOCATION    LABEL       BRCH    SPCL            COND    STR         ADDRESS     COMMENTS

                                                .
                                                .
                                                .
                        ORG                                         27000B
            MICPR01     EQU                                         27011B
            MICPR02     EQU                                         27065B
                                                .
                                                .
                                                .
            MICPR07     EQU                                         27270B
            MICPR010    EQU                                         27315B
27000                   JMP     RJ30                                MICPR01     START ADDRESS 1
27001                   JMP                                         MICPR02     START ADDRESS 2
27002                   JMP                                         MICPR03     START ADDRESS 3
                                                .
                                                .
                                                .
27007                   JMP                                         MICPR07     START ADDRESS 7
27010                   JMP                                         MICPR010    START ADDRESS 10
                        END

* THE BEGINNING OF THE MICROPROGRAM WITH ENTRY POINT
* LABEL MICPR01 SHOULD THEN ORG AT LOCATION 27011B.

                                                .
                                                .
                                                .
```

Each label referenced by a JMP micro-order must be defined in a microprogram that maps the module. In most cases, the number of required starting addresses will be unknown until the number of prepared microprograms uses all (or almost all) 256 locations in a module. To allow for these cases, module addresses can include the RJ30 micro-order to modify the target address by using bits 3 through 0 of the OCT main memory instruction. The microprogram pointed to by using the JMP,RJ30 microinstructions should be simply a table of starting addresses of other microprograms. Examples of mapping techniques are discussed further in section 6.

Using the information provided and your present and anticipated microprogramming requirements, you can determine whether or not your module should be mapped. You should also be able to determine the starting addresses of some of your microprograms. The module mapping microprogram should consist of a MICMXE control command, an ORG psuedo-microinstruction specifying the first module location (e.g., 27000), a list of EQU pseudo-microinstructions associating values with labels, a sequence of branch microinstructions, and an END pseudo-microinstruction. After preparing and microassembling the mapping microprogram, load it into the desired Writable Control Store (WCS) board by using the microdebug editor (MDE) or WLOAD subroutine. (Refer to sections 10 and 11 for information on loading.) Once the module map is loaded into WCS, MDE or WLOAD can be used to load each microprogram into WCS beginning at the microprogram's starting address.

8-21. ALGN. The form of the ALGN psuedo-microinstruction is:

LABEL	OP	OPERAND
—	ALGN	—

ALGN alters the control memory address so that subsequent microwords start on a 16-word boundary (i.e., the next microword is located at the next address where the lower 4 bits of the address are zero). This is useful for setting the origin of tables which are indexed by the lower four bits of a branch microinstruction (i.e., using the RJ30, J74, etc., micro-orders). Examples of the use of ALGN (and some of the other pseudo-microinstructions) appear in part 4.

8-22. THE END PSEUDO-MICROINSTRUCTION. The form of the END pseudo-microinstruction is:

LABEL	OP	OPERAND
—	END	—

The END pseudo-microinstruction marks the end of a microprogram. This must be the last source record in any microprogram.

8-23. EQU. The form of the EQU pseudo-microinstruction is:

LABEL	OP	OPERAND
label	EQU	*expression*

The EQU pseudo-microinstruction associates the value of the *expression* with the label. This is useful for symbolically referencing locations external to the microprogram (i.e., branch target addresses). Examples of EQU might look like:

Character column:

	1	10	30
Fields:	Field 1	Field 2	Field 6
Content:	HALT RELO START	EQU EQU EQU	34000B 36000B RELO

8-24. DEF. The form of the DEF pseudo-microinstruction is:

LABEL	OP	OPERAND
label	DEF	*expression*

The DEF pseudo-microinstruction generates a 24-bit microword with the contents equal to the absolute value of the *expression* address in control memory. The "*label*" field may be left blank. Examples of the use of the DEF pseudo-microinstruction might look like:

Character column: 1, 10, 30

Fields:	Field 1	Field 2	Field 6
Content:	AD1	DEF DEF DEF	SRF+150 ASGNOP 416B

DEF is not normally used for user microprogramming.

8-25. THE ONES AND ZERO PSEUDO-MICROINSTRUCTIONS. The form of the ONES and ZERO pseudo-microinstructions are:

LABEL	OP	OPERAND
label	ONES	—
label	ZERO	—

The ONES and ZERO pseudo-microinstructions each generate a microword with the content equal to either all ones or zeros, respectively. The "*label*" field may be blank. An example of the use of ONES is:

Character column: 1, 10

Fields:	Field 1	Field 2
Content:	NEG 1	ONES

An example of using ZERO would be:

Character column: 1, 10, 40

Fields:	Field 1	Field 2	Field 7
Content:	NULL	ZERO	NO BITS

ONES and ZERO are not normally used for user microprogramming.

8-26. SUMMARY

The information presented thus far should bring you to the point where your microprogram is complete and ready to microassemble then execute using the information in part III. The control command and pseudo-microinstructions are summarized below.

- Control commands (start in column one):

  MICMXE,B,L,T,C,R(or S)
  $CODE=*FNAME*[:[*security*] [:[*crlabel*]]] [,REPLACE]
  $PAGE=*title*
  $LIST
  $NOLIST
  $PUNCH
  $NOPUNCH

- Pseudo-microinstructions:

Columns	1-9	10	30-39
	LABEL	**OP**	**OPERAND**
	—	ORG	*expression*
	—	ALGN	—
	—	END	—
	label	EQU	*expression*
	label	DEF	*expression*
	label	ONES	—
	label	ZERO	—

See figure 4-4 for a summary of all the micro-orders you have available for microinstructions.

PART III
Microprogramming Support Software and Hardware

Section 9
USING THE RTE MICROASSEMBLER

USING THE RTE MICROASSEMBLER

SECTION 9

This section provides instructions for actually microassembling your microprograms. The assumption here is that you have prepared your microprogram using the information from part II of this manual. It is also assumed that the RTE Microassembler is present in the RTE operating system. Refer to section 3 in this manual for guidelines on preparing for microprogramming. Some additional information on using the RTE system is provided but, for complete coverage, it is expected that you will refer to the RTE system manuals.

This section provides information on executing the microassembler and information on output such as:

- Binary object code
- Microassembled listings
- Symbol table output

In addition you will find information on the RTE Microassembler Cross-Reference Generator and microassembler messages output to the list device and operator's console.

9-1. USING THE MICROASSEMBLER

As described in section 8, the microassembler accepts fixed-field microprogram source records of up to 72 characters in length. Each source record contains either one microinstruction, one psuedo-microinstruction, or one microassembler control command. The microassembler processes the input source records and produces the binary object code of the microprogram. If specified by the initial microassembler control command (MICMXE), the microassembler also produces a microprogram listing in both symbolic and octal format, a symbol table, and error messages. Refer to sections 4 and 8 for descriptions of microinstructions acceptable by the microassembler. Section 8 also contains a description of pseudomicroinstructions and microassembler control commands. The following paragraphs provide a procedure for microassembling a microprogram. The procedure assumes that you are using the RTE system console and that the microassembler program, MICRO, is disc resident. If MICRO is available only on paper tape, load it using the RTE LOADR as described in the *RTE Operating Manual*. If the microprogram source is not in a disc file, MICRO can read it from some input device in the system. Section 3 provides more information on preparing to use microprogramming support software.

9-2. EXECUTION COMMAND

The microassembler may be scheduled in the RTE system with one of the following commands. All parameters are optional. (The instructions that follow this definition explain one method of executing the microassembler.)

RU,MICRO,*input,list,output,lines,console*
ON,MICRO,*input,list,output,lines,console*

- The *"input"* parameter indicates from what logical unit (LU) the source is to be read; the default is LU 5, an input device. If the *"input"* LU is 2, the system disc, the source is read from the system LS tracks. You must move the source onto the LS tracks prior to entering the ON command.

NOTE

If MICRO is run from the File Manager (:RU,MICRO), the *input* default is LU 1, not LU 5.

- The *"list"* parameter indicates to what logical unit the listing is to be written. The default is LU 6, the standard list device.
- The *"output"* parameter indicates to what logical unit the object code is to be directed. The default is LU 4, possibly a paper tape punch, or magnetic tape (some output device).
- The *"lines"* parameter indicates the number of printable lines on the list device, exclusive of a three-line header. The default is 56.
- The *"console"* parameter indicates the logical unit to which special messages are written. The default is LU 1, the operator console.

If the microprogram was prepared and stored in a disc file using the method suggested in section 8, perform the final edit and prepare to microassemble the program as follows:

- Press any key on the system console to get an RTE prompt (*). Then enter RU,FMGR to get a FMGR prompt (:). Make the following FMGR entries one at a time:

  LS
  MS, *name,* MICRO

  NOTE:

  ,MICRO required for RTE IV only.

  where:

  name is the name you assigned to the microprogram during program preparation. The system outputs the following:

  FMGR 015
  LS LU *lu* TRACK *trk*

  where:

  FMGR 015 is a "non-error" message, *lu* is the LU number of the disc, and *trk* the disc track number.

- Run the microassembler program by entering the following command after the FMGR prompt:

  RU, MICRO, 2, *list,output,lines,console*

  where:

  2 is the logical unit (LU) number of the disc LS track. In this procedure, it is assumed that the microprogram source was input to the disc as described above. If you are using some other input device, insert that device's LU number. If no input device is specified, this parameter defaults to LU number 1 or 5 as explained at the beginning of paragraph 9-2. The other parameters were explained previously.

- The program title, MICROASSEMBLER, is printed and pass 1 begins. If the "T" parameter is included in the MICMXE microassembler control command (in the source microprogram), the microassembler prints the symbol table at the conclusion of pass 1. Pass 2 begins immediately and the microassembler outputs the listing ("L" parameter) and if the "R" parameter was specified, relocatable object tape; this completes the microassembly.

NOTE

If pass 2 fails to begin, check that the "output" device is turned on. The microassembler will cycle in a loop until the punch is turned on.

Paragraphs 9-3 through 9-7 describe the various outputs of the microassembler. Error messages and information messages are described in paragraph 9-8.

9-3. THE MICROASSEMBLER OUTPUT

The following paragraphs describe all forms of output from the RTE Microassembler. The forms are:

- Binary object code.
- Source and octal microprogram listing.
- Symbol table.
- Messages.

The cross reference generator, which can be an output of the microassembler if the "C" option is specified in the MICMXE control command, is described in paragraph 9-7.

9-4. BINARY OBJECT CODE

The standard object code output by the microassembler to a disc file or some other output device consists of one or more microinstruction records. Appendix E shows the format as it appears on paper tape. One microinstruction record holds up to 27 microinstructions and 5 16-bit words of header information. Each source microinstruction requires 32 bits (two words) in the object format: an 8-bit address and 24 bits for the microinstruction. Therefore, the length of the microinstruction record comprises:

Five words of header plus 2n words for n microinstructions (two words for each microinstruction)

5 + 2n words for one microinstruction record.

The maximum number of microinstructions in one microinstruction record is 27. Consequently, the maximum record length equals 5+ (2× 27): 59 words. The last object record is a four-word End Record. When the microprogram consists of more than 27 microinstructions, a series of instruction records are produced with the last one haveing 27 or less microinstructions. For example, if 57 microinstructions are assembled, three microinstruction records and an End Record are produced as follows:

- Microinstruction record 1, consisting of 5 words of header and 54 words for 27 microinstructions: 59 words total.
- Microinstruction record 2, consisting of 5 words of header and 54 words for 27 microinstructions: 59 words total.
- Microinstruction record 3, consisting of 5 words of header and 6 words for 3 microinstructions: 11 words total.
- The End Record, consisting of 4 words.
- The total microassembler object code is 133 words for the microprogram.

The standard object format is accepted by all programs that accept standard relocatable format. Therefore, the object code can be stored from an imput device into a disc file as a binary relocatable by the FMGR STore command. If the microprogram includes a $CODE microassembler control command as described in section 8, the microassembler automatically stores the object code into a disc file.

The microassembler outputs non-standard HP ROM Simulator object code to the device if the "B" and "S" parameters are included in the MICMXE microassembler control command as described in section 8. Appendix E also shows the format of this type of object tape.

9-5. MICROASSEMBLER LISTING OUTPUT

The microassembler prints the microprogram source and the generated octal code on the system list device if the "L" parameter is included in the MICMXE microassembler control command (Refer to section 8 for details on MICMXE.) Appendix G (the base set) is an example of listing output. Section 14 provides examples of user microprograms. Note that from left to right the listing output contains a line number (decimal), the CM address (octal), the 24-bit microinstruction content at that address in octal form, then the seven fields of microinstructions.

9-6. SYMBOL TABLE OUTPUT

The microassembler prints a symbol table on the list device if the"T" parameter is included in the MICMXE microassembler control command (section 8). An example symbol table output is shown here. The actual content will, of course, depend upon your microprogram. The left column of the symbol table lists the symbols or labels used in the microprogram. Absolute octal addresses for the symbols are also output. If addresses are terminated by the letter "X" it indicates a symbol defined by an EQU pseudo-microinstruction in the microprogram.

SYMBOL TABLE

MOVE	032412X
GOTO	032421X
RET	032427X
LAST	032717X
OUT	032011
ERR1	032012

9-7. USING THE CROSS-REFERENCE GENERATOR

Assuming that the RTE Microassembler Cross-Reference Generator program is configured into the RTE software system, it is run automatically by the microassembler if the microprogram includes the "C" parameter in its MICMXE microassembler control command. However, you can run the generator independently by using either an RTE or FMGR command as follows:

ON,MXREF,*input,list,lines, console*

RU,MXREF,*input,list,lines,console*

The parameters are optional and correspond to those defined for the microassembler execution command described in paragraph 9-2. Informative messages and error messages output by the Cross-Reference Generator (MXREF) are described in paragraphs 9-8 and 9-9. Additional points about the Cross-Reference Generator follow:

- MXREF does not flag erroneous statements. In fact, MXREF looks at only the label and expression fields, using field 2 and, in some cases, field 3 to determine the instruction format.

- Statements which contain invalid mnemonics in field 2 are treated as word type IV micro-instructions, causing field 6 to be cross-referenced as an expression.

- MXREF will cross-reference characters in the label and expression fields of statements which do not permit labels or expressions.

- In the cross-reference output, the first line number is the line on which the symbol was defined (ie., appears in the label field); subsequent line numbers are lines on which the symbol was referenced. (If the symbol appears in the label field of more than one statement, subsequent "definitions" are cross-referenced as references to the first occurrence.)

- MXREF flags undefined and unreferenced symbols with the messages:

  NOT DEFINED
  NOT REFERENCED

- The output does not exceed 72 characters per line.

- MXREF outputs some summary statistics which may be of general interest, viz.:

  number of symbols (defined and undefined)
  number of references (excluding definitions)
  number of source lines (including control commands).

The first four mentioned above allow MXREF to cross-reference programs which may not be correct micro-programs. The resulting cross-reference listing may be useful in determining the external symbols which must be defined with an EQU statement, or in finding all references to a misspelled symbol. An example MXREF output is shown below.

```
PAGE 0001 RTE MICRO CROSS-REFERENCE REV.A 760718

SYMBOLS=0012  REFERENCES=0013  SOURCE LINES=0144

COMPARE    0071  0134
ENDCHK     0133  0105
EXIT       0143  0045  0055
HORI       0030  0115
INTCHK     0105  0087  0090
INTEXIT    0112  **NOT REFERENCED**
INTRTN     0122  0040
SETY       0050  0139
SORT       0036  0031
STRTPASS   0062  0138
SUBTRACT   0089  0085
SWAP       0096  0088
```

9-8. MESSAGES

The microassembler and Cross-Reference Generator output two kinds of messages. Error messages are output to the system list device; informative messages are output to either the system list device or to the operator's console (which is not necessarily logical unit 1). Informative messages and error messages are described in paragraphs 9-9 and 9-10 respectively.

9-9. INFORMATIVE MESSAGES

The applicable one of these two messages are printed on the system list device:

END OF PASS *n*: NO ERRORS

This is the normal pass-completion message where *n* is the pass number.

END OF PASS *n*: *e* ERRORS

This message indicates the number of errors detected during the pass; *n* is the pass number and *e* is the number of error messages.

The messages that can be output to the operator's console follow:

/MICRO: RE-INPUT SOURCE AND *GO

This message means that the microassembler was unable to get necessary disc tracks when the microprogram source was input from a device other than the disc. To recover, reposition the source, and schedule the microassembler with the RTE GO command (GO,MICRO, etc.). This-message can appear between the two microassembly passes and before the cross-reference generation.

/MICRO: END

This is the normal conpletion message for the microassembler.

/MICRO: END WITH ERRORS

Error messages appear on the list device.

/MICRO: ABORT

This message means that the microassembler detected an irrecoverable error and aborted.

/MXREF: END

This is the normal completion message for the Cross-Reference Generator.

/MXREF: RE-INPUT SOURCE AND *GO

Same as for the microassembler RE-INPUT message except applicable to the Cross-Reference Generator when the "C" option's used with the "MIC" control command.

/MXREF: ABORT

This message indicates that a irrecoverable error was detected in the Cross-Reference Generator.

9-10. ERROR MESSAGES

The microassembler checks each microinstruction for errors during microassembly. If an error is detected, an error message is written to the list device. Following all error messages for a source record, the source record itself is printed. The form of the error message is:

**ERROR *e* IN *ln1* (See *ln2*) *message*:

where:

e is an error number defined in table 9-1;

ln1 is the line number of the source line containing the error;

ln2 is the line number of the previous source line (if any) containing the same error.

message is the error message.

Table 9-1 gives the complete meaning of each error message recovery procedure, and/or the microassembler action taken.

Table 9-1. Microassembler and Cross-Reference Generator Error Messages

ERROR NUMBER	MESSAGE/MEANING/RECOVERY
1	DUPLILCATE LABEL IN FIELD 1. The microinstruction label is the same as a previously used label or EQU symbol. This occurrence of the symbol is ignored and its first definition holds.
2	INVALID OP IN FIELD 2. A NOP micro-order is inserted in field 2.
3	INVALID SPECIAL IN FIELD 3. A NOP is inserted in field 3.
4	INVALID CONDITION IN FIELD 4. An ALZ is inserted in field 4.
5	INVALID ALU IN FIELD 4. A PASS micro-order is inserted in field 4.
6	INVALID MODIFIER IN FIELD 4. A HIGH micro-order is inserted in field 4.
7	INVALID STORE IN FIELD 5. A NOP is inserted in field 5.
8	INVALID S-BUS IN FIELD 6. A NOP is inserted in field 6.
9	INVALID SENSE IN FIELD 5. Micro-order in field 5 is not RJS and is ignored.
10	MISSING ORG. Origin is set to 27000B.
11	INVALID CONSTANT IN FIELD 6. The Operand of a word type II microinstruction is out of range. A value of 0 is inserted in field 6.
*12	$CODE IGNORED: NO BUFFER SPACE. Insufficient memory for object code buffer. Object code is only punched on tape (if B parameter included in MICMXE microassembler control command).
*13	$CODE IGNORED: CANNOT BUILD FILE. Object code is punched only on tape (if B parameter included in MICMXE microassembler control command. This message is followed by the FMGR error code.

Table 9-1. Microassembler and Cross-Reference Generator Error Messages (Continued)

ERROR NUMBER	MESSAGE/MEANING/RECOVERY
*14	INVALID FILE REFERENCE. Syntax error occurred in *filename, security,* or *crlabel* specification. (Refer to the *Batch and Spool Manual.*) Object code is only punched on tape (if B parameter included in MICMXE microassembler control command).
15	NOT TYPE-3 SPECIAL IN FIELD 3. A NOP is inserted in field 3.
16	NOT TYPE-1 or 2 SPECIAL IN FIELD 3. A NOP is inserted in field 3.
17	NOT TYPE-4 SPECIAL IN FIELD 3. A NOP is inserted in field 3.
*18	INVALID CONTROL COMMAND. The microassembler assumes the parameter defaults of the MICMXE control command.
19	INVALID EXPRESSION IN FIELD 6. Branch address is out of permitted range, or target label address is undefined. A value of 0 is inserted into field 6.
**20	NO SOURCE. Microprogram source input device is not ready or the microassembler program (MICRO) was given incorrect input device LU number. Check input device; and MICRO command. Make necessary correction and microassemble again.
*21	MISSING END. The microprogram has no END statement. Correct and microassemble again.
*22	SYMBOL TABLE OVERFLOW. The microprogram has too many labels; or insufficient memory to build symbol table.
23	ADDRESS OUT OF RANGE IN FIELD 6. Branch address is out of permitted range. A value of 0 is inserted into field 6.
*24	LABEL NOT ALLOWED IN FIELD 1. The characters in field 1 are ignored.
*25	FIELDS 4 & 5 MUST BE BLANK. These fields are ignored in word type IV instructions.
26	ADDRESS SPACE OVERFLOW. Branch address is greater than 37777B (16383). A value of 0 is inserted into field 6.
**27	INVALID OR MISSING MICRO COMMAND. The MICMXE microassembler control command is incorrect or missing; microassembly aborts. Correct the line and microassemble again.
*28	DUPLICATE MICRO OPTION IGNORED. A parameter appears more than once in the MICMXE control command. The first appearance is accepted; the others are ignored.
*29	FILE I/O ERROR. This message is followed by a FMGR error code. Object code is punched only on tape (if B parameter included in MICMXE microassembler control command).
**30	INVALID MICRO OPTIONS. A microassembler control command has incorrect parameter(s). The parameter(s) is ignored.
*31	INVALID LABEL IN FIELD 1. The label contains a plus (+) or minus (–) sign or begins with a percent (%) character.
*32	SECOND $CODE IGNORED. Only one $CODE control command is allowed; subsequent ones are ignored.

Table 9-1. Microassembler and Cross-Reference Generator Error Messages (Continued)

ERROR NUMBER	MESSAGE/MEANING/RECOVERY
*33	EXPRESSION NOT ALLOWED IN FIELD 6. The characters in field 6 are ignored.
	CROSS REFERENCE GENERATOR MESSAGES
1	SYMBOL TABLE OVERFLOW
2	NO SOURCE

NOTES:

1. Messages flagged with a single asterisk (*), have no effect on generated code. Non-recoverable errors are flagged with a double asterisk (**).

2. Unless the microassembly process is aborted (/MICRO: ABORT message listed on system console), you can correct any of the above errors by using the Microdebug Editor and execute the microprogram from WCS. However, the resulting object code is not suitable for burning pROM's. To burn pROM's, you must correct the microprogram source and reassemble to get an error-free object code direct from the microassembler.

Section 10
USING THE RTE MICRODEBUG EDITOR

USING THE RTE MICRODEBUG EDITOR

SECTION 10

The Microdebug Editor (MDE) allows you to load microprogram object code into WCS, debug the code, and execute the microprogram. Using the debugging features as illustrated in section 14, you may also write short microprograms using the MDE. In order to use MDE, it is necessary that the WCS boards be assigned subchannel base addresses or initialized for the transfer of the microcode. Complete information required to write WCS initialization programs is given in the Driver DVR36 Manual. Example WCS initialization procedures are included in section 14.

MDE provides its own prompt character ($) and responds to its own set of operator commands. When you use MDE, you must observe the operator command syntax (described in table 10-1) and the following conventions:

- A numeric parameter is assumed to be positive unless preceded by a minus sign (−).
- A numeric parameter with the letter "B" suffix indicates the parameter is octal. Otherwise the numeric parameter is assumed to be decimal.
- Two adjacent commas (,,) or colons (::) mean a parameter assumes its default value.
- Leading blanks (spaces) and blanks preceding or following a comma or a colon are ignored.
- All inputs must be terminated by a carriage return (CR).

Table 10-1. MDE Operator Command Syntax

ITEM	MEANING
UPPER CASE	These characters are literals and must be specified as shown.
lower case	These characters only indicate the type of information required.
REad	This combination means that the RE is literal and must be used as shown; the remaining characters are for information only and need not be used.
[,*item*]	Items within brackets are optional. You can default the item by omitting it or by replacing it with a comma if other items follow it.
[,*item1* ,*item2* ,*item3*]	This indicates that any one of the items listed may be used. You can default the selection by omitting it or by replacing it with a comma if other items follow it.
item1 *item2* *item3*	This indicates that one of the items listed must be used.
namr	This indicates one parameter with up to two subparameters separated by colons. Subparameters are allowed on the first parameter only. Examples: *namr=filename* [:*security code* [:*crlabel*]] -and- *namr*= logical unit number

10-1. SCHEDULING MDE

You can schedule the Microdebug Editor program (MDEP) by using either an RTE ON command or an FMGR RU command. (MDEP can also be called by another program as shown at the end of this section.) To schedule MDEP use either of the following commands:

ON,MDEP[,*lu1*[,*lu2*[,*lu3*[,*lu4*]]]]

RU,MDEP[,*lu1*[,*lu2*[,*lu3*[,*lu4*]]]]

where:

lu1 is the logical unit (LU) number of the console you are going to use to communicate with MDE;

lu2 is the LU number of the WCS board you will be using;

lu3 is the LU number of an additional WCS board (if required);

lu4 is the LU number of a third WCS board (if required).

Upon initial execution, MDE must determine the computer type you are using by making the following request:

```
COMPUTER TYPE: 1=21MX,2=21MX E-SERIES
TYPE(1 OR 2)?
NOTE: 2 is also the response for F-Series Computers
```

You must respond by entering the number "2". This request will not appear with any subsequent use of MDE unless the RTE system is re-booted or MDE is rescheduled.

MDE requires the driver DVR36 and WCS I/O Utility routine WLOAD for its operations. MDE locks all WCS logical units in a WCS LU table (WCSLT); any LU's added to the WCSLT are also locked. You can load, read, modify, debug, and dump microprogram object code by using MDE operator commands. MDE, when used as routine MDES, may also perform these operations in your applications environment. The MDE operations work with all the WCSLT LU's and with control memory addresses issued by the operator commands. Termination of MDEP (or the MDES calling program) unlocks all WCS logical units.

10-2. MDE COMMANDS

Table 10-2 summarizes the commands for using the MDE; more detailed explanations of the commands are given below. MDE will not allow operations in the base set area of control memory. The valid range of control memory address parameters is 2000 through 37777 octal. MDE outputs a dollar sign ($) character as a prompt.

Table 10-2. Summary of Microdebug Editor Commands

CONTROL COMMANDS	DESCRIPTION
??	Explains error code.
EX	Terminates MDE.
I/O COMMANDS	**DESCRIPTION**
DU	Dumps specified binary object code of current WCS-resident microprogram(s) to a LU or disc file.
LD	Loads microprogram binary object code onto WCS (write verified).
LU	Add or delete WCS logical units to or from a WCS LU table (WCSLT).
EDIT COMMANDS	**DESCRIPTION**
DE	Delete microinstruction at specified control memory addresses by replacing with NOP's.
RE	Replace microinstruction at specified address.
SH	Show microinstruction at specified address on the operator console.
DEBUG COMMANDS	**DESCRIPTION**
BR	Set breakpoint into microprogram at specified control memory address.
CL	Clear breakpoint in microprogram at specified control address.
LC	Locate object code in control memory for use with breakpoint.
PR	Set up additional parameters for use with next MDE RU command.
RU	Execute microprogram by executing the appropriate main memory instruction.
SE	Set registers to values desired for next execution of MDE RU command.

10-3. ?? COMMAND

This command expands an MDE error code. (MDE error codes are listed and defined in table 10-3.) The command format is:

??[,*number*]

where:

number is the error number. If *number* is omitted, the last error code issued is expanded. If *number* is *xx*, error code *xx* is expanded. If number is 99, all error codes are expanded. (Refer to table 10-3)

10-4. EXIT COMMAND

This command terminates the MDE. (If in MDES, returns to calling program.) The command format is:

EXit

10-5. DUMP COMMAND

This command transfers the contents of WCS to a file or logical unit. The command format is:

DUmp,*namr1*[,*xxxxx*[,*yyyyy*]]

where:

namr1 is the logical unit number or the name of a file to which the object code is to be transferred. If *namr1* is a file, the file is created by this command.

xxxxx and *yyyyy* are the upper and lower control memory addresses of the object code to be transferred. The range *xxxxx* to *yyyyy* inclusive are transferred for all LU's in the WCS logical unit table (WCSLT). If *xxxxx* and *yyyyy* are zeros (default values), all logical units in the WCSLT are transferred.

10-6. LOAD COMMAND

This command loads the binary object code into WCS; the entire load is write verified. The command format is:

LD,*namr1*

where:

namr1 is the logical unit number or the name of a file from which binary object code is to be transferred. If *namr1* is a file, it may have been created by the DU command or by microassembly of a $CODE control statement.

Any microprograms residing in WCS that are overlayed by an LD command are lost.

10-7. LU COMMAND

This command adds or deletes WCS logical units to or from the WCSLT and enables or disables WCS LU's that are in the WCSLT. The command format is:

LU[,*lu1*[,*lu2*[,...*lux*]]]

where:

lu1, lu2, etc. are WCS LU's for MDE use. A maximum of 12 LU entries are permitted. A negative LU number causes the LU to be deleted from the WCSLT. An LU entry prefixed by the letter "E" logically enables that LU and, prefixed by the letter "D" disables that LU. (The WCS board or boards must already be physically enabled.) Valid LU numbers must be in the range 0 through 63.

MDE responds to the LU command by outputting a status table as follows:

LU#	RANGE	STATUS
lu1	*xxxxx-yyyyy*	*z*
lu2	*xxxxx-yyyyy*	*z*
.		
.		
lux	*xxxxx-yyyyy*	*z*

where:

lu1, lu2, etc., are the WCS LU's currently used by MDE;

xxxxx-yyyyy is the range of control memory set for a particular LU;

z is "1" for an enabled LU, "0" for a disabled LU (disabled includes downed LU's), or "P" for a pseudo-disabled (physically-enabled) LU.

The LU command adds LU's to the WCSLT in the order they are entered. If the LU parameters are defaulted, the current WCSLT is displayed. All LU's in the WCSLT are locked by MDE and released when MDE or the calling program is terminated.

10-8. DELETE COMMAND

This command deletes a microinstruction or range of microinstructions from WCS. The deleted microinstructions are replaced by NOP micro-orders (PASS in the ALU field). The command format is:

DElete,*xxxxx*[,*yyyyy*]

where:

xxxxx and *yyyyy* are the lower and upper control memory addresses of the range of microinstructions to be deleted. If *yyyyy*=0 (default), only *xxxxx* is deleted.

10-9. REPLACE COMMAND

This command replaces a microinstruction or range of microinstructions in WCS. The command format is:

REplace,*xxxxx*[,*yyyyy*[,*O*]]

where:

xxxxx and *yyyyy* are the lower and upper control memory addresses of the range of microinstructions to be replaced. If *yyyyy*= 0 (default), only *xxxxx* is considered. The optional letter "O" causes the object code as well as the micro-orders of each microinstruction to be displayed as each replace is made.

MDE responds to the REPLACE command as follows:

xxxxx field2 field3 field4 field5 field6 zzz zzzzzz
$$

where:

field2 through *field6* are the micro-orders of the microinstruction at control memory address *xxxxx* and *zzz zzzzzz* is the object code of the microinstruction. $$ is a prompt for your response.

You may respond to the $$ prompt as follows:

nfield2,nfield3,nfield4,nfield5,nfield6
www wwwwww

/ or nn or A

where:

nfield2 through *nfield6* are the desired replacement micro-orders for each field of the new microinstruction. The field micro-orders must be entered in the order shown. If any field is defaulted by ,, or omitted, that field remains the same as in the original microinstruction.

www wwwwww is the new microinstruction (in octal) displayed by MDE if the REPLACE command was used with the optional letter "O". If *www* or *wwwwww*= 0 (default), the old value remains.

leaves the current microinstruction unchanged and moves to the next one. If control memory address *yyyyy* is exceeded, the REPLACE command is terminated.

nn is a positive integer from 1 through 99 and causes the REPLACE command to move its pointer *nn* locations in control memory, displaying each microinstruction as it increments. If *yyyyy* is not exceeded, the last microinstruction displayed is the one ready to be replaced. If *yyyyy* is exceeded, the REPLACE command is terminated.

The letter "A" terminates the REPLACE command; all the remaining microinstructions are unchanged.

Each time a microinstruction is replaced the new microinstruction is microassembled and the REPLACE command pointer moves to the next microinstruction. If *yyyyy* is exceeded, the REPLACE command is terminated.

10-10. SHOW COMMAND

This command displays a microinstruction or range of microinstructions residing in WCS. The command format is:

SHow,*xxxxx*[,*yyyyy*[,O]]

where:

xxxxx and *yyyyy* are, respectively, the lower and upper control memory addresses of the range of microinstructions to be displayed. If *yyyyy*=0 (default), only *xxxxx* is displayed. The optional letter "O" causes the object code as well as the microinstruction to be displayed.

MDE responds to the SHOW command as follows:

xxxxx field2 field3 field4 field5 field6 zzz zzzzzz

.

.

.

yyyyy field2 field3 field4 field5 field 6 zzz zzzzzz

where:

field2 through *field6* are the micro-orders of the microinstruction at a particular control memory address and *zzz zzzzzz* is the object code of the microinstruction.

10-11. BREAKPOINT COMMAND

This command sets a breakpoint or breakpoints at a control memory address or addresses. This command may also simply display the current set of breakpoints. The command format is:

BReakpoint[,*break1*[,*break2*[,*break3*]]]

where:

break1, break2, and *break3* are the control memory addresses of the breakpoints to be set. If *break1*=0 (default), the current set of breakpoints is displayed. The maximum number of breakpoints that can be set is three.

MDE responds to the BREAKPOINT command as follows:

BREAK1 *xxxxx*
BREAK2 *xxxxx*
BREAK3 *xxxxx*

where:

BREAK1, BREAK2,and BREAK3 designate the breakpoints and *xxxxx* is the control memory address of a breakpoint.

Before setting a breakpoint, you must locate the desired control memory address by using a LOCATE (LC) command. Also, observe the following rules when using breakpoints:

- When a breakpoint executes, all registers (except the counter) that can be displayed by the SET command (paragraph 10-16) are saved. Note that the IR and the M-register are two of the registers that are not saved.
- A breakpoint cannot be set on a microinstruction that uses any bits in the Instruction Register.
- A breakpoint can be set within a microsubroutine but, if this is done, it cannot be reentered.
- A breakpoint cannot be set at the control memory address of a microinstruction passing data from the T-register within two microinstructions following a READ micro-order.
- A breakpoint can be set on a conditional branch microinstruction but it cannot be reentered.
- A breakpoint may be set on a microinstruction that uses a register which is lost when breaking; however, the register will not be restored if execution continues.
- A breakpoint may be set on a microinstruction that uses any one of a set of Special micro-orders but continued execution will be unpredictable. This set of Special micro-orders is: INCI, IOFF, IOG, IOI, ION, and IOO.
- Breakpoints cannot be set in the CM area occupied by the MDE breakpoint object code.
- If there is no control memory entry point address available for MDE, debug operations using breakpoints cannot be performed.
- If you do not have enough room in control memory for your microprograms and the MDE object code, either you must overlay some of your object code or debug operations using breakpoints are not allowed.
- The counter cannot be saved on the E-Series or F-Series Computer.

10-12. CLEAR COMMAND

This command clears breakpoints previously set by a BREAKPOINT command. The command format is:

CLear[,*break1*[,*break2*[,*break3*]]]

where:

break1, break2, and *break3* are the control memory addresses of breakpoints to be cleared. If *break1* = 0 (default), then all breakpoints are cleared. The maximum number of breakpoints that can be cleared is three.

10-13. LOCATE COMMAND

This command locates the breakpoint object code in control memory to enable breakpoints to be set. Also, this command moves breakpoint object code from a buffer in memory to control memory. The command format is:

LC,*xxxxx*,*yyyyy*

where:

xxxxx is the starting control memory address of the sequence of breakpoint object code. The object code is moved and will occupy up to 114 (162 octal) control memory locations beginning with *xxxxx*. Location *yyyyy* is the breakpoint reentry point in control memory. Location *yyyyy* must be a valid control memory entry point address but must not be used by any microprograms.

As an example of LOCATE command usage, suppose a microprogram occupies CM addresses 34020B to 34153B and the breakpoint object code can be placed into "unused" addresses 34200B to 34362B. Assuming that entry point 34002B is not used by a microprogram, the example LOCATE command would be:

LC,34200B,34002B

Every time the LOCATE command is used all breakpoints are cleared; they can be reset with the BREAKPOINT command for use with the relocated object code. Breakpoint object code can be located across two WCS LU's provided that both LU's are enabled.

10-14. PARAMETERS COMMAND

This command sets up parameters in memory for use with the main memory instruction that calls the microprogram to be executed. These parameters are in addition to those that may be passed via registers. The command format is:

PR

MDE responds as follows:

P+ 1=*contents1*
P+ 2=*contents2*
P+ 3=*contents3*
P+ 4=*contents4*
P+ 5=*contents5*
P+ 6=*contents6*
P+ 7=*contents7*
P+ 8=*contents8*
P+ 9=*contents9*
P+ 10=*contents10*

P+*x*=

where:

P+ 1,P+ 2, etc., are the memory locations relative to the instruction that calls the microprogram; *contents1, contents2,* etc., are the octal contents of each location; x is an integer from 1 through 10; and P+*x*= is a prompt for you to enter new contents or leave the old contents unchanged.

Each location in the range P+ 1 through P+ 10 is displayed one at a time (followed by the prompt P+ x=) to allow you to create the desired calling instruction parameters. You can respond to the prompt with the following:

/ or R or *xxxxx* or DEF.*yy* or A

where:

The / character leaves the current location unchanged; the letter "R" designates the current location as a valid return address for the microprogram; *xxxxx* is a decimal number from -32767 through 32767 or an octal number from -77777B through 77777B; DEF.*yy* creates a DEF to address P+*yy*; the letter "A" terminates the PARAMETERS command and all remaining locations are left unchanged.

10-15. RUN COMMAND

This command executes a microprogram. If required, program parameters can be preset using the PARAMETERS or SET commands.

CAUTION

It is strongly recommended that your RTE system be in a non-critical or a single-user operating mode before you execute a microprogram. Execution of an unproven microprogram can have unpredictable and undesirable results, including the destruction of the system.

The command format is:

RUn [,105*yyy*B / ,101*zzz*B]

where:

105*yyy*B and 101*zzz*B are OCT instruction values corresponding to control memory entry point addresses;

yyy and *zzz* are octal values which you should predetermine by using the information given in section 6.

If you default the optional RUN command parameters, the RUN command will do one of two things depending on the last return from microprogram execution. If the last return was from a breakpoint, the RUN command will resume execution at the most recent breakpoint. If the last return was a normal return, the RUN command will reexecute the last main memory instruction used to link with the microprogram. When a RUN command executes, one of the following messages should be output upon return from microprogram execution:

RETURN P+*xx*

where:

xx is a decimal number from 1 through 10 and the message indicates a normal return, or

BREAK *yyyyy*

where:

yyyyy is the address of a breakpoint and the message indicates a return from a breakpoint.

Note that the RUN command cannot enable a disabled WCS LU.

10-16. SET COMMAND

This command sets the saveable registers for the next RUN command. This command also displays the contents of the saveable registers at the last break in the execution or last return from a RUN command. The command format is:

SEt[,*p1*[,*p2*...[*p25*]]]

where:

p1, p2, etc., are any of the following:

A (A-register)	S1
B (B-register)	S2
X (X-register)	S3
Y (Y-register)	S4
O (O-register)	S5
E (E-register)	S6
S (S-register)	S7
L (L-register)	S8
P (P-register)	S9
FLAG (CPU Flag)	S10
DSPL (Display Register)	S11
DSPI (Display Indicators)	SP (Stack Pointer)
CNTR (Counter) Always= 0	

If the SET command is given without any parameters, all register values are shown.

MDE responds to the SET command by displaying any of the requested values as follows:

A=*xxxxxx*	FLAG=*x*	S5=*xxxxxx*
B=*xxxxxx*	DSPL=*xxxxxx*	S6=*xxxxxx*
X=*xxxxxx*	DSPI=*xx*	S6=*xxxxxx*
Y=*xxxxxx*	CNTR= 0	S7=*xxxxxx*
O=*x*	S1=*xxxxxx*	S8=*xxxxxx*
E=*x*	S2=*xxxxxx*	S9=*xxxxxx*
S=*xxxxxx*	S3=*xxxxxx*	S10=*xxxxxx*
L=*xxxxxx*	S4=*xxxxxx*	S11=*xxxxxx*
P=*xxxxx*		SP=*xxxxxx*

Register n=xxxxxx

Register n=

where:

x, xx, xxx, or *xxxxxx* are the contents or the condition of a particular register or flag in octal or binary; *Register n* is the first register in your set of registers and *Register n=* is a prompt for you to enter a new value in register *n* or leave the old unchanged.

The prompt is displayed after each requested register. You can respond to the prompt with the following:

/ or *xxxxx* or A

where:

/ leaves the current register unchanged and moves to the next requested register; *xxxxx* is an octal number from -77777B to 77777B or a decimal number from -32767 to 32767; and the letter "A" terminates the SET command and all remaining registers are left unchanged. Note that MDE always outputs octal numbers.

All registers except A, B, X, Y, O, E, and DSPL are set to zero for a normal return from microprogram execution. The counter cannot be used with breakpoints. All other registers not saved by MDE cannot be assumed to remain in a given state during debug operations.

NOTE

All numbers output from the MDE are in octal. MDE does not designate this however. If you are entering numbers and you desire octal form, so designate by following the number with B.

10-17. MESSAGES

Table 10-3 lists all MDE error messages.

Table 10-3. Microdebug Editor Error Messages

ERROR CODE	MESSAGE/MEANING
MDE000	MDE BREAK. Break set into program ID segment.
MDE001	WCSLT FULL. WCS logical unit table is full. Use the LU command to display current entries in table and to delete unwanted LU's.
MDE002	ILLEGAL PARAMETER. Illegal parameter or subparameter in input.
MDE003	WCSLT LU LOCKED. One or more WCS LU's in the WCSLT are already locked by another program.
MDE004	NO RN AVAILABLE. A resource number to lock WCS LU'S is not available.
MDE005	INPUT ERROR. Illegal command or command syntax incorrect.
MDE006	ILLEGAL LU. LU given to MDE is not driven by driver DVR36.
MDE007	ILLEGAL DEVICE. Attempted I/O operation with a device having equipment type (driver number) of 30 or higher.
MDE008	ERROR # UNDEFINED. The error number specified does not exist.
MDE009	LU # UNDEFINED. The LU number given to MDE to be removed from the WCSLT is not in the WCSLT.
MDE010	CHECKSUM OR REC. FORMAT ERROR. Invalid record format or checksum error.
MDE011	NO LU'S. WCS can't be loaded or dumped because the WCSLT is empty or has no LU's set up for the desired control memory address range.
MDE012	VERIFY ERROR. A write verify error occurred during the last I/O operation to WCS.
MDE013	NO DCPC. The last requested I/O operation did not complete due to a non-responding DCPC channel.

Table 10-3. Microdebug Editor Error Messages (Continued)

ERROR CODE	MESSAGE/MEANING
MDE014	INVALID ADDRESS. Invalid WCS address specified; or last requested I/O operation did not complete; or attempted to set a breakpoint in MDE microcode or on a reentry address; or attempted to clear non-existent breakpoint; or attempted to set reentry address in MDE microcode; or locate not completed.
MDE015	ADDRESS CONFLICT. The address associated with and assign base address, enable, or write request conflicts with another WCS subchannel. Last requested I/O operation did not complete.
MDE016	DATA OVERRUN. The loading of data into WCS overran the available WCS. Loading is partially complete.
MDE017	LU DISABLED. A WCS LU requested for an I/O operation is psuedo-disabled, disabled, or down.
MDE018	FMP ERROR -XXXXX. An FMP call resulted in the error condition described by the listed error code (-XXXXX). Refer to FMP error codes in the Batch-Spool Monitor manual.
MDE019	I/O ERR EOF EQT XX. An end-of-file occurred on EQT entry number XX.
MDE020	MICRO ERR XX. Microassembler error XX occurred during a REPLACE command.
MDE021	ILLEGAL REGISTER. The register requested by a SET command is not valid for MDE.
MDE022	NO MACRO. The attempted RUN command had no prior main memory instruction call to a microprogram; or attempted setting a breakpoint without MDE breakpoint microcode located; or breakpoint reentry address not a valid control memory entry point address or no WCS LU contains the reentry address.
MDE023	USER MICRO ERR. User microprogram returned incorrectly.
MDE024	BKTBL FULL. Breakpoint table is full. Use CL command to delete some breakpoints before trying to set new ones.

10-18. RESTRICTIONS ON USING THE MICRODEBUG EDITOR

Microprograms provide you with a very privileged mode of computer operation. In an RTE operating system, a microprogram executes beyond the control of the RTE system and, if improperly designed, can destroy the system. This means that it is imperative that you exercise an extra measure of caution before executing a developmental microprogram.

Subroutine MDES locks all WCS LU's that it uses, thereby preventing any I/O operations to WCS from another user in a multi-user RTE environment. This ensures that the object code of your microprogram will remain intact but does not prevent another user's program from executing an instruction that enters your object code.

The LoaD command uses WCS I/O Utility routine WLOAD to load into WCS using the LU array in the WCSLT. Object code from two microprograms having the same control memory addresses cannot be developed simultaneously (i.e., no two microprograms can occupy the same control memory locations at the same time).

10-19. CALLING MDE

As previously mentioned, you can prepare a program for the purpose of calling MDE as a subroutine (MDES) or scheduling MDE as a program (MDEP). Remember that MDEP and MDES are separate software modules.

Figure 10-1 and figure 10-2 show respectively, the Assembly language and FORTRAN calling sequences to schedule MDEP and to call MDES. MDES may also be called via a breakpoint in the microprogram object code; if this is done, some additional rules for using MDES must be observed.

Subroutine MDES is functionally identical to MDEP. The main difference is that an MDES EX command returns to the calling program rather than terminating the program. The software saveable registers are set to their values when MDES is called instead of being set to 0 as in MDEP. Neither MDEP nor MDES will clear breakpoints when exited; you must clear any breakpoints when you finish debugging your object code. Figure 10-3 outlines a recommended sequence of interactive debugging operations between you, MDES, and your MDES calling program.

```
Purpose:  To programmatically schedule the program MDEP.

Format:          EXT EXEC
                 .
                 .
          SCHED  JSB EXEC      TRANSFER CONTROL TO RTE
                 DEF RTN       RETURN POINT
                 DEF ICODE     REQUEST CODE
                 DEF MDEP      NAME OF PROGRAM TO SCHEDULE
                 DEF P1        )
                 DEF P2        | OPTIONAL
                 DEF P3        } PARAMETERS
                 DEF P4        )
          RTN    EQU *
                 .
                 .
          ICODE  DEC 23 OR 24  23=SCHEDULE W/WAIT,24=NO WAIT
          MDEP   ASC 3,MDEP    NAME OF PROGRAM
          P1     DEC LU1       OPERATOR CONSOLE LU(DEFAULT=1)
          P2     DEC LU2       WCS LU
          P3     DEC LU3       WCS LU
          P4     DEC LU4       WCS LU

          DIMENSION MDE(3)
          ICODE=23 OR 24
          MDE(1)=2HMD
          MDE(2)=2HEP
          MDE(3)=2H
          CALL EXEC(ICODE,MDE,I1,I2,I3,I4)

          I1 thru I4 are identical to the Assembly language
          schedule request parameters P1 thru P4.
```

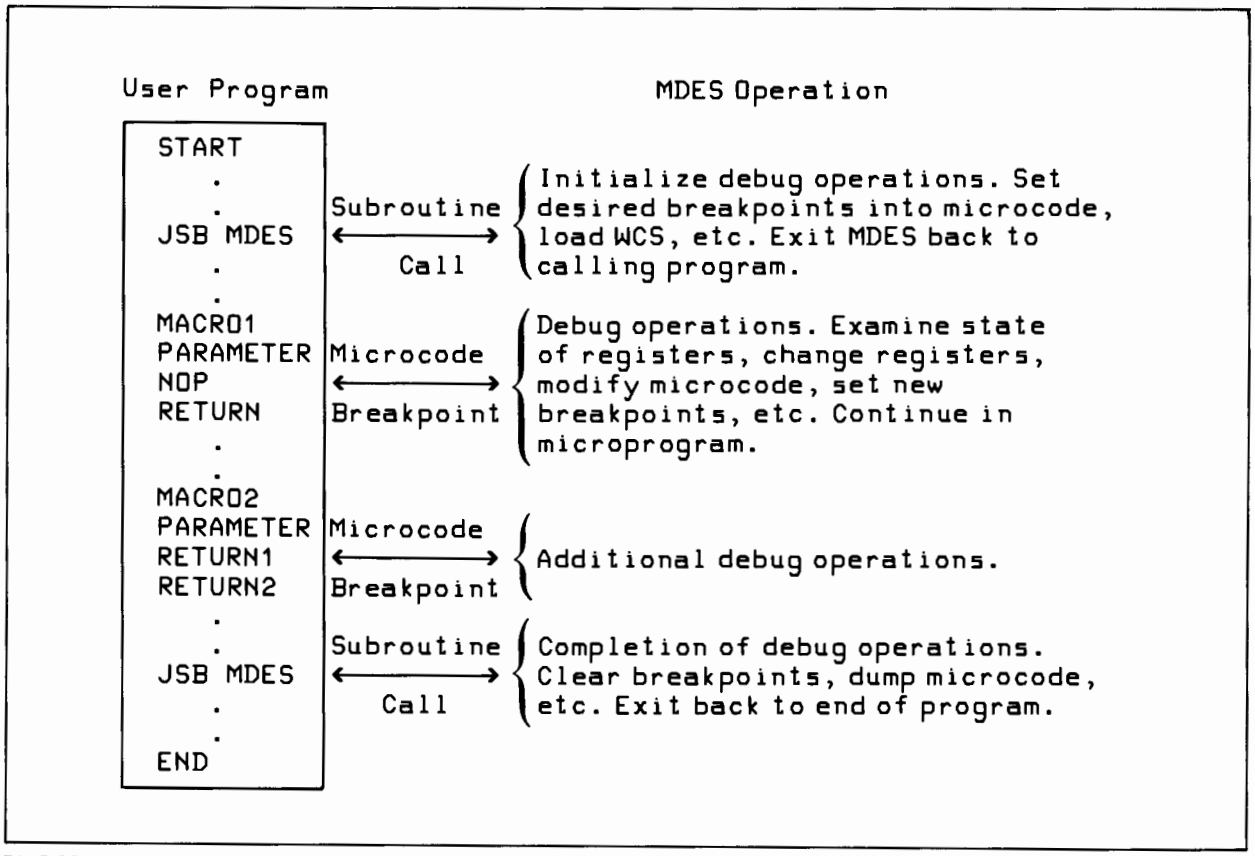
7115-28

Figure 10-1. Scheduling MDE (MDEP)

```
Purpose:  To call the utility subroutine MDES.

Format:          JSB MDES     JUMP SUBROUTINE
                 DEF RTN      RETURN POINT
                 DEF P1       )
                 DEF P2       | OPTIONAL
                 DEF P3       } PARAMETERS
                 DEF P4       )
                 DEF P5
          RTN    EQU *
                  .
                  .
                  .
          P1     DEC LU1      OPERATOR CONSOLE(DEFULT=1)
          P2     DEC LU2      WCS LU
          P3     DEC LU3      WCS LU
          P4     DEC LU4      WCS LU
          P5     BSS 1        ERROR CODE(0=SUCCESSFUL
                              COMPLETION,-1=SUBROUTINE
                              ABORTED)

          CALL MDES(I1,I2,I3,I4,I5)

          I1 thru I5 are identical to P1 thru P5 in the
          Assembly language call.
```

7115-29

Figure 10-2. Calling MDE (MDES)



7115-30

Figure 10-3. Interactive Debugging Operations

Section 11
WRITABLE CONTROL STORE (WCS) SUPPORT SOFTWARE

WRITABLE CONTROL STORE (WCS) SUPPORT SOFTWARE

SECTION 11

Section 8 describes a method used to prepare a microprogram and then store it in a system file. The microprogram source could also have been entered through the system input device. When you prepare a microprogram and enter it into the system, essentially you have just another file of data; even after microassembly, you still have just a file of micro-object code in a disc file. In order to make your microprogram (file) effective (i.e., executable through use of main memory UIG instructions 105xxx octal codes) the microprogram must be placed in control memory. As emphasized previously (in sections 1 and 3), your facility for dynamic control memory (CM) is Writable Control Store (WCS), which is where you want to place your micro-object code.

NOTE

> Although you may of course execute microroutines when they reside in any facility of CM (e.g., FAB and UCS as well as WCS), WCS is essential for microprogram development and dynamic microprogramming. (Dynamic microprogramming is defined as the ability to swap microprograms in and out of WCS as desired.) More information on this is in paragraph 11-2.

This section outlines the hardware and software necessary to transfer your microprogram (from the file you created in the RTE system) into WCS then, modify your microprogram as required for proper execution.

11-1. WCS HARDWARE

Before anything can be done about moving microprograms from main memory to control memory you have to have a WCS board or boards installed in the I/O section of the computer and properly configured for CM and the RTE system. Some details on the WCS boards you can use follow but for complete board configuration and installation information refer to the *HP 13197A Writable Control Store Reference Manual.* You should also refer to section 3 to review the steps necessary to prepare for microprogramming with the RTE system.

You may use the HP 13197A WCS board in the computer for dynamic microprogramming. The HP 13197A WCS has a capacity of 1024 microwords (1K) which is four CM modules. No hardware configuring is necessary to use the 13197A WCS. If one WCS board is used, it is advised (in the WCS manual) that it be installed in SC 10 in the computer. The driver takes care of setting appropriate CM addresses on the board from addresses assigned in your microprogram (the driver is described in paragraph 11-2).

For normal use, a maximum of three WCS boards can be connected with the CM cables supplied. Standard maximum WCS configurations (capacities) are 3K of WCS in the Computer for either an RTE II or RTE IV system.

11-2. WCS SOFTWARE

Manipulating microwords between main memory and WCS via the I/O section is the task of the WCS microprogramming support software. Driver DVR36 and the WCS I/O Utility (library) routine WLOAD comprise this software.

DVR36 drives the WCS boards for data transfers (of micro-object code through the I/O section while conforming to constraints for the RTE system I/O. The driver ensures that no two enabled WCS boards have the same CM addresses assigned. Control requests, write requests (writing microroutines to WCS), and read requests (reading microroutines from WCS) are possible using DVR36. WLOAD coordinates between the system and WCS. WLOAD uses DVR36 to perform its operations and move large quantities of micro-object code to WCS. Also, if so configured, DVR36 utilizes DCPC for transfers.

WCS boards must be initialized (i.e., assigned subchannel base addresses) for the transfer of microprogram object code to the boards. WCS initialization is required whenever the RTE system is booted up. Complete information required to write WCS initialization programs is given in the Driver DVR36 manual. (Section 14 contains an example initialization procedure for the 1K WCS (HP 13197A).) The WCS initialization program can be included in the RTE system during system generation or loaded on-line. (Refer to the RTE operating manual for information on system generation and program loading.)

To transfer microprograms between WCS and a main memory buffer or to make control requests to WCS, you call the driver *directly* with an RTE system EXEC call. To load WCS with microprograms from a file or LU, you use WLOAD. The procedures to use for calling the driver or WLOAD in Assembly language or FORTRAN are detailed in the DVR36 and WLOAD manual (reference section 3 for the manual part number, object software part numbers, and procedures for including the software (loading) in the RTE system.) Complete configuring information is also contained in the driver manual where appropriate RTE system manual references are also made. Section 14 in this manual (examples) provides additional details on using FORTRAN to control WCS operations including initializing, locking, unlocking, enabling, and disabling your WCS boards, and executing your microprogram in the system. Note that, with the HP 13197A WCS board, your subchannels should have different LU's assigned at configuration time.

The Microdebug Editor also uses DVR36 and WLOAD to perform microprogram editing and execution tasks with WCS. All the information you need to operate the driver and utility routine with the Microdebug Editor is included in section 10. All the information required to operate with the WCS microprogramming support software directly in the RTE system is included in the driver manual and you will not have to get involved in operating details unless you so desire.

Section 12
USING pROM GENERATION SUPPORT SOFTWARE AND HARDWARE

USING pROM GENERATION SUPPORT SOFTWARE AND HARDWARE

SECTION 12

This section provides instructions for generating pROM mask tapes by using the pROM Tape Generator program (PTGEN). The mask tapes enable a microprogram to be fused ("burned") into programmable read-only memory (pROM) semiconductor integrated circuits (IC's.). Before generating pROM tapes, the microprogram should be completely debugged and its source should be corrected and microassembled again to provide the object code required by PTGEN. PTGEN can provide a variety of pROM mask formats, including those of a variety of pROM vendors. Note that the program must be in the system prior to use and see section 3 for preparatory information.

12-1. USING THE pROM TAPE GENERATOR

Run program PTGEN by entering the following command:

RU,PTGEN,*userin,list,objectin,ptapein,ptapeout*

The command parameters are defined as follows:

userin is the logical unit (LU) that you will use to respond to PTGEN queries. The default is LU 1.

list is the LU on which all PTGEN queries and error messages are written. The default is LU 1.

objectin is the LU from which the microassembler object code is read. If this is LU 2, the disc file name will be requested. The default is LU 5. Note that the object code must be produced by the microassembler, not by the Microdebug Editor.

ptapein is the LU from which the punched pROM mask tapes are read for verification. This LU must accept the output of the *ptapeout* LU. The default is LU 5.

ptapeout is the LU on which the pROM mask tapes are punched. This should be a paper tape punch to be accepted by most pROM vendors. The default is LU 4.

pROM mask tape generation is divided into three phases: Initialize, Punch, and Verify. A temporary disc file (named ??PTMP) will be created during the Initialize Phase if the *objectin* parameter specifies a logical device other than the disc. This temporary file is purged before PTGEN terminates. Each phase includes a series of queries to which you must respond. In most cases, you can default a response by entering a "null line"; i.e., a blank (space) character. Also, in making responses, you need only enter the first letter of the following words: YES, NO, COMMENTS, REPLACE, OCTAL, DECIMAL, and ALL. PTGEN error messages are described at the end of this section.

Each PTGEN query shown in this section is preceded by a reference number; this number is not part of the actual query.

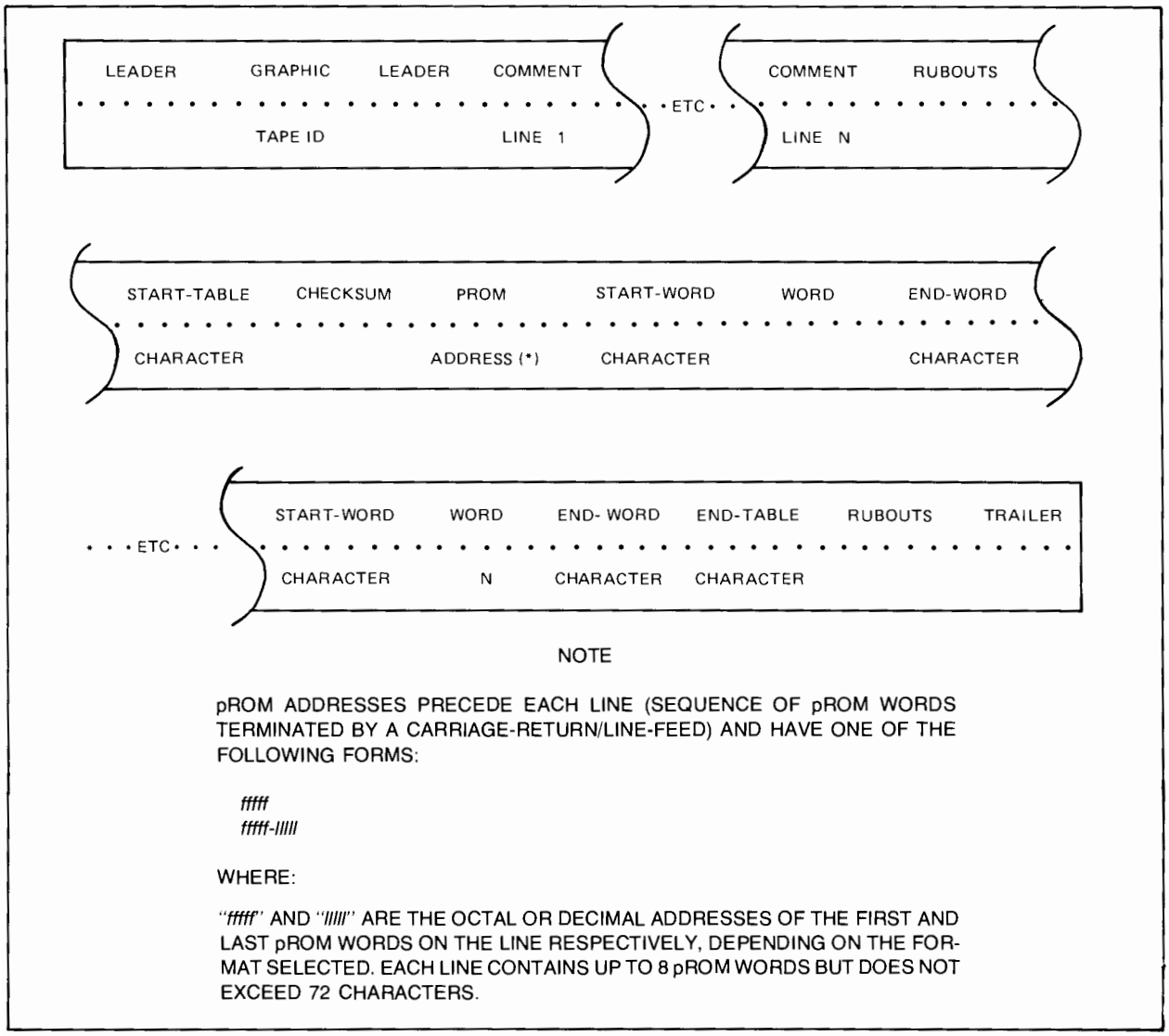
12-2. INITIALIZE PHASE

During the Initialize Phase, you must set up the desired format of the pROM mask tapes. (Figure 12-1 shows the general format for the mask tapes.) The Initialize Phase queries are listed and described below.

1.0 NUMBER OF WORDS PER PROM?
Respond with the number of words (locations) to be contained in each pROM.

1.1 NUMBER OF BITS PER PROM WORD?
Respond with the number of bits per microinstruction contained in each pROM. This should be a divisor of 24, the number of bits per microinstruction. The acceptable values are 1, 2, 3, 4, 6, 8, 12, and 24.

1.2 UNUSED-LOCATION LEVEL (H/L)?
Respond with H or L to indicate the level used to initialize unused portions of the pROM (due to the use of the ORG and ALGN psuedo-microinstructions). If you respond with a null line, the default is H. If H is specified, all ones are generated; otherwise, the buffer is initialized to zeros.



7115-32

Figure 12-1. General Tape Format

1.3 PUNCH TAPE ID (Y/N)?
Respond with Y or N to punch or omit the mask tape ID (identification). The format of the punched tape ID is :

aaaaa-aaaaa (*bb-bb*)

where:

aaaaa-aaaaa represents the low and high control memory address and *bb-bb* represents the left and right bit number represented in the truth table. Note that "*a*" is octal and "*b*" is decimal. The graphic presentation of the tape ID is such that when you look at the punched tape, the hole patterns form recognizable characters.

1.4 DEFAULT VENDOR FORMAT (NAME)?
If desired, respond with the name of a pROM vendor and thereby default to that vendor's format, bypassing much of the Initialize Phase. The vendors recognized by PTGEN are: HP, INTEL, MMI, and SIGNETICS. (Refer to table 12-1 for vendor formats.) If you specify one of these vendors, the dialogue continues at query 3.0; if you enter a null line, the dialogue continues at 2.0.

2.0 NUMBER OF COMMENT LINES?
Enter the number of comment lines. These usually identify the user and the contents of the tape and are punched preceding the truth table.

2.1 PUNCH RUBOUTS (Y/N)?
If you enter Y, a series of rubout characters are punched on the mask tapes before and after the truth table; if N, none.

2.2 PUNCH CHECKSUM (Y/N)?
Enter Y or N to punch or omit a checksum. The checksum is a numeric string of four decimal characters that represents the number of high-level characters in the truth table. If startand end-table characters delimit the table, the checksum is punched immediately after the start-table character.

2.3 START-TABLE,END-TABLE CHARACTERS?
If startand end-table characters are required to delimit the truth table, enter the two characters, separated by a comma (,); enter a null line if the characters are not required.

2.4 START-WORD,END-WORD CHARACTERS?
If startand end-word characters are required to delimit each word in the truth table, enter the two characters, separated by a comma; enter a null line if the characters are not required.

2.5 HIGH-LEVEL,LOW-LEVEL CHARACTERS?
Enter the required highand low-level characters, separated by a comma. If you enter a null line, the default characters are H and L for the high and low levels.

2.6 PROM ADDRESS FORMAT (O/D,1/2)?
If desired, the pROM addresses (not the control memory address) can precede each "line" punched from the truth table. (A "line" refers to a sequence of pROM words, terminated by a carriage return and line feed.) The response consists of two parts, separated by a comma. The first part of the response is either of the letters "O" or "D" and indicates whether the addresses are to be punched in octal or decimal form. The second part of the response indicates whether one or two addresses are to be punched for the pROM words in a line; a "1" provides only the first address; a "2" provides both the first and last addresses. A null response suppresses the punching of any pROM addresses.

Table 12-1. Default Formats by Vendor

ITEM	HP	INTEL	MMI/SIGNETICS
Number of comment lines	3	5	9
Rubouts punched	No	Yes	Yes
Checksum punched	Yes	No	No
Start/end-table characters	—	—	S,E
Start/end-word characters	—	B,F	B,F
High/low-level characters	H,L	P,N	H,L
pROM address format	D,2	0,1	0,1

Note: The formats generated are as follows:

Intel	BPNF format as defined in Intel's 1976 data catalog.
MMI (Monolithic Memories, Inc.)	TWX ASCII BHLF format as defined in MMI's 1973 through 1976 pROM device data sheets.
Signetics	Accepts both the Intel and MMI formats given above.
HP	This format is recognized by the HP pROM Writer (part no. 12909-16005), which is supported only in DOS and BCS environments.

Parts that HP has used with PTGEN tapes are:

pROM PART	21MX	21MX E-SERIES
4K	Signetics 82S115	Signetics 82S141
1K	MMI 6301	MMI 6301
1K (Using HP pROM Writer)	Harris 1024	Harris 1024

The following queries depend on the type of logical unit specified by the *objectin* parameter in the RU,PTGEN command; only one of the queries will be asked.

3.0 OBJECT CODE FILE NAME?
This query is asked if you specified LU 2 as the *objectin* parameter. Respond by entering the name of the disc file in which the microassembler was directed to store the microprogram object code. The file name has the following format:

filename[:*security*[:*crlabel*]]

(Refer to the *Batch and Spool Monitor Manual* for details.) The documentation map in the preface shows the part no.

3.1 TEMPORARY FILE NAME?
If you did not specify LU 2 as the *objectin* parameter, PTGEN must store the object code in a temporary disc file during the Punch Phase for use during the Verify Phase. PTGEN automatically attempts to create this file (using ??PTMP as the file name); the query is given only if the attempt fails. You may respond to the query by entering a file name, optionally followed by the word "REPLACE", as follows:

filename[:*security*[:*crlabel*]][,REPLACE]

If a name conflict arises and REPLACE is specified, the existing file is purged and a new file is created. If a name or access conflict arises and REPLACE is not specified or the existing file cannot be purged, the query is repeated. You may respond with a null line to default the query. In that case, you will have to re-input the source for the Verify Phase.

12-3. PUNCH PHASE

After the Initialize Phase, the pROM mask tapes are punched. One mask tape is punched for each pROM I.C. containing w locations of b bits each, as specified during the Initialize Phase. The number of mask tapes punched for w locations of object code equals $24/b$. The truth table for the most significant bits is punched first. A complete truth table is always punched, using the unused-location character to represent unused portions of the pROM.

The pROM mask tapes are punched according to the specifications you give to PTGEN during the Initialize Phase. Carriage-return and line-feed sequences are appropriately punched in the truth table to aid visual verification of mask tapes when listing them off-line. Before punching each mask tape, PTGEN asks if you want to modify any comment lines; if you do not, it uses the comments from the previous mask tape.

The queries asked during the Punch Phase are listed and described in the following paragraphs.

4.0 NEXT PUNCH ADDRESS, BIT-NUMBER?
Respond by entering a null line to skip or terminate the Punch Phase and go to the Verify Phase. Other acceptable responses are:

aaaaa,bb
aaaaa,ALL
ALL

ALL or *aaaaa*, ALL means that all object code or all bit fields within the specified address range is to be punched. The *aaaaa,bb* means that object code for a specific pROM is to be punched. The "*a*" is an octal address and the "*b*" is a decimal (or octal, if followed by B) bit number in the range to be punched. These are normalized to the lowest address and the left-most bit number in the truth table. For example, if the address specified for a 4x256 pROM is 2100,20 the truth table punched will include the addresses 2000 through 2377 and bits 23 through 20.

4.1 REPLACE COMMENTS FOR TAPE *aaaaa,bb*?
The *aaaaa,bb* is similar to the specification described for 4.0, above. Respond with Y to modify comments; with N to leave the comment lines unchanged from the previous mask tape. Comments are initialized to one blank character each.

4.2 COMMENT LINE *n*:
Respond with a null line to leave the comment unchanged from the previous mask tape. Otherwise, enter the new comment line. Comment lines may be up to 72 characters long. This query is repeated for each comment line, where *n* is the comment line number.

After the pROM tapes are punched, query 4.0 is repeated (see above).

12-4. VERIFY PHASE

After all of the pROM mask tapes have been punched, they may be verified by reading them via the *ptapein* device. When loading a punched pROM tape, it must be positioned in the reader so that the graphic ID (if there is one) will not be read. Also, the tape must be positioned before any comment lines, regardless of whether or not you intend to verify comments. The queries and messages of the Verify Phase are listed and described in the following paragraphs.

5.0 NEXT VERIFY ADDRESS,BIT-NUMBER?
Respond with a null line to terminate the Verify Phase. Other acceptable responses are:

aaaaa,bb
aaaaa,ALL[,COMMENTS]
ALL[,COMMENTS]

ALL or *aaaaa*,ALL means that all object code or all bit fields within the specified address range is to be verified. Also, if either of these two responses is given, then the mask tapes must be loaded in the same order in which they were punched. The *aaaaa,bb* means that object code for a specific pROM is to be verified. The "*a*" is an octal address and the "*b*" is a decimal (or octal, if followed by B) bit number in the range to be verified. These are normalized to the lowest address and the left-most bit number in the truth table. (Refer to 4.0 in the Punch Phase.) If COMMENTS is specified, the comment lines are verified.

5.1 RELOAD OBJECT TAPE AND *GO
This message is omitted if the object code can be read from a disc file. If this message is issued, PTGEN suspends itself to allow you to load the object code tape in the *objectin* device. After you load the object tape, enter the RTE GO command to resume the verification operation. Note that if the object tape is incorrectly positioned in the tape reader, PTGEN is aborted after the GO command is given.

5.2 LOAD PROM TAPE *aaaaa,bb* AND *GO
After this message is issued, PTGEN suspends itself to allow you to load a pROM mask tape in the *ptapein* device. Load the mask tape and enter the GO command. If the verify operation is successful and comments are not to be verified, the next pROM tape is verified or PTGEN resumes at query 5.0.

If a verify error is detected, the error is reported and the pROM mask tape is repunched. You may change the comment lines on the new pROM tape to distinguish it from the erroneous mask tape.

If comments are to be verified (COMMENTS specified when specifying address range), the dialogue continues with the following:

5.3 COMMENTS FOR TAPE *aaaaa,bb*
This line is followed by a display of all of the comment lines.

5.4 ERRORS IN COMMENTS (Y/N)?
Respond with N or a null line if the comments are valid. The Y response is treated as a verify error.

5.5 REPLACE COMMENTS FOR TAPE *aaaaa,bb*?
Respond with Y to modify comments; respond with N or a null line to leave comments unchanged.

5.6 COMMENT LINE *n*:
Respond with a null line to leave the comment unchanged or enter a new comment line. The comment line may include up to 72 characters. This query is repeated for each comment line; n is the comment line number.

After the new mask tape has been punched, PTGEN resumes at query 5.0 (or 5.1 if you are verifying all of the mask tapes). If ALL or aaaaa,ALL was specified, repunched mask tapes should not be verified until after all of the tapes in the original range have been processed.

12-5. pROM TAPE GENERATOR ERROR MESSAGES

The error messages that might be issued by the pROM tape generator (PTGEN) are as follows:

1 INVALID FILE SPECIFICATION OR EXTRA INPUT.
The file designation was not in the proper format or REPLACE was misspelled.

2 INVALID VENDOR NAME.
The vendor name was misspelled or is not among those recognized by PTGEN. In the latter case, enter a null line and proceed to specify the details of the pROM tape format.

3 NO OBJECT CODE.
An END record was encountered as the first record, or a null line was entered in response to query 3.0

4 INVALID RESPONSE OR EXTRA INPUT.
The response was not in the proper format or was not a proper response (e.g., not Y or N).

5 INVALID NUMBER OR EXTRA INPUT.
The response was an improperly formed number or not in the required range.

6 I/O ERROR READING OBJECT CODE.
Self explanatory.

7 CANNOT CREATE TEMPORARY FILE.
This message is followed by a File Manager error code.

8 CANNOT PURGE TEMPORARY FILE.
This message is followed by a File Manager error code.

9 CANNOT OPEN OBJECT CODE FILE.
This message is followed by a File Manager error code.

10 INVALID OBJECT CODE RECORD.
This could be due to a checksum error, or the record might not have been created by the microassembler.

11 INVALID ADDRESS SPECIFICATION OF EXTRA INPUT.
The response was not in the proper format or COMMENTS was misspelled.

12 ADDRESS NOT FOUND IN OBJECT CODE.
The pROM address range specified is not included in the object code. This might be due to typing the wrong address.

13 I/O ERROR READING RESPONSE.
A transmission error occurred on the input device; PTGEN aborts.

14 INSUFFICIENT MEMORY.
There is insufficient memory for the pROM or comment buffer. In the case of the comment buffer, if some space can be allocated it is indicated by the following message:

nnnn LINES AVAILABLE

15 VERIFY ERROR — pROM TAPE REPUNCHED.
An error occurred in verifying the punched pROM mask tape. This might be due to an affirmative response to query 5.4, an I/O error, or a compare error. In these cases, the error message is followed by one of the following messages, respectively:

TAPE *aaaaa,bb*
TAPE *aaaaa,bb* LINE nnnn
TAPE *aaaaa,bb* LINE *nnnn* COLUMN *cc*

If *nnnn* equals the number of comment lines, an I/O error occurred while reading one of the comments.

12-6. pROM HARDWARE

When the mask tapes have been generated and pROM's fused you may mount them on one of the boards available for installation in the computer. The HP 13304A Firmware Accessory Board can hold 3.5K microwords of control memory. Details on mounting pROM's, configuring, and installing this accessory are contained in the *HP 13304A Firmware Accessory Board Installation and Service Manual.* The FAB board is installed in the computer under the CPU board. The 2K microword capacity HP 13047A User Control Store board may have pROM's mounted and be installed in the I/O section of the computer. Details for pROM mounting and installation are contained in the *HP 13047A User Control Store Kit Installation and Service Manual*, part no. 13047-90001.

Section 13
USING SPECIAL FACILITIES OF THE COMPUTER

USING SPECIAL FACILITIES OF THE COMPUTER

SECTION 13

There are two functions of the HP 1000 E-Series and F-Series Computers that can be considered as special facilities. These include the block I/O data transfer feature and the Microprogrammable Processor Port (MPP), also available for data transfers. Either of these facilities is controlled by a microprogram written by you, stored in control memory, and called into execution with a UIG instruction in the manner described in preceding sections of this manual. In F-Series Computers the MPP is used to interface the Hardware Floating Point processor (FPP) with the CPU. Therefore, the MPP is not available for user designed hardware on F-Series Computers.

The block I/O facility is, in essence, a microprogramming technique for executing high-speed data transfers through the I/O section. It is made possible because of special signal lines on the I/O backplane. Although the I/O section is used, the process is not a standard I/O transfer operation. Paragraph 13-1 explains the block I/O data transfer facility.

The MPP may be used for interfacing special external hardware to the HP 21MX E-Series Computer (e.g., computer-to-computer linking) under direct microprogram control. Very high data-transfer rates are possible using the MPP which is, in essence, another microprogramming technique that controls special signal lines. These signal lines are on a specifically designated connector which is not part of the I/O section. Paragraph 13-5 explains the MPP facility.

The information on block I/O and the MPP in this section relates specifically to the microprogramming techniques involved in controlling these facilities. Example microprograms are provided simply to illustrate the techniques involved. Your actual application design should be based on these examples and the information contained in the other applicable sections of this manual. WCS and its microprogramming support software can be used to control microprogram placement in control memory in the same manner as any other microprogram (refer to section 11). A summary of typical transfer rates obtainable appears under paragraph 13-8.

Either of these special facilities will require special interfacing hardware that will be controlled by the applicable microprogram. Information that you will need for the hardware design is contained in the *HP 21MX M-Series and E-Series Computers I/O Interfacing Guide,* part no. 02109-90006. The *I/O Interfacing Guide* also contains details you will need on the specific signals (pin numbers, etc.,) controlled by the micro-orders shown in the microprograms in this section.

13-1. BLOCK I/O DATA TRANSFERS

Block I/O data transfers into or out of main memory through the I/O section are performed by using the IOI and IOO S-bus and Store field micro-orders in microprograms *without* the IOG Special field micro-order in any of the four previous microinstructions. When used in the manner shown in the example microprograms (paragraphs 13-2 through 13-4), these two micro-orders cause backplane signals $\overline{\text{BIOI}}$ and $\overline{\text{BIOO}}$, respectively, to be generated which may be utilized by specially designed hardware for non-standard I/O data transfers. A strobe signal ($\overline{\text{BIOS}}$) is generated at interval P4 (35 nanoseconds) to be used by the hardware/microprogram combination to obtain the high data-transfer rates. If IOG is used in the microprogram to synchronize the Control Processor and I/O section to T2 for "standard" I/O operations, the above-mentioned signals are not generated. Table 4-1 explains the normal use of the IOG, IOI, and IOO micro-orders and the other micro-orders shown in the following example microprograms. (Specifically, IRCM and SKPF are applicable.)

Transfers for block I/O are made on a full 16-bit word basis with up to 32K words being transferred (depending upon available memory). The main memory calling sequence for each of the example microprograms is shown in the microprogram comments. The direction of transfer (in or out) is designated by whether the IOI (S-bus field, "input") or IOO (Store field, "output") micro-order is used and this depends upon the microprogram called. Input microprograms are described in paragraphs 13-2 and 13-3. An output microprogram is described in paragraph 13-4. When using these microprograms, as well as any microprogram, it is the programmer's responsibility to be aware of the total system and times taken for bursts, word counts, etc. Interrupts should not be held off for so long that data is lost.

The *I/O Interfacing Guide* provides some suggestions on variations of the transfer techniques shown and guidelines on hardware data buffering. Also see the *I/O Interfacing Guide* for a comparison of block I/O and DCPC transfer techniques.

13-2. BLOCK I/O BYTE PACKING BURST INPUT MICROPROGRAM

Operation of the block I/O microprogram shown in EXAMPLE 1 is explained by the comments included in the listing. The microprogram performs its own STC, as shown in lines BURSTIN through REALSC, for several reasons. (Lines, as mentioned here, refer to labels in the microprogram examples that follow.) First, having the RTE operating system execute a STC at the Assembler level incurs

considerable operating system overhead. Second, having the user program execute a STC at the Assembler level requires turning off Memory Protect. If the microprogram detects a DMS or Memory Protect violation, it is very complex and time-consuming to correctly indicate these conditions to the operating system.

The data transfer takes place with the interrupt system on the Memory Protect enabled, so that DMS and Memory Protect interrupts, as well as any other emergency interrupts, are detectable.

FAKESC and REALSC work together to allow execution of a STC with Memory Protect enabled. Refer to the coding techniques discussion in section 7 (performing microprogrammed I/O with Memory Protect and interrupts on), for a complete explanation.

The IOFF micro-order in line SETPM prevents the HOI conditional tests in lines WAIT1 and WAIT2 from detecting I/O interrupts. I/O interrupts so held off remain pending (i.e., are not lost) and may be serviced at the termination of the microprogram. To operate correctly as block I/O micro-orders, the SKPF RJS tests following lines SKPF1 and SKPF2; and, the IOI's in lines BURST1 and BURST2, require that an IOG *not* be executed in any of the three preceeding microinstructions. However, this does require a hardware modification (see the *I/O Interfacing Guide*.)

EXAMPLE 1: BLOCK I/O BYTE PACKING BURST INPUT MICROPROGRAM

```
MICMXE,L                                          SPECIFIES 1000 E-SERIES OR F-SERIES.
$CODE=BI001                                       SAVE MICRO-OBJECT ON DISC.
            ORG                          34000B   105600 MAPS TO 34000

*
* BLOCK I/O BYTE PACKING BURST INPUT MICROPROGRAM
*
* THIS MICROPROGRAM:
* 1. INPUTS DATA IN A "BURST" MANNER.
* 2. PACKS THE INPUT DATA AND STORES IT IN MAIN MEMORY.
* 3. IS INTERRUPTIBLE BY EMERGENCY INTERRUPTS (I.E., PARITY ERROR, DMS, MEMORY PROTECT);
*    POWER FAIL AND I/O INTERRUPTS WILL NOT BE SERVICED DURING THE BURST DATA TRANSFER.
* 4. ASSUMES THAT THE I/O CARD PASSING DATA TO THE CPU INDICATES PRESENCE OF A SINGLE
*    BYTE BY SETTING THE I/O CARD'S FLAG AND THAT IN THE EVENT OF AN EMERGENCY
*     INTERRUPT INCOMING DATA IS NOT LOST.
* 5. REQUIRES THE FOLLOWING CALLING SEQUENCE;
*    LDA COUNT A = NEGATIVE BYTE COUNT
*    LDB BUFAD B = BUFFER ADDRESS
*    LDX SC     X = SELECT CODE
*    CLE         INITIAL ENTRY TO MICROCODE
*    OCT 105600 MICROPROGRAM OP CODE,
* 6. HAS A MAXIMUM TRANSFER RATE OF ABOUT 500 KB/S (KILOBYTES/SECOND) IN A NON-DCPC
*    ENVIRONMENT. IN A TYPICAL DCPC ENVIRONMENT, BURST RATES UP TO 250 KB/S ARE
*    ATTAINABLE.
*
            JMP                          BURSTIN    SAVE ENTRY POINTS
            ALGN
BURSTIN     JMP    CNDX    E             ODDBYTE    RETURN FROM INTERRUPT
*                                                   AFTER ODD NUMBER BYTES
            JSB                          STCNTRL    EXECUTE STC,C
*
```

EXAMPLE 1: BLOCK I/O BYTE PACKING BURST INPUT MICROPROGRAM (Continued)

```
                        DEC    S3     P          SAVE P.
SETPM           IOFF    INC    PNM    B          M = BUFFER ADDRESS,
*                                                P = NEXT BUFFER ADDRESS,
*                                                HOLD OFF I/O INTERRUPTS.
WAIT1     JMP   CNDX    HOI           INT1       EMERGENCY INTERRUPTS?
SKPF1                           PASS
          JMP   CNDX    SKPF   RJS    WAIT1      NO, WAIT FOR DATA READY.
*
BURST1          L4             S4     IOI        S4(11-4) = BYTE 1.
                L4             S4     S4         S4(15-8) = BYTE 1.
                        INC    A      A          UPDATE BYTE COUNT
END1      JMP   CNDX    ALZ           WRTE1      COUNT = 0? YES, WRTE BYTE.
*
WAIT2     JMP   CNDX    HOI           INT2       EMERGENCY INTERRUPTS?
SKPF2                   PASS                     ALLOW STATUS UPDATE
          JMP   CNDX    SKPF   RJS    WAIT2      NO, WAIT FOR DATA READY.
*
BURST2                         L      IOI        L(7-0) = BYTE 2.
                        IOR    S4     S4         S4(15-8, 7-0) = BYTES 1,2.
WRTE12    WRTE  MPCK           TAB    S4         WRTE PACKED DATA, DO MPCK.
                        INC    PNM    P          UPDATE BUFFER ADDRESS.
                        INC    A      A          UPDATE BYTE COUNT.
END2      JMP   CNDX    ALZ    RJS    WAIT1      COUNT = 0? NO, CONTINUE.
          JMP                          DONE       YES, EXIT.
*
WRTE1     WRTE  MPCK           TAB    S4         WRTE BYTE 1, DO MPCK.
                        INC    P      P          UPDATE BUFFER ADDRESS.
*
DONE            ION            B      P          B = LAST BUFFER ADR. + 1.
          READ  RTN     INC    PNM    S3         FIX P, START FETCH FOR
*                                                NEXT INSTRUCTION IN MAIN
ODDBYTE   READ          INC    PNM    B          GET PARTIALLY PACKED WORD
          IMM           LOW    IRCM   101B       FORM AND EXECUTE
                ASG            S4     TAB         CLE INSTRUCTION
          JSB                          STCNTRL    EXECUTE STC,C
          JMP                          WAIT2      GET SECOND BYTE
*
INT1      IMM           LOW    IRCM   105B       CLEAR EXTEND REG
          JMP                          INTRPT
INT2      IMM           LOW    IRCM   305B       SET EXTEND REG
INTRPT          ASG     DEC    B      B          EXECUTE CLE OR
*                                                CCE AND SAVE
*                                                BUFFER ADDRESS
                ION     PASS   P      S3         FIX P, EXIT TO
          JMP                          6B         TO HORI ROUTINE
*
*
STCNTRL   IMM   L4      CMLO   L      303B       L=STC 0,C
          IMM           CMLO   S4     376B       S4=1
                        IOR    S4     S4         S4=STC 1,C
FAKESC                  PASS   IRCM   S4         IRCM=STC 1,C
                        IOR    S4     X          S4=STC SC,C
REALSC                  PASS   CNTR   S4         IRCM=STC SC,C
          RTN   IOG
                        END
```

13-3. BLOCK I/O ADDRESS/DATA BURST INPUT MICROPROGRAM

Operation of a block I/O microprogram to input an address and data is shown in EXAMPLE 2. Explanation of the microprogram is provided in the comments included in the listing. As explained for the previous microprogram, the microprogram performs its own STC, as shown in lines BURSTIN through REALSC, for the reasons explained in paragraph 13-2. Lines FAKESC and REALSC work together to allow execution of a STC with Memory Protect enabled. Refer to the coding techniques discussion in section 7 (performing microprogrammed I/O with Memory Protect and interrupts on) for a complete explanation.

EXAMPLE 2: BLOCK I/O ADDRESS/DATA BURST INPUT MICROPROGRAM

```
MICMXE,L                                                SPECIFY 21MX E-SERIES.
$CODE=BI002                                             SAVE MICRO-OBJECT ON DISC.
            ORG                             34000B      105600 MAPS TO 34000B

*
* BLOCK I/O ADDRESS/DATA BURST INPUT MICROPROGRAM
*
* THIS MICROPROGRAM:
* 1. INPUTS, IN A "BURST" MANNER, AN ADDRESS FOLLOWED BY THE DATA TO BE WRITTEN INTO THAT
*    ADDRESS IN MAIN MEMORY.
* 2. IS INTERRUPTIBLE BY EMERGENCY INTERRUPTS (I.E., PARITY ERROR, DMS, MEMORY PROTECT);
*    POWER FAIL AND I/O INTERRUPTS WILL NOT BE SERVICED DURING THE BURST TRANSFER.
* 3. ASSUMES THAT THE I/O CARD PASSING AN ADDRESS OR DATA TO THE CPU WILL INDICATE
*    PRESENCE OF A SINGLE ADDRESS OR DATA ITEM BY SETTING THE I/O CARD'S FLAG
*    AND THAT DATA IS NOT LOST IN THE EVENT OF AN EMERGENCY INTERRUPT.
* 4. REQUIRES THE FOLLOWING CALLING SEQUENCE;
*    LDA COUNT      A = POSITIVE WORD COUNT
*    LDB SC         B = SELECT CODE
*    CLE             INITIAL ENTRY TO MICROCODE
*    OCT 105600 MICROPROGRAM OP CODE.
* 5. HAS A MAXIMUM TRANSFER RATE OF ABOUT 500 KP/S (KILO-PAIRS/SECOND, ONE PAIR = 1
*    ADDRESS AND 1 DATA) IN A NON-DCPC ENVIRONMENT. IN A TYPICAL DCPC ENVIRONMENT RATES
*    UP TO 250 KP/S ARE ATTAINABLE.
*
            JMP                             BRSTIN      SAVE ENTRY POINTS.
            ALGN
BRSTIN                      DEC     S3      P           STORE P
STCNTRL     IMM     L4      CMLO    L       303B        L=STC 0,C
            IMM             CMLO    S4      376B        S4=1
                            IOR     S4      S4          S4=STC 1,C
FAKESC                      PASS    IRCM    S4          IRCM=STC 1,C
                            IOR     S4      B           S4=STC SC,C
REALSC                      PASS    CNTR    S4          IRCM=STC SC,C
                    IOG
            JMP     CNDX    E               BRSTDTA
BRSTADR     JMP     CNDX    HOI             INTADR      EMERGENCY INTERRUPTS?
                            PASS                        INTERFACE FLAG SET?
            JMP     CNDX    SKPF    RJS     BRSTADR     NO, GO TO BRSTADR
                                    M       IOI         M = BUFFER ADDRESS.

*
BRSTDTA     JMP     CNDX    HOI             INTDTA      EMERGENCY INTERRUPTS?
                            PASS                        INTERFACE FLAG SET?
            JMP     CNDX    SKPF    RJS     BRSTDTA     NO, GO TO BRSTDTA
                                    S4      IOI         S4 = DATA
BRSTEND     WRTE    MPCK            TAB     S4          WRITE DATA INTO MEMORY.
                            DEC     A       A           UPDATE PAIR COUNT.
DONE        JMP     CNDX    ALZ     RJS     BRSTADR     COUNT = 0? NO, CONTINUE.
            READ    RTN     INC     PNM     S3          =0, FIX P, START FETCH.
INTADR      IMM             LOW     IRCM    101B        CLEAR EXTEND REGISTER
            JMP                             INTRPT
INTDTA      IMM             LOW     IRCM    301B        SET EXTEND REGISTER
INTRPT              ASG     PASS    P       S3          EXECUTE CLE OR CCE AND FIX P
            JMP                             6           EXIT TO HALT OR INTERRUPT
            END                                          MICROROUTINE
```

13-4. BLOCK I/O WORD BURST OUTPUT MICROPROGRAM

Operation of the block I/O microprogram shown in EXAMPLE 3 is explained by the comments included in the listing. Similar considerations for interrupts and IOG as explained for EXAMPLES 1 and 2 also apply for this microprogram.

EXAMPLE 3: BLOCK I/O WORD BURST OUTPUT MICROPROGRAM

```
MICMXE,L                                                    SPECIFIES E-SERIES OR F-SERIES
$CODE=BI003                                                 SAVE MICRO-OBJECT ON DISC.
            ORG                             34000B          105600 MAPS TO 34000.
*
*BLOCK I/O BURST OUTPUT MICROPROGRAM
*
* THIS MICROPROGRAM:
* 1. OUTPUTS DATA IN A "BURST" MANNER.
* 2. IS INTERRUPTIBLE BY EMERGENCY INTERRUPTS (I.E., PARITY ERROR, DMS, MEMORY PROTECT);
*    POWER FAIL AND I/O INTERRUPTS WILL NOT BE SERVICED DURING THE BURST DATA TRANSFER.
* 3. ASSUMES THAT THE I/O CARD RECEIVING DATA FROM THE CPU IS READY TO RECEIVE DATA AND
*    CONTAINS A DATA BUFFER LARGE ENOUGH TO HOLD THE ENTIRE BURST.
* 4. REQUIRES THE FOLLOWING CALLING SEQUENCE;
*    LDA COUNT     A = POSITIVE WORD COUNT
*    LDB BUFAD     B = BUFFER ADDRESS
*    LDX SC        X = SELECT CODE
*    OCT 105600 MICROPROGRAM OP CODE.
* 5. HAS A MAXIMUM TRANSFER RATE OF ABOUT 1000 KW/S (KILO-WORDS/SECOND) IN A NON-DCPC
*    ENVIRONMENT. IN A TYPICAL DCPC ENVIRONMENT, RATES UP TO 400 KW/S ARE ATTAINABLE.
*
            JMP                             BURSTOUT        SAVE ENTRY POINTS.
            ALGN
BURSTOUT                    DEC     S3      P               SAVE NEXT INSTRUCTION ADDRESS
            READ            INC     PNM     B               READ DATA, INITIALIZE P,M
SETIR               IOFF            IRCM    X               IR(5-0) = SC, IOFF HOLDS
*                                                             OFF I/O INTERRUPTS.
BURST1                              IOO     TAB             BURST DATA OUT OF MEMORY.
                            INC     PNM      P              UPDATE P,M
            JMP     CNDX    HOI              INTRPT         EMERGENCY INTERRUPTS?
            READ            DEC     A        A              READ NEXT DATA, UPDATE COUNT.
END1        JMP     CNDX    ALZ     RJS      BURST1         COUNT = 0? NO, CONTINUE.
*
DONE                ION             B       P               B = LAST BUFFER ADDRESS + 1
            READ    RTN     INC     PNM     S3              START FETCH FOR NEXT INSTRUCTION
*                                                             IN MAIN MEMORY.
INTRPT              ION     DEC     B       P               B = NEXT BUFFER ADDRESS
                                    P       S3              FIX P, EXIT TO HALT-OR-
            JMP                             6               INTERRUPT MICROROUTINE
            END
```

13-5. MICROPROGRAMMABLE PROCESSOR PORT

The Microprogrammable Processor Port (MPP) permits external hardware to be directly connected to the E-Series Computer and interfaced under direct microprogrammed control. Applications possible with the MPP include computer-to-computer communications, adaptation of specialized performance accelerating hardware, a fast or special I/O channel (similar in function to the DCPC), etc. The MPP special facility is comprised of a hardware/microprogram combination. The hardware interface is summarized below. A microprogram which may be used as a basis for your MPP design is discussed in paragraph 13-8. Note that the MPP facility has nothing to do with the I/O section. The Microprogrammable Processor Port is used in the F-Series Computer to interconnect the Hardware Floating Point Processor to the CPU, to enable directly microprogrammed arithmetic floating point operations and chained calculations.

13-6. HARDWARE INTERFACE

As illustrated in figure 2-1 and in appendix H, the MPP physical interface consists of a connector on the computer. This connector is located behind the Operator Panel (Refer to the *I/O Interfacing Guide* for the location and designation.) The MPP signal lines are present at this connector and these signals are ultimately under microprogram control. Table 13-1 summarizes some of the MPP physical interface. The use of every one of these signals is ultimately to be determined by the designer. Where use is mentioned in the table it is only a suggestion. Micro-orders mentioned are defined in table 4-1 in this manual. The actual design and use of the MPP must be determined by you (the user) and all information in this section should be interpreted as guidelines for design. Details on signal levels, connector pin number assignments, and other interface hardware design information for MPP use will be found in the *HP 21MX M-Series and E-Series Computers I/O Interfacing Guide,* part no. 02109-90006. The port is available for user designed hardware in the E-Series only. The F-Series Computer Hardware Floating Point Processor occupies the port.

Table 13-1. MPP Signal Summary

SIGNALS	DESCRIPTION
MPPIO 0 thru 15	Two-way MPPIO signal lines that provide the main data link for the MPP to the computer (CPU) S-bus. Under control of micro-orders affecting the S-bus.
PP5	Output timing line can be used to synchronize with the computer for data transfers.
$\overline{\text{PLR0}}$	Output L-register signal line under control of L-register micro-orders. L-register bit 0 is used for an address line to enable the device connected to the MPP.
$\overline{\text{STOV}}$	Input signal line. State can be tested by the word type III Conditional field OVFL micro-order. Possible use to designate overflow from a set Overflow register.
PIRST	Output signal line. Can be used to sense the IR (IRCM micro-order in Store field).
$\overline{\text{PP1SP}}$	Output signal line activated by a MPP1 micro-order in the word type I Special field. Could be used to designate "first operand to follow."
PP2SP	Output signal line activated by a MPP2 micro-order in the word type I Special field. Could be used to designate "second operand to follow."
MPBST	Output signal line activated by a MPPB micro-order in the word type I Store field. Could be used to generate a store (e.g., repeated four times to store in a 64-bit group of data, where data is being output on the S-bus).
MPBEN	Output signal line activated by a MPPB micro-order in the word type I S-bus field could be used to gate data into the computer on the S-bus (e.g., receive back computed data repeatedly).
$\overline{\text{MPP}}$	Input signal line. State can be tested by the word type III conditional field MPP micro-order. Could be used to sense when device transfer is complete.

13-7. MPP & MBIO CONSIDERATIONS

MPP and MBIO microprograms are used to provide fast alternative I/O paths. Both require the design of special purpose hardware to transfer data to and from the computer, and use of specific micro-orders to provide sequencing and data transfer signals. The major consideration that arises during MPP or MBIO transfers is a control processor freeze induced by either memory refresh or DCPC. Since MBIO and DCPC share the I/O bus, MBIO can contaminate DCPC data if MBIO signals BIOI or BIOO remain enabled during the DCPC transfer. This can be avoided by placing a READ, RJ30 or WRTE micro-order 1 or 2 microinstructions before the IOI or IOO, causing the control processor to freeze.

When a freeze occurs on a WRTE microinstruction the S-Bus to Store operation is performed twice. For instance, the TAB IOI transfer in the following line of microcode is performed twice, once before the freeze, and at the end of the freeze.

```
WRTE    PASS    TAB    IOI(OR MPPB)
```

If the user designed hardware utilizes the signal as an acknowledge or an "increment the buffer pointer", then erroneous information as illustrated below will be transferred. This can be avoided by transferring the data into a scratchpad and the scratchpad into TAB.

```
        PASS    SI     IOI(OR MPPB)
WRTE    PASS    TAB    SI
```

The CPU CNTR represents the lower 8 bits of the IR, of which the lower 6 bits are commonly referred to as the select code when an I/O instruction is executed. For MBIO transfers executing concurrently with DCPC, the MBIO select code does not remain stable for the duration of the MBIO cycle because DCPC takes control of the Select Code bus at P4 (BIOS) and causes unaddressing of the MBIO interface and loss of MBIO data. A different addressing scheme, such as set control, should be employed for the MBIO interface. This will free up the CPU CNTR to be used as a word count register to be incremented or decremented in the special field for MBIO output transfers. The CPU CNTR can not be used during a MBIO input transfer because the I/O bus is disabled from driving the S-bus whenever the Select Code (lower 6 bits of the CNTR) is less than seven.

When using MPP and MBIO, the user designed hardware must account for CPU timing restrictions. The SKF and MPP signals must be stable by P4 of the jump conditional microinstruction to prevent state changes in the conditional logic on the CPU.

MPPB can be falsely decoded from a jump address of a word type IV microinstruction. Consequently qualifying the MPPB micro-order with MPP/or MPP2 will enable the hardware to distinguish "real" from "false" MPBEN signals.

13-8. MPP MICROPROGRAM (E-SERIES ONLY)

An example microprogram that can be used for the MPP is included below. The actual microprogram used must be prepared by you, for your application, using the information in applicable sections of this manual, and in particular, the micro-orders shown in table 13-1. The appropriate CM locations, UIG instructions (main memory/control memory linkage) and microprogramming support software should be used in the same manner as for preparation and use of any other microprogram.

Note that with the MPP design, the key is to have a data buffer large enough to hold the entire burst. The example microprogram operates in a no "hand shaking" manner to transfer data in 256 word bursts. At label BURST data is written into memory using a four microinstruction loop. Additional comments appear in the microprogram.

EXAMPLE 4: MPP MAXIMUM DATA RATE BURST INPUT MICROPROGRAM

```
MICMXE,L                                                SPECIFY 21MX E-SERIES
$CODE=MPP01                                             SAVE MICRO-OBJECT ON DISC
            ORG                         34000B          105600 MAPS TO 34000
*
* MPP MAXIMUM DATA RATE BURST INPUT MICROPROGRAM
*
* THIS MICROPROGRAM:
* 1. INPUTS DATA IN A "BURST" MANNER.
* 2. IS INTERRUPTIBLE BEFORE THE BURST STARTS, BUT IS NOT INTERRUPTIBLE DURING THE BURST,
* 3. ASSUMES THAT THE DEVICE UTILIZING THE MPP FACILITY CONTAINS A DATA BUFFER LARGE
*    ENOUGH TO HOLD THE ENTIRE BURST,
* 4. ASSUMES A BURST MAXIMUM OF 256 WORDS,
* 5. REQUIRES THE FOLLOWING CALLING SEQUENCE
*    LDA COUNT      A = POSITIVE WORD COUNT
*    LDB BUFAD      B = BUFFER ADDRESS
*    OCT 105600     MICROPROGRAM OP CODE
* 6. HAS A MAXIMUM DATA RATE OF ABOUT 1500 KW/S (KILO-WORDS/SECOND) IN A NON-DCPC
*    ENVIRONMENT. IN A TYPICAL DCPC ENVIRONMENT RATES UP TO 500 KW/S ARE ATTAINABLE.
*
            JMP                         BURSTIN         SAVE ENTRY POINTS
            ALGN
BURSTIN                 DEC     S3      P               SAVE NEXT INSTRUCTION ADDRESS
                                CNTR    A               CNTR = + WORD COUNT
*
WAIT        JMP     CNDX    HOI             INTRPT      ANY INTERRUPTS?
                            PASS                        UPDATE STATUS FLAGS
            JMP     CNDX    MPP     RJS     WAIT        NO, WAIT FOR DATA READY
                            INC     PNM     B           M = BUFFER ADDRESS,
*                                                        P = NEXT BUFFER ADDRESS
BURST                           S4      MPPB
            WRTE    MPCK            TAB     S4          WRITE DATA INTO MEMORY
                    DCNT    INC     PNM     P           UPDATE CNTR, P, M
            JMP     CNDX    CNT8    RJS     BURST       COUNT = 0? NO, CONTINUE
*
DONE                            B       P               B = LAST BUFFER ADDRESS + 1
            READ            INC     PNM     S3          FIX P, START NEXT FETCH
                    RTN             A       CNTR        A = 0 = BURST COMPLETE
*
INTRPT                          P       S3              FIX P, EXIT TO HALT-OR-
            JMP                             6            INTERRUPT MICROROUTINE
            END
```

13-9. SUMMARY OF MPP TRANSFER RATES

Some typical transfer rates obtainable using the special facilities of the computer are summarized in table 13-2. Actual figures will depend upon your design.

Table 13-2. Special Facilities Transfer Rate Summary

FUNCTION	RATES
BLOCK I/O DATA TRANSFERS	
Input (256 words or less*):	2.28M bytes/second (maximum)
Output (256 words or less*):	3.17M bytes/second (maximum)
MICROPROGRAMMABLE PROCESSOR PORT	
Burst (16 words or less*):	5.7M words/second (maximum)
Continuous:	1.59M words/second (maximum)
*Transfer rates for larger numbers of words depend upon the size of the block to be transferred. Note that DCPC and memory refresh factors have been incorporated in the figures shown.	

13-10. HARDWARE FLOATING POINT PROCESSOR (F-SERIES ONLY)

The following paragraphs provide information for the user who wishes to directly microprogram the Floating Point Processor (FPP) to perform arithmetic floating-point operations and chained calculations. The FPP data formats and operations are described in addition to FPP microprogramming techniques.

The FPP includes the Arithmetic section and the Control section.

The Arithmetic section includes the hardware required to carry out the FPP commands. It contains the shift registers and arithmetic logic units necessary to perform arithmetic and logical operations on data.

The Control section includes the hardware necessary to control the functions of the Arithmetic section.

In addition, the floating point processor's internal registers may function as an accumulator register. This allows intermediate results to be stored in the FPP for successive floating point operations which eliminates the need to store the result in memory and immediately retrieve it.

13-11. CONTROLLABLE FUNCTIONS

Figure 13-1 illustrates a functional block diagram of the CPU, the Microprogrammable Processor Port (MPP), and the floating point processor.

The MPP provides the link between the floating point processor and the computer.

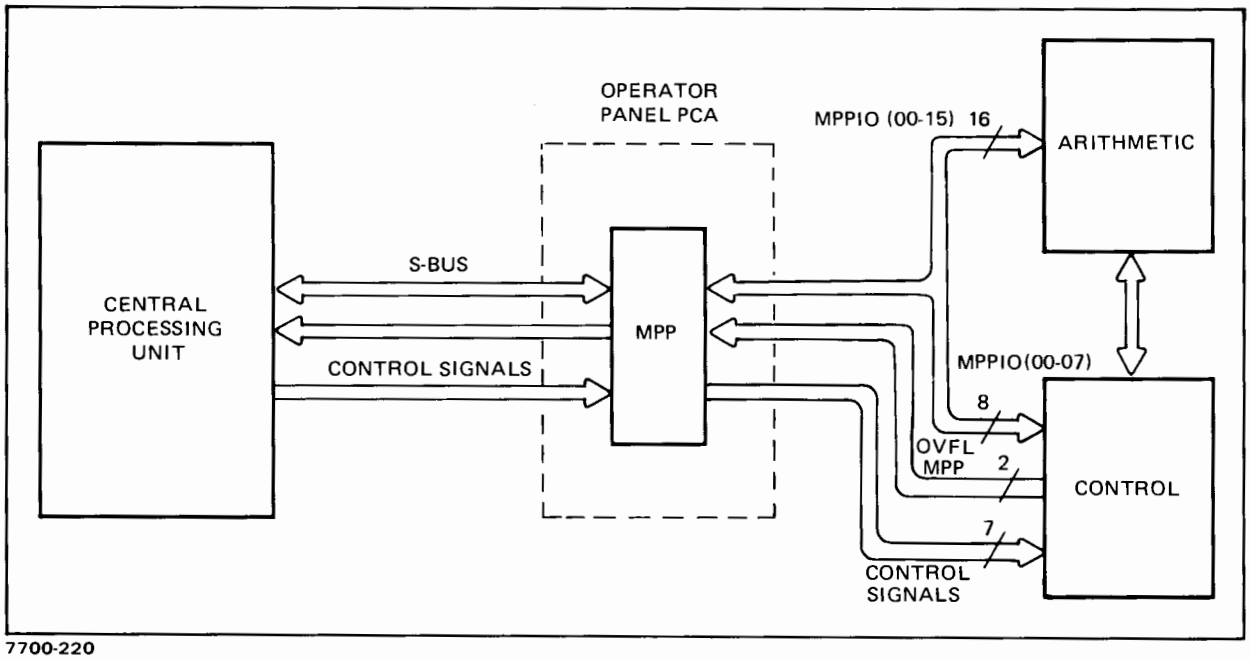


Figure 13-1. FPP Overall Functional Block Diagram

13-12. DATA FORMATS

The two floating-point data formats in figure 13-2 are available to the microprogrammer. Furthermore, the user may specify that the 8-bit exponent of the floating-point formats be "expanded" to 10-bits for internal use only, by the FPP.

13-13. FPP INSTRUCTION WORD FORMAT

The FPP instruction word is used to execute a floating-point operation. The exponent format, type of operation, source of operands, and the operand format are determined by the instruction word.

The FPP instruction word is specified by bits 7-0 of the instruction opcode. The following paragraphs and figure 13-3 describe the instruction word format.

13-14. EXPONENT FORMAT

Bit 7 of the FPP instruction word allows the user to increase the number of exponent bits used by the FPP during operations from 8 bits to 10 bits. Thus during FPP accumulator operations, the intermediate result in the FPP accumulator may exceed the standard 8-bit exponent length without losing accuracy, but the result retrieved from the FPP must be within the underflow or overflow range listed in table 13-3 for bit 7 clear (standard 8-bit exponent). Remember, this 10-bit exponent is internal to the FPP only and is not available to the user as a final result.

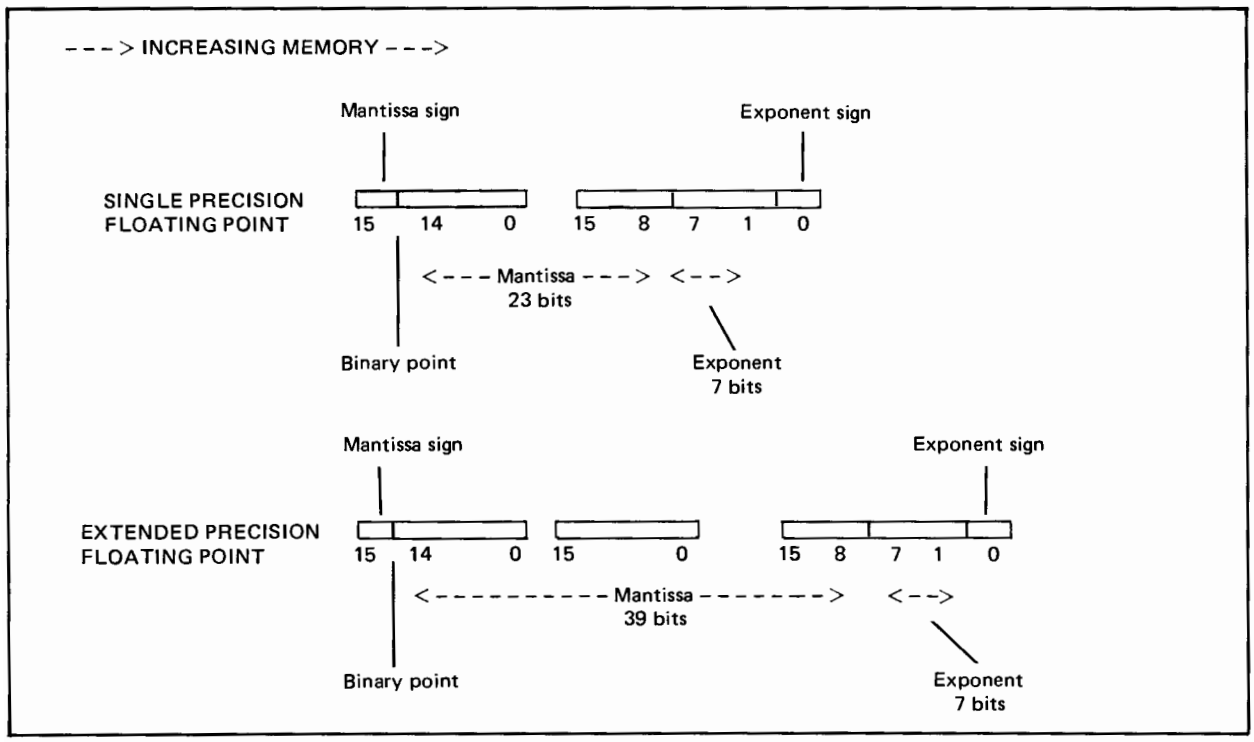

 INCREASING MEMORY --->
Mantissa sign
Exponent sign
SINGLE PRECISION FLOATING POINT
15 14 0 15 8 7 1 0
<--- Mantissa ---> <-->
23 bits
Binary point
Exponent
7 bits
Mantissa sign
Exponent sign
EXTENDED PRECISION FLOATING POINT
15 14 0 15 0 15 8 7 1 0
<---------- Mantissa -------> <-->
39 bits
Binary point
Exponent
7 bits

Figure 13-2. Floating Point Data Format

Table 13-3. Overflow and Underflow Ranges

WORD LENGTH	BIT 7 = 0	
	OVERFLOW RANGE (LARGEST NEGATIVE, LARGEST POSITIVE)	UNDERFLOW RANGE (SMALLEST NEGATIVE, SMALLEST POSITIVE)
Two-word	-2^{127}, $(1 - 2^{-23})\ 2^{127}$	$-2^{-129}\ (1 + 2^{-22})$, 2^{-129}
Three-word	-2^{127}, $(1 - 2^{-39})\ 2^{127}$	$-2^{-129}\ (1 + 2^{-38})$, 2^{-129}
	BIT 7 = 1 (NOTE 3)	
Two-word	-2^{511}, $(1 - 2^{-23})\ 2^{511}$	$-2^{-513}\ (1 + 2^{-22})$, 2^{-513}
Three-word	-2^{511}, $(1 - 2^{-39})\ 2^{511}$	$-2^{-513}\ (1 + 2^{-38})$, 2^{-513}

NOTE:
1. If a result lies outside the given overflow range, the maximum positive floating point number (all ones) is returned and the CPU overflow flag is set.
2. If a result lies inside the given underflow range, zero is returned as the result and the CPU overflow flag is set.
3. These overflow and underflow ranges pertain only to two- and three-word intermediate results left in the FPP.

7	6 5 4	3 2	1 0
EXPONENT FORMAT	OPERATION	OPERAND SOURCE	OPERAND LENGTH
0 Standard 1 Expanded	000 Add 001 Subtract 010 Multiply 011 Divide 110 Reserved 111 Reserved	00 Both operands in CPU. 01 First operand in CPU; second in accumulator. 10 First operand in accumulator; second in CPU. 11 Both operands in accumulator.	00 Two words 01 Three words 10 Reserved 11 Reserved
	100X0 Fix to single integer 100X1 Fix to double integer 101X0 Single integer to floating point 101X1 Double integer to floating point X= 0, for operand from CPU; 1, for operand from accumulator		

Figure 13-3. FPP Instruction Word Format

13-15. FPP OPERATION

Bits 6-4 of the FPP instruction word specify the arithmetic operation (add, subtract, multiply, or divide). Each of these arithmetic operations requires two operands, both of which must be the same precision — i.e., both operands 32 bits or 48 bits.

Bits 6-2 of the instruction word specify a "fix" or "float" operation with bit 3 indicating whether the single operand is in the FPP accumulator or will be transferred from the CPU.

13-16. OPERAND SOURCE

When executing an arithmetic operation, bits 3 and 2 of the FPP instruction word specify the "source" of the first and second operand, respectively. A "1" indicates the operand is in the FPP accumulator; a "0" indicates the operand will be transferred from the CPU.

For example, if bits 3 and 2 equal "1" and "0", respectively, the first operand required for an arithmetic operation is in the FPP accumulator and the second operand will be transferred from the CPU.

When executing a "fix" or "float" operation, only bit 3 of the FPP instruction word specifies the operand source.

13-17. OPERAND LENGTH

Bits 1 and 0 of the FPP instruction word specify the operand length. Operands consisting of two or three words may be specified. (Refer to figure 13-2 for the floating-point data format.)

For example, to perform extended precision floating-point operations bits 1 and 0 must be "0" and "1", respectively.

For "fix" and "float" operations, bit 2 of the FPP instruction word specifies the integer length. Bit 2 equal to "1" indicates a 32-bit integer, whereas bit 2 equal to "0" indicates a 16-bit integer.

13-18. DATA OPERATIONS

Listed below are the operations performed by the FPP and the operand sequence. Each operation, except for "fix" and "float", requires two normalized operands.

OPERATION	FIRST OPERAND (A)	SECOND OPERAND (B)
Addition (A+B)	Augend	Addend
Subtraction (A−B)	Minuend	Subtranhend
Multiplication (A) (B)	Multiplicand	Multiplier
Division (A/B)	Dividend	Divisor
Fix to Integer	Floating Point Number	—
Integer to Float	Integer	—

13-19. FIX AND FLOAT OPERATIONS

The "fix" operations are used to convert a floating point number to either single or double integer format and the "float" operation is used to convert a single or double integer to floating point format.

For "fix to single integer" operations, zero is returned as the result if the magnitude of the exponent of the floating point number is <0. An overflow condition will result if the magnitude of the exponent of the floating point number is >=16.

For "fix to double integer" operations, zero is returned as the result if the magnitude of the exponent of the floating point number is <0. An overflow condition will result if the magnitude of the exponent of the floating point number is >=32.

13-20. ACCUMULATOR OPERATIONS

The FPP accumulator capabilities allow the microprogrammer to perform chained floating point operations. This feature eliminates the need to store a result in memory and then immediately fetch it for the next operation, thus reducing memory overhead time. For example, the result of a floating point operation may be left in the FPP to serve as either the divisor or dividend in a subsequent divide operation.

13-21. MPP MICRO-ORDERS

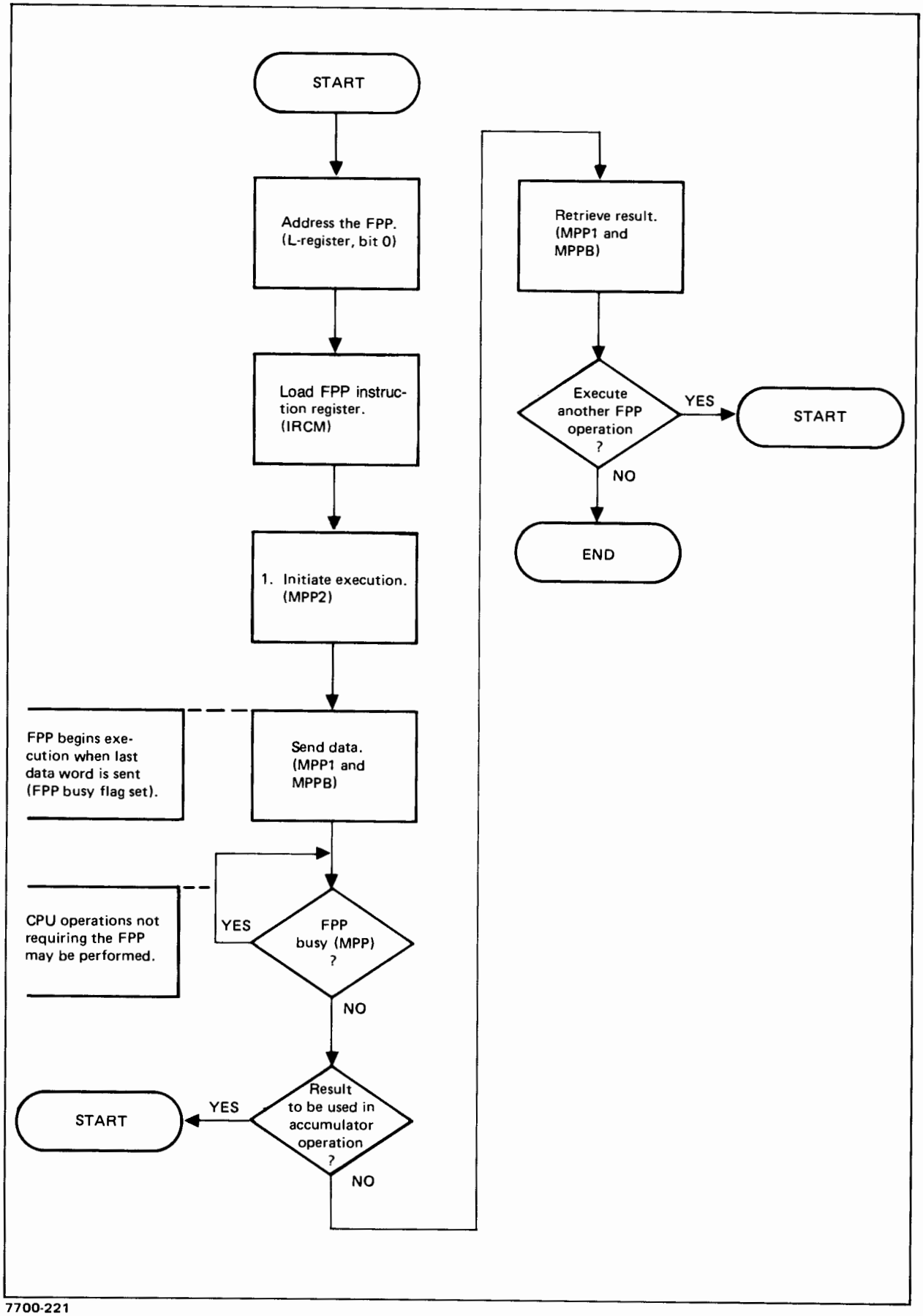
The following paragraphs describe the MPP micro-orders required to microprogram the FPP. Figure 13-4 illustrates the microprogramming sequence used to execute a typical FPP operation.

13-22. FPP INSTRUCTION STORE

The IRCM micro-order causes the lower eight bits of the CPU S-bus to be loaded into the FPP instruction register if the FPP is not currently executing an instruction. (The FPP instruction register may be loaded without addressing the FPP.)

The following microinstruction will prepare the FPP to multiply a three-word operand transferred from the CPU by the three-word operand in the FPP accumulator.

```
                ALU/
        OP/     MOD/            S-BUS/
LABEL   BRCH SPCL COND   STR     ADDR      COMMENT

                   .
                   .
                   .
        IMM        LOW   IRCM    045B      LOAD FPP INSTRUCTION REGISTER.
                   .
                   .
                   .
```



7700-221

Figure 13-4. Typical FPP Microprogramming Sequence Flowchart

13-23. FPP ADDRESSING

The FPP must be addressed before any operation may be initiated and before testing for the FPP ready condition.

Bit 0 of the CPU L-register equal to "0" is used to address the FPP.

The FPP must be addressed at least one microinstruction before executing micro-orders MPP2, MPPB, or MPP1, and two microinstructions before executing the MPP micro-order.

NOTE

A microinstruction may be saved since the CPU FETCH routine clears the L-register and CPU flag.

13-24. INSTRUCTION EXECUTION

The MPP2 micro-order in the Special field causes the FPP to initiate execution of the instruction held in the FPP instruction register. Execution begins when the last word of the operand(s) is transferred to the FPP by the user. The FPP busy flag is set until execution is completed.

13-25. OPERAND TO FPP

The MPPB micro-order in the Store field and the MPP1 micro-order in the Special field are used to transfer 16 bits (the most-significant word first) of an operand from the CPU to the FPP.

Sixteen bits, where bit 15 is the most-significant bit, are transferred each time the micro-orders are executed. Therefore, MPPB and MPP1 must be executed twice for each two-word operand and three times for each three-word operand.

The following example is one way to transfer a three-word operand to the FPP beginning at the address in the P-register.

```
                ALU/
        OP/             MOD/            S-BUS/
LABEL   BRCH    SPCL    COND    STR     ADDR    COMMENT

                                .
                                .
                                .
        READ            INC     PNM     P       GET ADDRESS OF
                                                FIRST WORD.
                MPP1    PASS    MPPB    TAB     SEND FIRST WORD.
        READ            INC     PNM     P
                MPP1    PASS    MPPB    TAB     SEND SECOND WORD.
        READ            INC     PNM     P
                MPP1    PASS    MPPB    TAB     SEND THIRD WORD.
                                .
                                .
                                .
```

13-26. RESULT TO CPU

The MPPB micro-order in the S-bus field and the MPP1 micro-order in the Special field are used to transfer 16 bits (the most-significant word first) of the result from the FPP to the CPU. Sixteen bits, where bit 15 is the most-significant bit, are transferred each time the micro-orders are executed.

Note that the result transferred from the FPP must not be stored in the T-register in the same microinstruction since a memory refresh or the Dual Channel Port Controller (DCPC) could alter the T-register before the WRTE is executed. Instead, store the result in a temporary CPU register and in a subsequent microinstruction, transfer the result to the T-register.

The user should also be aware that the result is rounded during operation execution and not when it is retrieved. Thus, any result retrieved at a precision lower than that at which it was generated will result in an answer that has been truncated.

A three-word result is transferred from the FPP to memory starting at the address in the P-register as follows:

```
                ALU/
        OP/             MOD/            S-BUS/
LABEL   BRCH    SPCL    COND    STR     ADDR    COMMENT

                                .
                        INC     PNM     P       GET RESULTANT DESTINATION ADDRESS.
                MPP1            S4      MPPB    GET FIRST WORD OF RESULT FROM FPP.
        WRTE                    TAB     S4      STORE FIRST WORD OF RESULT.
                        INC     PNM     P       GET ADDRESS FOR SECOND WORD.
                MPP1            S4      MPPB    GET SECOND WORD.
        WRTE                    TAB     S4      STORE SECOND WORD.
                        INC     PNM     P
                MPP1            S4      MPPB    GET LAST WORD.
        WRTE                    TAB     S4
                                .
                                .
                                .
```

13-27. MPP1 MICRO-ORDER CONSIDERATIONS

The MPP1 micro-order resets the FPP control logic if data is not being transferred to and from FPP. Also, MPP1 must be specified whenever data is sent to or from the FPP.

13-28. FPP COMPLETE TEST

The MPP micro-order is used to determine if the FPP has completed the requested operation. The FPP must be addressed (using L-register bit 0) at least two microinstructions before MPP is tested.

NOTE

> The FPP ready status cannot be made until at least two microinstructions after MPP2 Special has been specified.

13-29. OVERFLOW DETECTION

The FPP sets the CPU overflow bit at the trailing edge of the first MPBEN signal (MPPB micro-order). Therefore, an overflow condition cannot be tested until after two microinstructions following the first MPPB S-Bus micro-order. If an overflow condition occurs, the user must clear the overflow flag bit.

13-30. MPP MICRO-ORDER SUMMARY

Table 13-4 summarizes the micro-orders required to microprogram the FPP.

13-31. FPP MICROPROGRAMMING CONSIDERATIONS

The following paragraphs describe some key points that the user should be aware of when writing microprograms which use the FPP.

13-32. FPP OPERATION EXECUTION TIMES

Table 13-5 lists the execution times for chained floating point calculations in which intermediate results are not transferred to and from the computer.

Table 13-4. Summary of FPP Control Micro-orders

MICRO-ORDER	FIELD	MEANING
IRCM	STORE	Load lower eight bits (FPP instruction) of CPU S-bus into FPP instruction register. The FPP busy flag must be clear (FPP ready) before executing this micro-order.
L	STORE	Bit 0 clear used to address FPP.
MPP2	SPECIAL	Execute instruction held in FPP instruction register and set FPP busy flag. Address FPP at least one microinstruction before executing MPP2.
MPPB and MPP1	STORE SPECIAL	Store 16 bits of operand into FPP. Address FPP at least one microinstruction before executing MPPB and MPP1.
MPPB and MPP1	S-BUS SPECIAL	Transfer 16 bits of result to CPU. Address FPP at least one microinstruction before executing MPPB and MPP1. Do not store the result in the T-register in the same microinstruction.
MPP1	SPECIAL	Reset FPP control logic.
MPP	COND	Test FPP ready status. Address FPP at least two microinstructions before executing MPP.

Table 13-5. FPP Operation Internal Execution Times

INSTRUCTION	COMPUTATION TIME (μsec)		
	MINIMUM	TYPICAL	MAXIMUM
Single-precision Floating Point			
Add/Subtract	0.68	1.36	3.28
Multiply	1.96	2.21	2.46
Divide	2.12	3.01	3.90
Conversion to single integer	0.67	1.38	1.85
Conversion to double integer	0.67	2.45	3.27
Conversion from single integer	0.63	1.25	1.78
Conversion from double integer	0.63	2.33	2.93
Extended-precision Floating Point			
Add/Subtract	0.68	1.36	4.16
Multiply	2.75	3.14	3.52
Divide	2.94	4.78	6.62
Conversion to single integer	0.67	1.38	1.85
Conversion to double integer	0.67	2.45	3.27
Conversion from single integer	0.63	1.25	1.78
Conversion from double integer	0.63	2.33	2.93

13-33. EXECUTION IN PROCESS

Once the FPP has begun execution of an operation, CPU operations not requiring use of the FPP, or a timing routine which waits for the FPP to complete execution may be executed.

If non-FPP operations are performed, ensure that upon return the FPP is addressed (L bit 0) at least two microinstructions prior to testing if the FPP is ready. In addition, the FPP instruction register must be reloaded with the proper FPP instruction if an IRCM micro-order had been executed and the result held in the FPP is to be transferred to the CPU.

If a timing routine is used, the time allowed for the FPP to complete an operation and the action required in the event of an FPP failure must be determined. A simple timing routine is shown below:

```
                    ALU/
        OP/         MOD/          S-BUS/
LABEL   BRCH   SPCL COND   STR    ADDR      COMMENT

                     .
                     .
                     .
WAIT    IMM    COV  CMLO   S3     337B      SET WORD COUNT CONSTANT
                                             AND CLEAR OVERFLOW.
WAIT1   RTN    CNDX MPP                      IF FPP DONE, RETURN.
                    DEC    S3     S3        DECREMENT COUNTER
        JMP    CNDX AL15   RJS    WAIT1
        JMP                        ERROUT    JMP TO ERROR ROUTINE.
                     .
                     .
                     .
```

13-34. INTERRUPT CONSIDERATIONS

If your microprogram is written such that interrupts are detected (which is recommended), it should execute a JSB to a microroutine that saves whatever is necessary (including intermediate results in the FPP) to allow the microprogram to continue after the interrupt is serviced, or to provide for complete restart of the microprogram.

The microroutine should also ensure that the FPP is addressed and the proper FPP instruction is stored in the FPP after servicing the interrupt.

13-35. MICROPROGRAMMED FPP OPERATION EXAMPLE

This paragraph contains an example on directly microprogramming the Hardware Floating Point Processor. The microprogram sums the product of two, one-dimensional arrays and stores the floating point result in the A and B registers. Figure 13-5 is the flowchart for the microprogram. Note that the program is interruptable. The microprogram assumes the following calling sequence is used:

```
OCT 105600     INVOKE FPP PROGRAM

NOP            USED FOR CURRENT ITERATION IF INTERRUPTED

DEF DIM        DIMENSION OF ARRAYS

DEF ADDRA      ADDRESS OF ARRAY A

DEF ADDRB      ADDRESS OF ARRAY B
```


13-36. MICROPROGRAMMING THE FLOATING POINT PROCESSOR

The following is a summary of the rules for user microprograms.

1. The FPP must be addressed before asserting any control signals except IRST. Address the FPP by setting the L-register bit 0 to the address of the FPP at least one microinstruction before asserting MPP2, MPPB in the store or S-bus field or MPP1. The FPP must be addressed at least two microinstructions before testing MPP. If an overflow occurs, the FPP does not set the CPU Overflow Flip-Flop until the trailing edge of the first MPPB in the S-bus field. Therefore overflow should not be tested until at least two microinstructions following the first S-bus field MPPB of an operation.

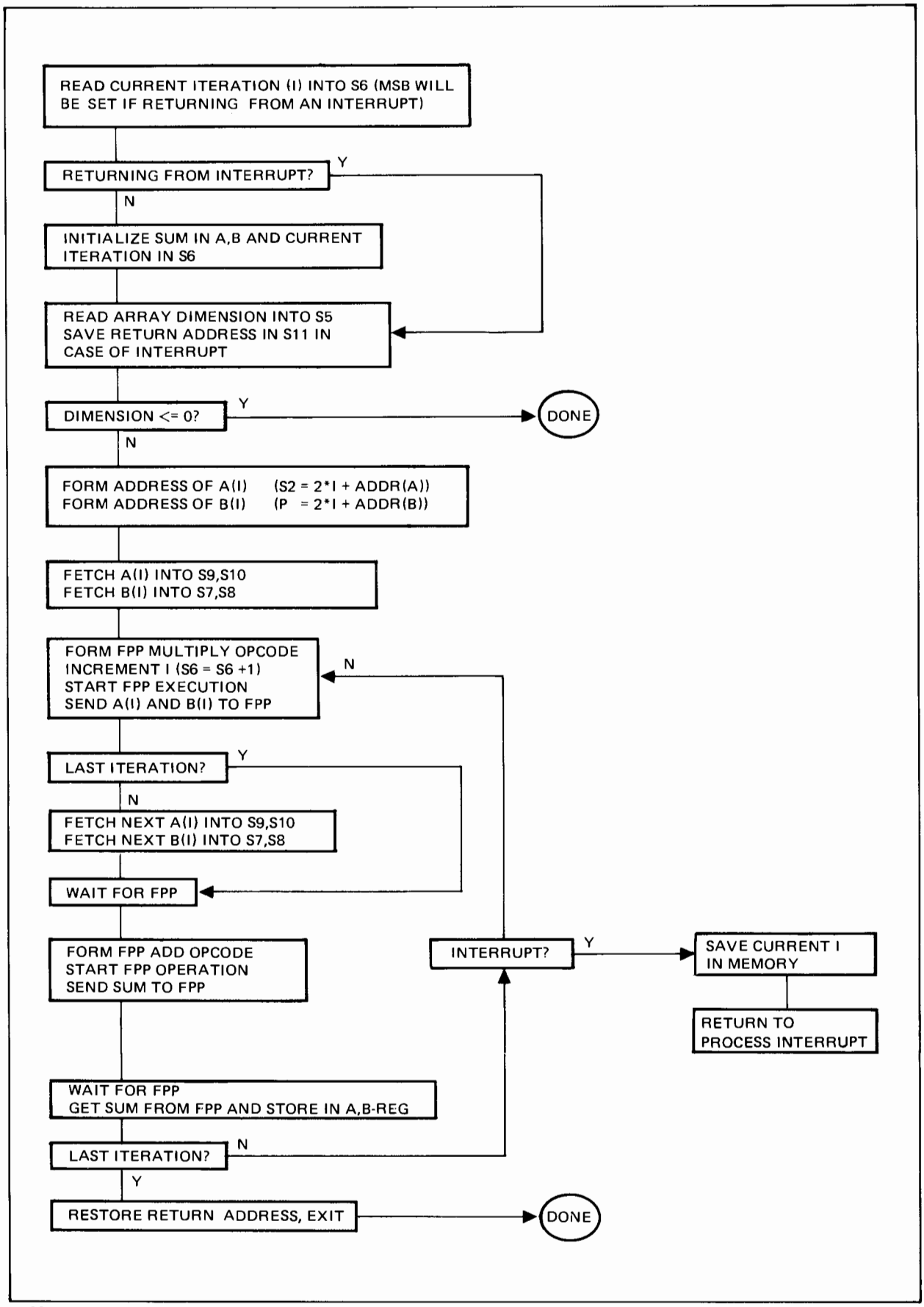
2. Assert MPP1 in the special field when asserting MPPB in the store or S-bus field. The FPP does not recognize MPPB unless it has been addressed and MPP1 is also asserted.

3. If a microinstruction S-bus field contains MPPB, the store field must not contain TAB. The result may not be retrieved from the FPP and stored in the memory data register in the same microinstruction, since memory refresh or DMA freeze may destroy the memory data register contents. Therefore, store the result in a temporary CPU register, and then transfer the result to the memory data register in a subsequent microinstruction.

4. Ensure that bits 1 and 0 of the FPP instruction register are set to the proper operand word length as described in paragraph 13-15. Also, in the case of FIX, IR bits 6-4 must equal 100, before retrieving the FIX result. If a result is retrieved from the process at a precision lower than the operation just performed, the result is truncated, rather than rounded. If the result is retrieved at a higher precision, the lower mantissa bits are zeros.

5. Floating point operands, except for zero, issued to the FPP must be normalized (sign bit is not the same sense as the most significant mantissa bit). Note that the FPP normalizes all of its floating point results, except for zero.

6. When executing chained operations, the FPP instruction register bits 1 and 0 may be changed in order to retrieve a result of precision that differs from the operation performed. For example after performing a 48 bit ADD, a 32 bit result may be retrieved from the FPP. However the precision of the next operation must agree with that of the previous floating point (48 bit ADD) operation.



7700-222

Figure 13-5. FPP Microprogramming Example Flowchart

EXAMPLE 5: FPP SUMS THE PRODUCT OF TWO ONE-DIMENSIONAL ARRAYS

```
                          MICMXE,L
                          $CODE='SUMAB::-48,REPLACE
                                   ORG                 34000B
                          HORI     EQU                 00006B
                          FETCH    EQU                 00000B
                          *
                          *
                          ************************************
                          * READ CURRENT ITERATION (I) INTO S6  *
                          * (MSB WILL BE SET IF RETURNING FROM  *
                          * AN INTERRUPT)                      *
                          ************************************
                          *
34000  017 101254         SUMAB         SOV  CMPS S6   TAB       S6 = CURRENT I
                          *
                          ****************************
                          * RETURNING FROM INTERRUPT?  *
                          ****************************
                          *
34001  327 140242                  JMP  CNDX AL15 RJS  REENT     JUMP IF RE-ENTERING
                          *
                          ****************************************************
                          * INITIALIZE SUM IN A,B AND CURRENT ITERATION IN S6  *
                          ****************************************************
                          *
34002  006 036147                           ZERO A               INITIALIZE SUM
34003  006 036207                           ZERO B
34004  006 037247                           ZERO S6              INITIALIZE CURRENT I
                          *
                          ****************************************************
                          * READ ARRAY DIMENSION INTO S5                       *
                          * SAVE ADDRESS IN CASE OF INTERRUPT (S11)            *
                          ****************************************************
                          *
34005  227 174707         REENT    READ      INC  PNM  P          START READ OF IMAX
34006  000 075507                            DEC  S11  P          SAVE P IN S11
34007  307 005047                  JSB                 INDIRECT  RESOLVE INDIRECTS
34010  010 001207                            PASS S5   TAB        S5 = IMAX
                          *
                          *************
                          * IMAX <= 0? *
                          *************
                          *
34011  327 104002                  JMP  CNDX AL15      DONE      FORGET IT IF IMAX<0
34012  320 004002                  JMP  CNDX ALZ       DONE      DITTO IF IMAX=0
                          ************************************************
                          * FORM ADDRESS OF ARRAY A (S2 = 2*I + ADDR(A))  *
                          * FORM ADDRESS OF ARRAY B (P  = 2*I + ADDR(B))  *
                          * NOTE THAT L0 = 0 WHEN ADDRESS IS FORMED       *
                          ************************************************
                          *
34013  227 174713                  READ COV  INC  PNM  P          START READ ON ADDR(A)
34014  001 152507                            DBLS L    S6         SET L = 2 * I
34015  307 005047                  JSB                 INDIRECT  RESOLVE INDIRECTS
34016  003 033047                            ADD  S2   M          S2 = ADDR(A(I))
34017  227 174707                  READ      INC  PNM  P          START READ ON ADDR(B)
34020  307 005047                  JSB                 INDIRECT  RESOLVE INDIRECTS
34021  003 033707                            ADD  P    M          P = ADDR(B(I))
                          *
                          **************************
                          * FETCH B(I) INTO S7,S8   *
                          * FETCH A(I) INTO S9,S10  *
                          **************************
                          *
34022  227 174707                  READ      INC  PNM  P          START READ ON B(I)
34023  007 174707                            INC  PNM  P          BUMP ADDR(B(I))
34024  230 001307                  READ      PASS S7   TAB        SAVE B(I) IN (S7 S8)
34025  010 042647                            PASS M    S2
34026  007 143047                            INC  S2   S2         BUMP ADDR(A(I))
34027  230 001347                  READ      PASS S8   TAB
34030  010 042647                            PASS M    S2         SET M = ADDR(A(I))
34031  007 143047                            INC  S2   S2         BUMP ADDR(A(I))
34032  230 001407                  READ      PASS S9   TAB        SAVE A(I) IN (S9 S10)
34033  010 001447                            PASS S10  TAB
```

```
0076                        *
0077                        ********************************
0078                        * FORM FPP MULTIPLY OPCODE      *
0079                        * INCREMENT I (S6 = S6 + 1)     *
0080                        * INITIATE FPP EXECUTION        *
0081                        * SEND A(I) AND B(I) TO FPP     *
0082                        ********************************
0083                        *
0084 34034  340 100607  LOOP     IMM       LOW  IRCM 040B      FPP MPY OPCODE
0085 34035  007 153251               MPP2 INC  S6   S6        START FPP, BUMP I
0086 34036  010 054432               MPP1 PASS MPPB S7        SEND OP1 = B(I)
0087 34037  010 056432               MPP1 PASS MPPB S8
0088 34040  010 060432               MPP1 PASS MPPB S9        SEND OP2 = A(I)
0089 34041  010 062432               MPP1 PASS MPPB S10

0091                        *
0092                        ********************
0093                        * LAST ITERATION?  *
0094                        ********************
0095                        *
0096 34042  007 152507                    INC  L    S6        SET L = I + 1
0097 34043  144 150762           LWF  L1  SUB       S5        SET FLAG = 1 IF DONE
0098 34044  006 036507                    ZERO L              RESET L FOR FPP
0099 34045  334 003002           JMP  CNDX FLAG      LOOP1     JUMP IF DONE
0100                        *
0101                        ******************************
0102                        * FETCH NEXT B(I) INTO S7,S8  *
0103                        ******************************
0104                        *
0105 34046  227 174707           READ      INC  PNM  P
0106 34047  007 174707                    INC  PNM  P         BUMP ADDR(B(I))
0107 34050  230 001307           READ      PASS S7   TAB       GET NEXT B(I)
0108 34051  010 001347                    PASS S8   TAB
0109                        *
0110                        ******************************
0111                        * FETCH NEXT A(I) INTO S9,S10 *
0112                        ******************************
0113                        *
0114 34052  230 042647           READ      PASS M    S2        SET M = ADDR(A)
0115 34053  007 143047                    INC  S2   S2        INC ADDR(A)
0116 34054  010 042647                    PASS M    S2
0117 34055  230 001407           READ      PASS S9   TAB
0118 34056  007 143047                    INC  S2   S2
0119 34057  010 001447                    PASS S10  TAB
0120                        *
0121                        *****************
0122                        * WAIT FOR FPP  *
0123                        *****************
0124                        *
0125 34060  306 044402  LOOP1    JSB  CNDX MPP  RJS  WAIT      WAIT FOR FPP
0126                        *
0127                        ******************************************************
0128                        * FORM FPP ADD OPCODE (ACCUMULATOR HAS FIRST OPERAND)  *
0129                        * INITIATE FPP EXECUTION                               *
0130                        * SEND SUM OPERAND TO FPP                              *
0131                        ******************************************************
0132                        *
0133 34061  340 020607           IMM       LOW  IRCM 010B      FPP ADD OPCODE
0134 34062  010 036751               MPP2                      START FPP
0135 34063  010 006432               MPP1 PASS MPPB A          SEND OP1 = SUM
0136 34064  010 010432               MPP1 PASS MPPB B
```

```
0138                    *
0139                    ****************************************
0140                    * WAIT FOR FPP                         *
0141                    * GET SUM FROM FPP AND STORE IN A,B-REG *
0142                    ****************************************
0143                    *
0144 34065  306 044402  LOOP2    JSB  CNDX MPP  RJS  WAIT      WAIT FOR FPP
0145 34066  010 020172               MPP1 PASS A    MPPB      SAVE SUM IN (A B)
0146 34067  010 020232               MPP1 PASS B    MPPB
0147                    *
0148                    ******************
0149                    *LAST ITERATION?  *
0150                    ******************
0151                    *
0152 34070  334 004002           JMP  CNDX FLAG      DONE      JUMP IF ALL DONE
0153                    *
0154                    **********************
0155                    * TEST FOR INTERRUPT  *
0156                    **********************
0157                    *
0158 34071  323 141602           JMP  CNDX HOI  RJS  LOOP      LOOP IF NO INT
0159                    *
0160                    ************************
0161                    * TEST FOR SINGLE STEP  *
0162                    ************************
0163                    *
0164 34072  336 041602           JMP  CNDX NSNG RJS  LOOP      LOOP IF SNGL STEP
0165                    *
0166                    ******************
0167                    * SAVE CURRENT I  *
0168                    * IN MEMORY       *
0169                    ******************
0170                    *
0171 34073  000 065707  INT                  DEC  P    S11       SET P   = SAVE ADDR
0172 34074  010 074647                       PASS M    P         SET M = SAVE ADDR
0173 34075  017 153247                       CMPS S6   S6        SET S6 = -S6 - 1
0174 34076  210 052036           WRTE MPCK PASS TAB  S6        SAVE IN MEMORY
0175                    *
0176                    *********************
0177                    * SERVICE INTERRUPT  *
0178                    *********************
0179                    *
0180 34077  320 000307           JMP                 HORI      HANDLE INTERRUPT

0182                    *
0183                    **************************
0184                    * RESET SAVE WORD        *
0185                    * RESTORE RETURN ADDRESS *
0186                    * RETURN TO MACRO CODE   *
0187                    **************************
0188                    *
0189 34100  000 065707  DONE                 DEC  P    S11       SET P = SAVE ADDRESS
0190 34101  007 174707                       INC  PNM  P         SET M = SAVE ADDRESS
0191 34102  006 037207                       ZERO S5             CREATE A ZERO TO
0192 34103  210 050036           WRTE MPCK PASS TAB  S5        RESET SAVE WORD
0193 34104  353 170507           IMM       CMLO L    374B      SET L = 3
0194 34105  003 075707                       ADD  P    P         SET P = RETURN ADDR
0195 34106  227 174707           READ      INC  PNM  P         START READ
0196 34107  320 000007           JMP                 FETCH     RETURN
0197                    *
0198                    **************
0199                    * WAIT LOOP  *
0200                    **************
0201                    *
0202 34110  340 100547  WAIT     IMM       LOW  CNTR 040B      SET COUNTER = 32
0203 34111  366 002002  WTLP     RTN  CNDX MPP                 RETURN IF DONE
0204 34112  010 036765               DCNT                      DECREMENT COUNTER
0205 34113  366 000742           RTN  CNDX MPP                 RETURN IF DONE
0206 34114  326 144442           JMP  CNDX CNT8 RJS  WTLP      ELSE LOOP 32 TIMES
0207 34115  355 165047           IMM       CMHI S2   172B      SET IRCM = MIA 00
0208 34116  010 042614               SOV  PASS IRCM S2
0209 34117  010 036746               IOG                       CAUSE MP INT
0210 34120  327 004007           JMP                 DONE      RETURN
```

```
                          *
                          ******************************
                          * RESOLVE INDIRECT ADDRESSING  *
                          ******************************
                          *
34121  230 000647  INDIRECT READ      PASS M    TAB       SAVE ADDRESS IN M
34122  367 140002          RTN  CNDX AL15 RJS            RETURN IF RESOLVED
34123  323 145042          JMP  CNDX HOI  RJS  INDIRECT  KEEP RESOLVING
34124  230 036747          READ                          BUT IF HOI AND
34125  336 045042          JMP  CNDX NSNG RJS  INDIRECT  NO SINGLE STEP
34126  000 065707                    DEC  P    S11       RESTORE P AND
34127  320 000307          JMP                 HORI      HANDLE INTERRUPT
                          *
                          *
                               END

END OF PASS 2: NO ERRORS
```

PART IV
Microprogramming Examples

Section 14
MICROPROGRAMS

MICROPROGRAMS

SECTION 14

The microprogramming examples in this section are arranged in order of advancing complexity and illustrate (among other things) concepts presented throughout the rest of this manual. Each microprogram is complete in itself and may be used directly in the computer or may be used as an example for creating your own microprograms. The following assumptions are made for the use of material in this section.

- The microprogramming support software (the microassembler, Microdebug Editor, driver DVR36, and WLOAD) must have been loaded into the RTE system. It is also assumed that the system equipment configuration (HP 21MX E-Series Computer, HP 13197A WCS, etc. installation) is compatible for microprogramming. (Refer to section 3 in this manual for more information on the steps necessary for preparing to microprogram.)

- RTE system equipment table entries (SC-to-LU relationship) must have been made.

The first examples use the MDE features to prepare and execute the microprograms. If you use the RTE Interactive Editor, then, the RTE Microassembler to prepare the larger examples, use the RTE Interactive Editor Tab function for determining the starting columns for micro-order fields. (Refer to section 8 for more information on preparation with the microassembler.

When you are ready to microassemble from the system LS tracks, the microassembler may be scheduled and used following the procedures outlined in section 9 of this manual. Control commands, error messages, etc., are described in section 9. Psuedo-microinstructions, etc., that you will need when preparing your source are described in section 8. The microassembled object will be placed in an RTE file you designate by the $CODE command and will be ready to be accessed and loaded into WCS. Information on WCS support software use (for moving your microprogram into WCS or out of WCS) may be found in section 11 in this manual.

In addition to the examples included in this section you may be interested in the microprogrammable algorithms appearing in three other reference manuals:

- Computer Approximations.
- The ACM Manual (Association of Computer Manufacturers).
- Art of Computer Programming, Volume III.

14-1. WCS INITIALIZATION

WCS boards must be initialized (i.e., be assigned subchannel base addresses) for the transfer of microprogram object code to the boards. WCS initialization is required whenever the RTE system is booted up. Complete information required to write WCS initialization programs is given in the Driver DVR36 manual.

The WCS boards can be initialized and controlled by the FMGR CN command as follows:

CN,*lu*,*n* [,*ba*]

where:

lu = a WCS LU number;

n = 1 = assign base address to WCS LU;

n = 2 = enable WCS LU;

n = 3 = disable WCS LU;

n = 4 = down WCS LU;

ba = base address to be assigned to WCS LU.

For example, to initialize and enable a 1K WCS board having LU number 11 and 12, the following sequence of CN commands could be used:

```
CN,11,1,34000B
CN,11,2
CN,12,1,35000B
CN,12,2
```

If the above command sequence were going to be used frequently, it could be set up as a TR (transfer) file and saved for later execution. Refer to the Batch-Spool Monitor Reference Manual for information on TR files.

14-2. MICROPROGRAMMING WITH MDE

The following three console run sheets provide examples of interactive sessions that illustrate the simplicity of using the Microdebug Editor program (MDEP). In the first console run sheet you use MDEP to prepare and execute a single-statement "microprogram" that simply decrements the A-register. Next, MDEP is used to prepare and execute a microprogram that performs a logical "and" on two octal numbers. This example illustrates the use of the READ and WRTE micro-orders. The MDE commands used in these examples are: LU, REplace, SEt, RUn, SHow, PR, EXit, and Abort. (Refer to section 10 for details on the MDE commands.) Note that the Abort (A) command only terminates another MDE command and does not terminate MDEP. Note also that these miniature "microprograms" are executable by MDEP without apparent microassembly.

If you did not attend the HP RTE microprogramming course, you may find it helpful to use these examples (following the run sheets step-by-step) as exercises for becoming familiar with MDEP. Make sure to initialize your WCS board(s) and use LU numbers appropriate for your computer installation. All operator entries are underlined in all examples.

EXAMPLE 1: DECREMENT A REGISTER, CONSOLE RUN SHEET

```
*ON,FMGR
:RU,MDEP

COMPUTER TYPE:  1=21MX,2=21MX E-SERIES
TYPE(1 OR 2)?2 (NOTE: 2 IS THE RESPONSE FOR F-SERIES ALSO.)
$LU,13

LU#       RANGE     STATUS
13  034000--034777   1
$RE,34000B
34000   LGS   STFL   NAND   S1    CNTR
$$READ,RTN,DEC,A,A
34000   READ  RTN    DEC    A     A
$$/
$SE,A
A   = 0


A   = 0
A   = 12345B
A   = 12345
A   = A
$RU,105600B
RETURN= P+01
$SE,A
A   = 12344


A   = 12344
A   = A
$EX
 $END MDEP
:EX
 $END FMGR
```

EXAMPLE 2: READ/WRITE MEMORY, CONSOLE RUN SHEET (Sheet 1 of 2)

```
*ON,FMGR
:RU,MDEP

COMPUTER TYPE:  1=21MX,2=21MX E-SERIES
TYPE(1 OR 2)?2
$LU,13

LU#       RANGE      STATUS
13  034000--034777   1
$RE,34000B,34003B
34000   LGS          XOR    S3     X
$$READ,NOP,PASS,L,A
34000   READ         PASS   L      A
$$/
34001          STFL  CMPS   A      CNTR
$$NOP,NOP,AND,S1,TAB
34001                AND    S1     TAB
$$/
34002          STFL  PASS   S11    S1
$$WRTE,MPCK,PASS,TAB,S1
34002   WRTE   MPCK  PASS   TAB    S1
$$/
34003          SRG1  CMPS          MEU
$$READ,RTN,INC,PNM,P
34003   READ   RTN   INC    PNM    P
$$A
$SH,34000B,34003B
34000   READ         PASS   L      A
34001                AND    S1     TAB
34002   WRTE   MPCK  PASS   TAB    S1
34003   READ   RTN   INC    PNM    P
$SE,A
A    = 0


A    = 0
A    = 377B
A    = 377
A    = A
```

EXAMPLE 2: READ/WRITE MEMORY, CONSOLE RUN SHEET (Sheet 2 of 2)

```
$PR
P+01= RETURN
P+02= RETURN
P+03= RETURN
P+04= RETURN
P+05= RETURN
P+06= RETURN
P+07= RETURN
P+08= RETURN
P+09= RETURN
P+10= RETURN


P+01= RETURN
P+01= 52525B
P+01= 52525
P+01= A
$RU,105600B
RETURN= P+02
$PR
P+01= 125
P+02= RETURN
P+03= RETURN
P+04= RETURN
P+05= RETURN
P+06= RETURN
P+07= RETURN
P+08= RETURN
P+09= RETURN
P+10= RETURN


P+01= 125
P+01= A
$EX
 $END MDEP
:EX
 $END FMGR
```

14-3. SHELL SORT EXAMPLE

This example illustrates a microprogrammed Shell sort technique which performs a sort of numeric data (assumed to be in a disc file). The theory of the technique is described in the reference material that is mentioned at the beginning of this section. The example illustrates the benefits of microprogramming a typical program that may be used repeatedly in a particular application. Included here are a FORTRAN program used to input the numbers to be sorted, list them (if so desired), and call a sort program. An Assembly language program is called to interface to a microprogram which performs the actual Shell sort.

Figure 14-1 is a flowchart that explains the microprogram. Annotated console run sheets are included that can be used to perform this same example in a step-by-step manner. The fully commented microprogram that performs the sort is included immediately after the console run sheets. Note that the Microdebug Editor is used to examine the progress of the sort.

When confidence in the ability of the microprogram to perform the sort is established, an application FORTRAN program is run (SRTST; which times the difference between the Assembler sort and the microprogrammed sort). The timing is accomplished in addition to the tasks already performed by the previously run test program.

The Assembly language program that runs the Shell sort (in competition with the microprogrammed version) is shown just before the console run sheet. Use the run sheet as an example to perform the execution and timing of the sort.

EXAMPLE 3: SHELL SORT, FORTRAN TEST PROGRAM

```
        PAGE  0001              FTN4 - RELEASE 24177C - JULY, 1972

0001  FTN4,L
0002        PROGRAM SRTST
0003        INTEGER P(5),CONS,PRINT,IDCB(144),NAME(3),IBUF(128)
0004        INTEGER TABLE(125)
0005        EQUIVALENCE (CONS,P(1)),(NMBR,P(2)),(PRINT,P(3))
0006        DATA NAME/2HN5,2H00,2H0 /
0007  C
0008  C GET RUN PARAMETERS
0009        CALL RMPAR(P)
0010  C
0011  C READ UNSORTED ELEMENTS FROM FILE N5000
0012        CALL OPEN (IDCB,IERR,NAME)
0013        DO 10 J=1,NMBR/125
0014        CALL READF (IDCB,IERR,IBUF)
0015        DO 20 I=1,125
0016  20    TABLE((J-1)*125 + I) = IBUF(I)
0017  10    CONTINUE
0018  C
0019  C LIST UNSORTED ELEMENTS ?
0020        IF (PRINT) 30,40,30
0021  30    WRITE (CONS,100) (TABLE(I),I=1,NMBR)
0022  100   FORMAT (/,(10@7))
0023  C
0024  C USE MDES TO INITIALIZE WCS
0025  40    CALL MDES (CONS)
0026  C
0027  C INDICATE START OF SORT
0028        WRITE (CONS,200)
0029  200   FORMAT (/," START OF SORT")
0030  C
0031  C EXECUTE SORT
0032        CALL SORT (NMBR, TABLE)
0033  C
0034  C INDICATE END OF SORT
0035        WRITE (CONS,300)
0036  300   FORMAT (/," END OF SORT")
0037  C
0038  C LIST SORTED ELEMENTS ?
0039        IF (PRINT) 50,60,50
0040  50    WRITE (CONS,100) (TABLE(I),I=1,NMBR)
0041  C
0042  C COMPLETE DEBUG OPERATIONS
0043  C I.E. CLEAR BREAKPOINTS, ETC.
0044  60    CALL MDES (CONS)
0045        CALL CLOSE (IDCB)
0046        END

**  NO ERRORS*      PROGRAM = 00587       COMMON = 00000
```

EXAMPLE 3: SHELL SORT, TEST ASSEMBLER INTERFACE

```
PAGE 0002 #01

0001                    ASMB.L
0002  00000                   NAM I2.1.7
0003*
0004*                       SORT INTERFACE PROGRAM
0005*
0006                          ENT SORT
0007                          EXT .ENTR
0008  00000 000000  NMBR  BSS 1
0009  00001 000000  TABLE BSS 1
0010  00002 000000  SORT  NOP
0011  00003 016001X       JSB .ENTR      GET PARAMETERS
0012  00004 000000R       DEF NMBR
0013*
0014  00005 162000R       LDA NMBR,I     A = NUMBER OF ELEMENTS
0015  00006 066001R       LDB TABLE      B = ADDRESS OF FIRST ELEMENT
0016  00007 000040        CLE            E = 0 = INITIAL ENTRY
0017  00010 105600        OCT 105600     INVOKE SORT MICROPROGRAM
0018*
0019  00011 126002R       JMP SORT,I
0020                      END
**  NO ERRORS *TOTAL **RTE ASMB 750420**
```

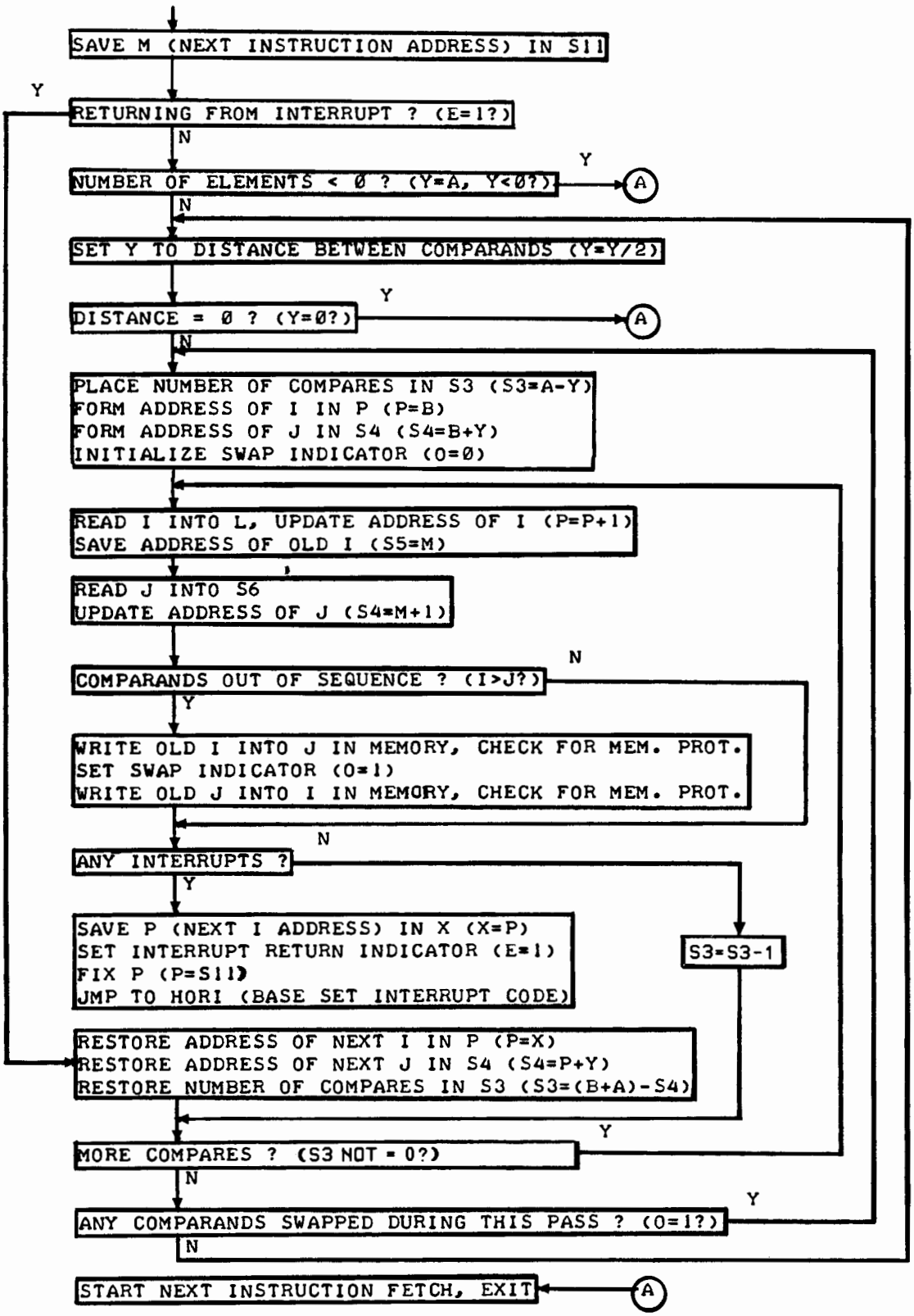


Figure 14-1. Example 3, Microprogrammed Shell Sort Flowchart

EXAMPLE 3: SHELL SORT; TEST, CONSOLE RUN SHEET (Sheet 1 of 2)

```
*ON,FMGR
:RU,EDITR                    CREATE MICROPROGRAM SOURCE FILE
SOURCE FILE?
/△
EOF
/T;10,15,20,25,30,40         SET TABS FOR MICROINSTRUCTION FORMAT
/ MICMXE,L;;;;;;21MX E-SERIES OR F-SERIES
/ $CODE='M2.1E,REPLACE;;;OBJECT TO DISC
                             USER SELECTED MICROPROGRAM OBJECT FILENAME
BODY OF
MICROPROGRAM

/ELC&M2.1E                   USER SELECTED MICROPROGRAM SOURCE FILENAME
  LS FILE 2 41
END OF EDIT
:RU,MICRO,2                  MICROASSEMBLE MICROPROGRAM
 /MICRO: END
:RU,SRTST,1,5,1              CONSOLE LU, NUMBER OF DATA, LIST FLAG (1=LIST)

 016440 136075 016336 152742 023501   UNSORTED DATA

COMPUTER TYPE:  1=21MX,2=21MX E-SERIES
TYPE(1 OR 2)?2
$LU,13

LU#       RANGE      STATUS
13  034000--034777   1
&LD,'M2.1E                   USE FILENAME IN $CODE STATEMENT
$LC,34600B,34417B            LOCATE MDE BREAKPOINT MICROPROGRAM, AND
$BR,34052B,34072B            PROVIDE AN UNUSED ENTRY POINT FOR MDE USE,
BREAK 1  34052               BEFORE SETTING BREAKPOINTS
BREAK 2  34072               SET BREAKPOINT IN SWAP MICROINSTRUCTIONS, AND
BREAK 3  0                   SET BREAKPOINT AT END OF ONE COMPLETE PASS
$EX

 START OF SORT
BREAK 34052                  BREAKPOINT IN SWAP MICROINSTRUCTIONS
$SE,L,S6
L    = 16440      S6   = 16336
                             ELEMENTS BEING SWAPPED

L    = 16440
L    = A
$RU
BREAK 34072                  AFTER BREAKING AT END OF PASS,
$CL,34072B                   REMOVE END OF PASS BREAKPOINT
BREAK 1  34052
BREAK 2  0
BREAK 3  0
$RU
```

EXAMPLE 3: SHELL SORT; TEST, CONSOLE RUN SHEET (Sheet 2 of 2)

```
BREAK 34052          <-- BREAKPOINT IN SWAP MICROINSTRUCTIONS
$SE,L,S6
L    = 16336      S6   = 136075
                  <-- ELEMENTS BEING SWAPPED

L    = 16336
L    = A
$RU
BREAK 34052          <-- BREAKPOINT IN SWAP MICROINSTRUCTIONS
$SE,L,S6
L    = 16440      S6   = 152742
                  <-- ELEMENTS BEING SWAPPED

L    = 16440
L    = A
$RU
BREAK 34052          <-- BREAKPOINT IN SWAP MICROINSTRUCTIONS
$SE,L,S6
L    = 16336      S6   = 152742
                  <-- ELEMENTS BEING SWAPPED

L    = 16336
L    = A
$RU

 END OF SORT         <-- NOTE: THESE ARE NEGATIVE NUMBERS

 136075 152742 016336 016440 023501   <-- CORRECTLY SORTED DATA
$CL                  <-- BE SURE TO REMOVE BREAKPOINTS !
BREAK 1  0
BREAK 2  0
BREAK 3  0
$EX
:EX
 $END FMGR
```

EXAMPLE 3: SHELL SORT, MICROPROGRAM (Sheet 1 of 3)

```
PAGE 0002 RTE MICRO-ASSEMBLER REV.A 760805

                          MICMXE,L
                          $CODE='M2.1E,REPLACE
                                   ORG  34000B
                          ******************************************************
                          *                                                    *
                          *                  LAB 2.1 MICROPROGRAM              *
                          *                                                    *
                          * THIS MICROPROGRAM SORTS AN INTEGER ARRAY INTO      *
                          * ASCENDING ORDER USING THE DIMINISHING INCREMENT    *
                          * TECHNIQUE (I.E. SHELL SORT).                       *
                          * REF: ART OF COMPUTER PROGRAMMING, VOL 3.           *
                          *                                                    *
                          * CALLING SEQUENCE                                   *
                          *      LDA NMBR     + NUMBER OF SORT ELEMENTS         *
                          *      LDB TABLE    ADDRESS OF FIRST ELEMENT          *
                          *      CLE          E=(0=INITIAL ENTRY,               *
                          *                      1=RETURN FROM INTERRUPT)       *
                          *      OCT 105600   INVOKE SORT MICROPROGRAM          *
                          *                                                    *
                          * AT END                                             *
                          *   CONTENTS OF TABLE SORTED                         *
                          *   A,B UNALTERED   E,O MAY BE ALTERED  X,Y ALTERED  *
                          *                                                    *
                          * NOTE                                               *
                          *   IN THE FOLLOWING COMMENTS, I AND J ARE THE TWO   *
                          *   SORT ELEMENTS BEING COMPARED                     *
                          *   (I.E. ARE THE COMPARANDS)                        *
                          *                                                    *
                          ******************************************************
                          HORI     EQU  6B
 34000  327 001007                JMP                 SORT      SAVE ENT POINTS
                                   ALGN
                          ******************************************
                          * SAVE M (NEXT INSTRUCTION ADDRESS) IN S11 *
                          ******************************************
 34020  010 033507        SORT                   S11  M         S11 = NEXT
                          ************************************     INSTR ADDR
                          * RETURNING FROM INTERRUPT ? (E=1?) *
                          ************************************
 34021  334 103042                JMP  CNDX E         INTRTN    YES, USE INTRTN
                          ****************************************
                          * NUMBER OF ELEMENTS < 0 ? (Y=A, Y<0?) *
                          ****************************************
 34022  010 007647                                Y    A         Y = A
 34023  327 103602                JMP  CNDX AL15      EXIT      Y<0 ? YES, EXIT
```

EXAMPLE 3: SHELL SORT, MICROPROGRAM (Sheet 2 of 3)

```
PAGE 0003 RTE MICRO-ASSEMBLER REV.A 760805

                          ************************************************
                          * SET Y TO DISTANCE BETWEEN COMPARANDS (Y=Y/2) *
                          ************************************************
 34024  010 073664        SETY          R1        Y    Y          Y = Y/2
                          ************************
                          * DISTANCE = 0 ? (Y=0?) *
                          ************************
 34025  010 072747                                 Y
                                JMP  CNDX ALZ      EXIT       Y=0 ? YES, EXIT
                          ****************************************
                          * PLACE NUMBER OF COMPARES IN S3 (S3=A-Y) *
                          * FORM ADDRESS OF I IN P (P=B)            *
                          * FORM ADDRESS OF J IN S4 (S4=B+Y)        *
                          * INITIALIZE SWAP INDICATOR (O=0)         *
                          ****************************************
 34027  010 072507        STRTPASS                L    Y
 34030  004 107107                     SUB  S3   A          S3 = COMPARES
 34031  010 011707                          P    B          P = ADDR OF I
 34032  003 011153                COV  ADD  S4   B          S4 = ADDR OF J,
                          *                                    O=0
                          ********************************************
                          * READ I INTO L, UPDATE ADDRESS OF I (P=P+1) *
                          * SAVE ADDRESS OF OLD I (S5=M)               *
                          ********************************************
 34033  227 174707        COMPARE  READ       INC  PNM  P          READ I, UPDATE P
 34034  010 033207                          S5   M          S5 = ADDR OF I
 34035  010 000507                          L    TAB        L = I
                          ******************************
                          * READ J INTO S6             *
                          * UPDATE ADDRESS OF J (S4=M+1) *
                          ******************************
 34036  230 046647                READ            M    S4         READ J
 34037  007 133147                     INC  S4   M          S4 = NEXT J ADDR
 34040  010 001247                          S6   TAB        S6 = J
                          ***************************************
                          * COMPARANDS OUT OF SEQUENCE ? (I>J?) *
                          ***************************************
 34041  014 152747                     XOR       S6         J SIGN = I SIGN?
 34042  327 142302                JMP  CNDX AL15 RJS  SUBTRACT    YES, SUBTRACT
 34043  012 136747                     PASL                    I SIGN = - ?
 34044  327 102602                JMP  CNDX AL15      INTCHK      YES, NO SWAP
 34045  327 002407                JMP                 SWAP        NO, SWAP
 34046  004 152747        SUBTRACT            SUB       S6         J - I < 0 ?
 34047  327 142602                JMP  CNDX AL15 RJS  INTCHK      NO, NO SWAP
                          ******************************************************
                          * WRITE OLD I INTO J IN MEMORY, CHECK FOR MEM. PROT. *
                          * SET SWAP INDICATOR (O=1)                           *
                          * WRITE OLD J INTO I IN MEMORY, CHECK FOR MEM. PROT. *
                          ******************************************************
 34050  012 137307        SWAP                PASL S7              S7 = OLD I
 34051  210 054036                WRTE MPCK       TAB  S7         J IN MEM = OLD I
 34052  007 150654                     SOV  INC  M    S5         M=ADDR OF I, O=1
 34053  210 052036                WRTE MPCK       TAB  S6         I IN MEM = OLD J
                          *                                        O=1
```

EXAMPLE 3: SHELL SORT, MICROPROGRAM (Sheet 3 of 3)

```
PAGE 0004 RTE MICRO-ASSEMBLER REV.A 760818

                          ********************
                          * ANY INTERRUPTS ? *
                          ********************
34054  323 143502  INTCHK   JMP  CNDX HOI  RJS  ENDCHK      NO, CHK PASS
                          *******************************************
                          * SAVE P (NEXT I ADDRESS) IN X (X=P)      *
                          * SET INTERRUPT RETURN INDICATOR (E=1)    *
                          * FIX P (P=S11)                           *
                          * JMP TO HORI (BASE SET INTERRUPT CODE)   *
                          *******************************************
34055  010 075607  INTEXIT                  X    P          X = NEXT I ADDR
34056  342 000607           IMM       LOW  IRCM 200B       IR(9-6)=1110=ELA
34057  011 136761                SRG1 ONE                    I.E. SET E
34060  010 065707                       P    S11        FIX P,
34061  320 000307           JMP                 HORI       JMP TO BASE SET
                          *                                  INTERRUPT CODE
                          ************************************************
                          * RESTORE ADDRESS OF NEXT I IN P (P=X)         *
                          * RESTORE ADDRESS OF NEXT J IN S4 (S4=P+Y)     *
                          * RESTORE NUMBER OF COMPARES IN S3 (S3=B+A-S4) *
                          ************************************************
34062  010 071707  INTRTN                  P    X          P = NEXT I ADDR
34063  010 072507                          L    Y
34064  003 075147                     ADD  S4   P          S4 = NEXT J ADDR
34065  010 006507                          L    A
34066  003 011107                     ADD  S3   B          S3 = B+A
34067  010 046507                          L    S4
34070  004 145107                     SUB  S3   S3         S3 = (B+A)-S4 =
34071  327 003547           JMP                 *+2          COMPARES
                          ****************************************
                          * MORE COMPARES ? (S3=S3-1, S3 NOT =0?) *
                          ****************************************
34072  000 045107  ENDCHK              DEC  S3   S3         MORE COMPARES ?
34073  320 041542           JMP  CNDX ALZ  RJS  COMPARE      YES, DO NEXT
                          ****************************************************
                          * ANY COMPARANDS SWAPPED DURING THIS PASS ? (O=1?) *
                          ****************************************************
34074  335 101342           JMP  CNDX OVFL      STRTPASS     YES, REDO PASS
34075  327 001207           JMP                 SETY         NO, NEXT PASS
                          ***************************************
                          * START NEXT INSTRUCTION FETCH, EXIT *
                          ***************************************
34076  227 164700  EXIT     READ RTN  INC  PNM  S11        START NEXT
                          END                                INSTR FETCH
END OF PASS 2: NO ERRORS
```

EXAMPLE 3: SHELL SORT, APPLICATION PROGRAM

```
        PAGE  0001              FTN4 - RELEASE 24177C - JULY, 1972

0001  FTN4,L
0002        PROGRAM SRTST
0003        INTEGER P(5),CONS,PRINT,IDCB(144),NAME(3),IBUF(128)
0004        INTEGER TABLE(125)
0005        EQUIVALENCE (CONS,P(1)),(NMBR,P(2)),(PRINT,P(3))
0006        DATA NAME/2HN5,2H00,2H0 /
0007  C
0008  C GET RUN PARAMETERS
0009        CALL RMPAR(P)
0010  C
0011  C READ UNSORTED ELEMENTS FROM FILE N5000
0012        CALL OPEN (IDCB,IERR,NAME)
0013        DO 10 J=1,NMBR/125
0014        CALL READF (IDCB,IERR,IBUF)
0015        DO 20 I=1,125
0016  20    TABLE((J-1)*125 + I) = IBUF(I)
0017  10    CONTINUE
0018  C
0019  C LIST UNSORTED ELEMENTS ?
0020        IF (PRINT) 30,40,30
0021  30    WRITE (CONS,100) (TABLE(I),I=1,NMBR)
0022  100   FORMAT (/,(10@7))
0023  C
0024  C USE MDES TO INITIALIZE WCS
0025  40    CALL MDES (CONS)
0026  C
0027  C INDICATE START OF SORT
0028        WRITE (CONS,200)
0029  200   FORMAT (/," START OF SORT")
0030  C
0031  C EXECUTE SORT
0032        CALL SORT (NMBR, TABLE)
0033  C
0034  C INDICATE END OF SORT
0035        WRITE (CONS,300)
0036  300   FORMAT (/," END OF SORT")
0037  C
0038  C LIST SORTED ELEMENTS ?
0039        IF (PRINT) 50,60,50
0040  50    WRITE (CONS,100) (TABLE(I),I=1,NMBR)
0041  C
0042  C COMPLETE DEBUG OPERATIONS
0043  C I.E. CLEAR BREAKPOINTS, ETC.
0044  60    CALL MDES (CONS)
0045        CALL CLOSE (IDCB)
0046        END

**  NO ERRORS*     PROGRAM = 00587      COMMON = 00000
```

EXAMPLE 3: SHELL SORT, ASSEMBLER SORT (Sheet 1 of 2)

```
PAGE 0002 #01

0001                    ASMB,L
0002  00000                   NAM ASORT,7
0003*****************************************************
0004*                                                   *
0005*                    LAB 2.2 ASSEMBLER SORT         *
0006*                                                   *
0007* THIS ASSEMBLER PROGRAM SORTS AN INTEGER ARRAY INTO *
0008* ASCENDING ORDER USING THE DIMINISHING INCREMENT    *
0009* TECHNIQUE (I.E. SHELL SORT).                       *
0010* REF: ART OF COMPUTER PROGRAMMING, VOL 3.           *
0011*                                                    *
0012* CALLING SEQUENCE                                   *
0013*      LDA NMBR     + NUMBER OF SORT ELEMENTS        *
0014*      LDB TABLE    ADDRESS OF FIRST ELEMENT         *
0015*      CLE          NOT REQUIRED FOR THIS PROGRAM,   *
0016*                     INCLUDED FOR COMPATIBILITY WITH *
0017*                     THE MICROPROGRAM CALL          *
0018*      JSB SORT     INVOKE SORT ASSEMBLER PROGRAM    *
0019*                                                    *
0020* AT END                                             *
0021*   CONTENTS OF TABLE SORTED                         *
0022*   O MAY BE ALTERED   A,B,X,Y,E ALTERED             *
0023*                                                    *
0024* NOTE                                               *
0025*   IN THE FOLLOWING COMMENTS, I AND J ARE THE TWO   *
0026*   SORT ELEMENTS BEING COMPARED                     *
0027*   (I.E. ARE THE COMPARANDS)                        *
0028*                                                    *
0029*****************************************************
0030                          ENT SORT
0031                          EXT .ENTR
```

EXAMPLE 3: SHELL SORT, ASSEMBLER SORT (Sheet 2 of 2)

```
PAGE 0003 #01

0033  00000 000000  NMBR  BSS 1
0034  00001 000000  TABLE BSS 1
0035  00002 000000  SORT  NOP
0036  00003 016001X       JSB .ENTR     GET PARAMETERS
0037  00004 000000R       DEF NMBR
0038  00005 162000R       LDA NMBR,I    A = NUMBER OF ELEMENTS
0039  00006 002020        SSA           A < 0 ?
0040  00007 126002R       JMP SORT,I      YES, EXIT
0041  00010 001100  SETY  ARS           "Y" = "Y"/2 (SEE SORT MICROPROGRAM)
0042  00011 002003        SZA,RSS       "Y" = 0 ?
0043  00012 126002R       JMP SORT,I      YES, SORT DONE, EXIT
0044  00013 072057R       STA DSTNC     DSTNC = "Y" = DISTANCE BETWEEN "I" & "J"
0045  00014 103101  STRTP CLO           CLEAR SWAP INDICATOR
0046  00015 166000R       LDB NMBR,I    SET
0047  00016 007004        CMB,INB         CNTR
0048  00017 046057R       ADB DSTNC         TO NUMBER
0049  00020 076060R       STB CNTR            OF COMPARES
0050  00021 066001R       LDB TABLE
0051  00022 076061R       STB IPTR      IPTR = ADDRESS OF "I"
0052  00023 046057R       ADB DSTNC
0053  00024 076062R       STB JPTR      JPTR = ADDRESS OF "J"
0054  00025 162061R COMPR LDA IPTR,I
0055  00026 122062R       XOR JPTR,I    A = "I" XOR "J"
0056  00027 002021        SSA,RSS       SAME SIGNS ?
0057  00030 026035R       JMP SUB         YES, SUBTRACT
0058  00031 162061R       LDA IPTR,I
0059  00032 002020        SSA           "I" < 0 ?
0060  00033 026047R       JMP ENDCH       YES, DON'T SWAP
0061  00034 026042R       JMP SWAP        NO, SWAP
0062  00035 162061R SUB   LDA IPTR,I
0063  00036 003004        CMA,INA
0064  00037 142062R       ADA JPTR,I    A = "J" - "I"
0065  00040 002021        SSA,RSS       "I" > "J" ?
0066  00041 026047R       JMP ENDCH       NO, DON'T SWAP
0067  00042 102101  SWAP  STO           SET OVFL TO INDICATE A SWAP
0068  00043 162061R       LDA IPTR,I    SWAP
0069  00044 166062R       LDB JPTR,I      "I"
0070  00045 172062R       STA JPTR,I        AND
0071  00046 176061R       STB IPTR,I          "J"
0072  00047 036061R ENDCH ISZ IPTR      UPDATE "I" ADDRESS,
0073  00050 036062R       ISZ JPTR             "J" ADDRESS, AND
0074  00051 036060R       ISZ CNTR             CNTR. CNTR = 0 ?
0075  00052 026025R       JMP COMPR       NO, DO NEXT COMPARE
0076  00053 102201        SOC           ANY SWAPS THIS PASS ?
0077  00054 026014R       JMP STRTP       YES, REPEAT PASS
0078  00055 062057R       LDA DSTNC       NO, A = "Y",
0079  00056 026010R       JMP SETY            START NEW PASS
0080  00057 000000  DSTNC BSS 1
0081  00060 000000  CNTR  BSS 1
0082  00061 000000  IPTR  BSS 1
0083  00062 000000  JPTR  BSS 1
0084                     END
**  NO ERRORS *TOTAL **RTE ASMB 750420**
```

EXAMPLE 3: SHELL SORT, APPLICATION/TIMING CONSOLE RUN SHEET

```
*ON,FMGR
:RU,ASORT,1,5000  <---- RUN ASSEMBLY LANGUAGE SORT
                  <---- CONSOLE LU, NUMBER OF SORT ELEMENTS
 START OF SORT

 END OF SORT

                HOURS    MINUTES    SECONDS
 STOP  :          10         39      34.76
 START :          10         39      22.92  <---- RUN TIME = 11.84 SECONDS

:RU,MDEP  <---- LOAD WCS WITH SORT MICROPROGRAM

COMPUTER TYPE:   1=21MX,2=21MX E-SERIES
TYPE(1 OR 2)?2
$LU,13

LU#       RANGE     STATUS
13  034000--034777   1
$LD,'M2.1E  <---- USE FILENAME IN $CODE STATEMENT
$EX
 $END MDEP
:RU,MSORT,1,5000  <---- RUN MICROPROGRAMMED SORT
                  <---- CONSOLE LU, NUMBER OF SORT ELEMENTS
 START OF SORT

 END OF SORT

                HOURS    MINUTES    SECONDS
 STOP  :          10         41      15.87
 START :          10         41      14.45  <---- RUN TIME = 1.42 SECONDS!

:EX
 $END FMGR
```

14-4. MICROPROGRAMMED I/O OPERATION EXAMPLE

This paragraph contains an example of properly microprogrammed I/O operation in the RTE system environment. An Assembly language privileged section driver (DVA*xx*) is shown as it would appear "normally", then the microprogram enhanced driver (DVM*xx*) is shown. The FORTRAN IV program, shown first is used for executing the privileged I/O operation. The console run sheet and microprogram are included in the final part of this example.

```
        PAGE  0001              FTN4 - RELEASE 24177C - JULY, 1972

0001  FTN,L
0002        PROGRAM MPIO
0003        INTEGER IBUFR(5),P(5),CONS
0004        EQUIVALENCE (P(1),CONS),(P(2),LU)
0005        DATA IBUFL/5/
0006  C
0007  C GET CONSOLE LU, INPUT DEVICE LU
0008        CALL RMPAR (P)
0009  C
0010  C PERFORM INPUT FROM DEVICE
0011        CALL REIO (1,LU,IBUFR,IBUFL)
0012  C
0013  C DISPLAY INPUT DATA
0014        WRITE (CONS,100) IBUFR
0015  100   FORMAT (/,X,5A2,/)
0016        END

**  NO ERRORS*      PROGRAM = 00048       COMMON = 00000
```

The FORTRAN program used is the same whether the "normal" driver or enhanced version is used. The driver sections (initiation, privileged, completion) are prepared according to the guidelines in the *Real Time Executive III Software System Programming and Operating Manual,* part no. 92060-90004. Notice that the privileged section of the microprogram enhanced driver (the part that is microprogrammed) is much shorter than the complete Assembly language driver, thus, saving main memory space. The entire "old" privileged section is not needed with the new version. Now, from location PM*xx* you proceed immediately to the microprogram. This modified part of the driver saves the environment, inputs data, and is used when returning from control memory to restore the environment. Comments on the operation of the driver are included right in the listings.

Figure 14-2 is the flowchart for the microprogram. The console run sheet for microprogram preparation and the microprogram called from PM*xx* in the driver are shown last. Note that the microprogram saves the DMS status. The microprogram must be sensitive to DMS to operate properly in an RTE III system. SSM and JRS in the microprogram are DMS instructions. The EQU statements point branch instructions to these microroutines outside this microprogram. Note that Memory Protect status is checked and DMS status is properly restored on exit. This is an example of how to properly interface with the RTE system.

EXAMPLE 4: UNMODIFIED PRIVILEGED DRIVER (Sheet 1 of 3)

```
PAGE 0002 #01

0001                    ASMB,L
0002*
0003* SAMPLE PRIVILEGED DRIVER
0004*
0005* AN "*" IN COLUMN 19 INDICATES A STATEMENT THAT IS NOT
0006* REQUIRED FOR THE MICROPROGRAM ENHANCED VERSION (DVMXX)
0007* OF THIS SAMPLE PRIVILEGED DRIVER
0008*
0009  00000                    NAM DVAXX,0
0010                           ENT IAXX,PAXX,CAXX
0011                           SUP
0012*
0013*
0014* INITIATION SECTION
0015*
0016  00000 000000  IA07  NOP
0017  00001 072167R       STA SCODE      SAVE SELECT CODE
0018  00002 161665        LDA EQT6,I     GET CONWD
0019  00003 012200R       AND =B77       ISOLATE REQUEST CODE
0020  00004 052201R       CPA =B1        READ REQUEST ?
0021  00005 026007R       JMP BFCHK        YES, CONTINUE
0022  00006 026015R       JMP REJCT        NO, REJECT I/O REQUEST
0023  00007 161665  BFCHK LDA EQT6,I     GET CONWD
0024  00010 012202R       AND =B37777    ISOLATE BITS 15,14
0025  00011 052201R       CPA =B1        BUFFERED I/O ?
0026  00012 026017R       JMP RQOK         YES, DO I/O
0027  00013 052203R       CPA =B3        CLASS I/O ?
0028  00014 026017R       JMP RQOK         YES, DO, I/O
0029  00015 002404  REJCT CLA,INA          NO, ERROR
0030  00016 126000R       JMP IAXX,I     TAKE REJECT RETURN
0031  00017 062167R RQOK  LDA SCODE      A = SELECT CODE (SC)
0032  00020 032170R       IOR CLC       *CONFIGURE PRIVILEGED
0033  00021 072103R       STA PRCLC     *  SECTION CLC
0034  00022 062167R       LDA SCODE      CONFIGURE STC'S
0035  00023 032171R       IOR STC          IN
0036  00024 072045R       STA INSTC          INITIATION SECTION
0037  00025 072113R       STA PRSTC     *       & PRIVILEGED SECTION
0038  00026 022204R       XOR =B1200    *CHANGE TO LIA SC
0039  00027 072075R       STA PRLIA     *CONFIGURE PRIVILEGED SECTION LIA
0040  00030 161663        LDA EQT4,I     CLEAR EQT4
0041  00031 012205R       AND =B167777     BIT 12 TO ALLOW
0042  00032 171663        STA EQT4,I         NORMAL TIMEOUT
0043  00033 061774        LDA EQT15      SAVE
0044  00034 072160R       STA EQ15         EQT15
0045  00035 061663        LDA EQT4           & EQT4
0046  00036 072161R       STA EQ4              ADDRESSES
0047  00037 161667        LDA EQT8,I     GET DATA COUNT
0048  00040 002021        SSA,RSS        NEGATIVE ?
0049  00041 003004        CMA,INA          NO, SET NEGATIVE
0050  00042 072157R       STA COUNT
0051  00043 161666        LDA EQT7,I     SAVE
0052  00044 072156R       STA BUFAD        BUFFER ADDRESS
0053  00045 103700  INSTC STC 0,C        START DEVICE
0054  00046 002400        CLA            INDICATE OK INITIATION
0055  00047 126000R       JMP IAXX,I     RETURN
```

EXAMPLE 4: UNMODIFIED PRIVILEGED DRIVER (Sheet 2 of 3)

```
PAGE 0003 #01


0057*
0058*
0059* PRIVILEGED SECTION
0060*
0061  00050 000000  PAXX   NOP
0062  00051 103100         CLF 0          TURN OFF INTERRUPTS
0063  00052 106706         CLC 6          TURN OFF
0064  00053 106707         CLC 7            DCPC INTERRUPTS
0065  00054 072164R        STA ASV        SAVE A,
0066  00055 076165R        STB BSV             B,
0067  00056 001520         ERA,ALS             E,
0068  00057 102201         SOC
0069  00060 002004         INA
0070  00061 072166R        STA EOSV            O,
0071  00062 105743         STX XSV             X, &
0072  00064 105753         STY YSV             Y REGISTERS
0073*      SSM DMSTS      SAVE DMS STATUS !!  OMIT FOR RTE 2 !!
0074  00066 061770         LDA MPTFL      SAVE MEMORY PROTECT
0075  00067 072171R        STA MPTSV        FLAG
0076  00070 002404         CLA,INA        TURN OFF MEMORY
0077  00071 071770         STA MPTFL        FLAG
0078  00072 102100         STF 0          TURN ON INTERRUPTS
0079  00073 102500  PRLIA  LIA 0          GET DATA FROM I/O CARD
0080  00074 172150R        STA BUFAD,I    STORE DATA IN BUFFER
0081  00075 036150R        ISZ BUFAD      UPDATE BUFFER ADDRESS
0082  00076 036151R        ISZ COUNT      LAST DATA ?
0083  00077 026110R        JMP CLF0         NO, PREPARE FOR NEXT INPUT
0084  00100 103100         CLF 0          TURN OFF INTERRUPTS
0085  00101 106700  PRCLC  CLC 0          TURN OFF DEVICE
0086  00102 003400         CCA            SET TIMEOUT FOR
0087  00103 172152R        STA EQ15,I       ONE TICK & SET
0088  00104 162153R        LDA EQ4,I        BIT 12 IN EQT4 SO
0089  00105 032200R        IOR =B10000      RTIOC WILL CALL
0090  00106 172153R        STA EQ4,I        CA07 ON TIMEOUT
0091  00107 026112R        JMP EXIT
0092  00110 103100  CLF0   CLF 0          TURN OFF INTERRUPTS
0093  00111 103700  PRSTC  STC 0,C        ACTIVATE DEVICE FOR NEXT INPUT
0094  00112 062171R EXIT   LDA MPTSV      WAS MEMORY
0095  00113 002002         SZA              PROTECT ON ?
0096  00114 026125R        JMP EXIT1          NO, FORGET DCPC'S
0097  00115 065654         LDB INTBA      TURN
0098  00116 160001         LDA 1,I          DCPC'S
0099  00117 002020         SSA                 BACK
0100  00120 102706         STC 6                 ON
0101  00121 006004         INB                     IF
0102  00122 160001         LDA 1,I                  THEY
0103  00123 002020         SSA                         WERE
0104  00124 102707         STC 7                         ON
0105  00125 105755  EXIT1  LDY YSV        RESTORE Y,
0106  00127 105745         LDX XSV                 X,
0107  00131 103101         CLO                     O,
0108  00132 000036         SLA,ELA                 E, &
0109  00133 102101         STO
0110  00134 066165R        LDB BSV                 B REGISTERS
0111  00135 062171R        LDA MPTSV      RESTORE MEMORY
0112  00136 071770         STA MPTFL        PROTECT FLAG
```

EXAMPLE 4: UNMODIFIED PRIVILEGED DRIVER (Sheet 3 of 3)

```
PAGE 0004 #01

0113  00141 002002          SZA              WAS MEMORY PROTECT ON ?
0114  00142 026151R         JMP EXIT2          NO, LEAVE OFF
0115  00143 062172R         LDA ASV            YES, RESTORE A REGISTER
0116  00144 105715          JRS DMSTS EX1 RESTORE DMS STATUS
0117  00147 102100  EX1     STF 0            TURN ON INTERRUPT SYSTEM
0118  00150 126050R         JMP PAXX,I       EXIT
0119  00151 062172R EXIT2   LDA ASV          RESTORE A REGISTER
0120  00152 102100          STF 0            TURN ON INTERRUPT SYSTEM
0121  00153 105715          JRS DMSTS PAXX,I RESTORE DMS STATUS & RETURN
0122*
0123  00156 000000  BUFAD   BSS 1
0124  00157 000000  COUNT   BSS 1
0125  00160 000000  EQ15    BSS 1
0126  00161 000000  EQ4     BSS 1
0127  00162 000000  DMSTS   BSS 1
0128*
0129* END PRIVILEGED SECTION
0130*
0131*
0132*
0133* COMPLETION SECTION
0134  00163 000000  CAXX    NOP
0135  00164 002400          CLA              SET UP FOR NORMAL RETURN
0136  00165 165667          LDB EQT8,I       TRANSMISSION LOG TO B
0137  00166 126163R         JMP CAXX,I       RETURN
0138*
0139  00167 000000  SCODE   NOP
0140  00170 106700  CLC     CLC 0          *
0141  00171 103700  STC     STC 0,C
0142  00172 000000  ASV     BSS 1          *
0143  00173 000000  BSV     BSS 1          *
0144  00174 000000  EOSV    BSS 1          *
0145  00175 000000  XSV     BSS 1          *
0146  00176 000000  YSV     BSS 1          *
0147  00177 000000  MPTSV   BSS 1          *
0148*
0149*
0150* SYSTEM COMMUNICATION AREA
0151*
0152  01650           .     EQU 1650B
0153  01654           INTBA EQU .+4B
0154  01663           EQT4  EQU .+13B
0155  01665           EQT6  EQU .+15B
0156  01666           EQT7  EQU .+16B
0157  01667           EQT8  EQU .+17B
0158  01774           EQT15 EQU .+124B
0159  01770           MPTFL EQU .+120B
0160                        END
**  NO ERRORS *TOTAL **RTE ASMB 750420**
```

EXAMPLE 4: ENHANCED DRIVER (Sheet 1 of 2)

```
PAGE 0002 #01

0001                    ASMB,L
0002*
0003* SAMPLE PRIVILEGED DRIVER WITH MICROPROGRAM ENHANCEMENTS
0004*
0005  00000                    NAM  DVMXX,0
0006                           ENT  IMXX,PMXX,CMXX
0007                           SUP
0008*
0009*
0010* INITIATION SECTION
0011*
0012  00000 000000  IM07  NOP
0013  00001 072061R        STA SCODE      SAVE SELECT CODE
0014  00002 161665        LDA EQT6,I     GET CONWD
0015  00003 012063R        AND =B77       ISOLATE REQUEST CODE
0016  00004 052064R        CPA =B1        READ REQUEST ?
0017  00005 026007R        JMP BFCHK        YES, CONTINUE
0018  00006 026015R        JMP REJCT        NO, REJECT I/O REQUEST
0019  00007 161665  BFCHK LDA EQT6,I     GET CONWD
0020  00010 012065R        AND =B37777    ISOLATE BITS 15,14
0021  00011 052064R        CPA =B1        BUFFERED I/O ?
0022  00012 026017R        JMP RQOK         YES, DO I/O
0023  00013 052066R        CPA =B3        CLASS I/O ?
0024  00014 026017R        JMP RQOK       YES, DO I/O
0025  00015 002404  REJCT CLA,INA          NO, ERROR
0026  00016 126000R        JMP IMXX,I     TAKE REJECT RETURN
0027  00017 062061R RQOK  LDA SCODE      A = SELECT CODE (SC)
0028  00020 032062R        IOR STC        CONFIGURE STC IN
0029  00021 072037R        STA INSTC          INITIATION SECTION
0030  00022 161663        LDA EQT4,I     CLEAR EQT4
0031  00023 012067R        AND =B167777     BIT 12 TO ALLOW
0032  00024 171663        STA EQT4,I         NORMAL TIMEOUT
0033  00025 061774        LDA EQT15      SAVE
0034  00026 072052R        STA EQ15         EQT15
0035  00027 061663        LDA EQT4           & EQT4
0036  00030 072053R        STA EQ4              ADDRESSES
0037  00031 161667        LDA EQT8,I     GET DATA COUNT
0038  00032 002021        SSA,RSS        NEGATIVE ?
0039  00033 003004        CMA,INA          NO, SET NEGATIVE
0040  00034 072046R        STA COUNT
0041  00035 161666        LDA EQT7,I     SAVE
0042  00036 072045R        STA BUFAD        BUFFER ADDRESS
0043  00037 103700  INSTC STC 0,C        START DEVICE
0044  00040 002400        CLA            INDICATE OK INITIATION
0045  00041 126000R        JMP IMXX,I     RETURN
```

EXAMPLE 4: ENHANCED DRIVER (Sheet 2 of 2)

```
PAGE 0003 #01


0047*
0048*
0049* PRIVILEGED SECTION
0050*
0051  00042 000000  PMXX   NOP
0052                       MIC MIO,105600B,0 EQUATE MIO & MICROPROGRAM
0053  00043 105600         MIO              INVOKE MICROPROGRAM
0054  00044 000054R        DEF DMSTS        ADDRESS OF DMS STATUS SAVE WORD
0055  00045 000000  BUFAD BSS 1             BUFFER ADDRESS
0056  00046 000000  COUNT BSS 1             DATA COUNT
0057  00047 001770         DEF MPTFL        ADDRESS OF MEMORY PROTECT FLAG
0058  00050 000054R        DEF DMSTS        THESE 2 DEF'S ARE HERE SO THAT
0059  00051 100042R        DEF PMXX,I       MIH MAY INVOKE JRS EFFICIENTLY
0060  00052 000000  EQ15  BSS 1             ADDRESS OF EQT15
0061  00053 000000  EQ4   BSS 1             ADDRESS OF EQT4
0062  00054 000000  DMSTS BSS 1             DMS STATUS WORD
0063*
0064* END PRIVILEGED SECTION
0065*
0066*
0067*
0068* COMPLETION SECTION
0069  00055 000000  CM07  NOP
0070  00056 002400         CLA              SET UP FOR NORMAL RETURN
0071  00057 165667         LDB EQT8,I       TRANSMISSION LOG TO B
0072  00060 126055R        JMP CMXX,I       RETURN
0073*
0074  00061 000000  SCODE NOP
0075  00062 103700  STC   STC 0,C
0076*
0077*
0078* SYSTEM COMMUNICATION AREA
0079*
0080  01650          .     EQU 1650B
0081  01654          INTBA EQU .+4B
0082  01663          EQT4  EQU .+13B
0083  01665          EQT6  EQU .+15B
0084  01666          EQT7  EQU .+16B
0085  01667          EQT8  EQU .+17B
0086  01774          EQT15 EQU .+124B
0087  01770          MPTFL EQU .+120B
0088                       END
**  NO ERRORS *TOTAL **RTE ASMB 750420**
```

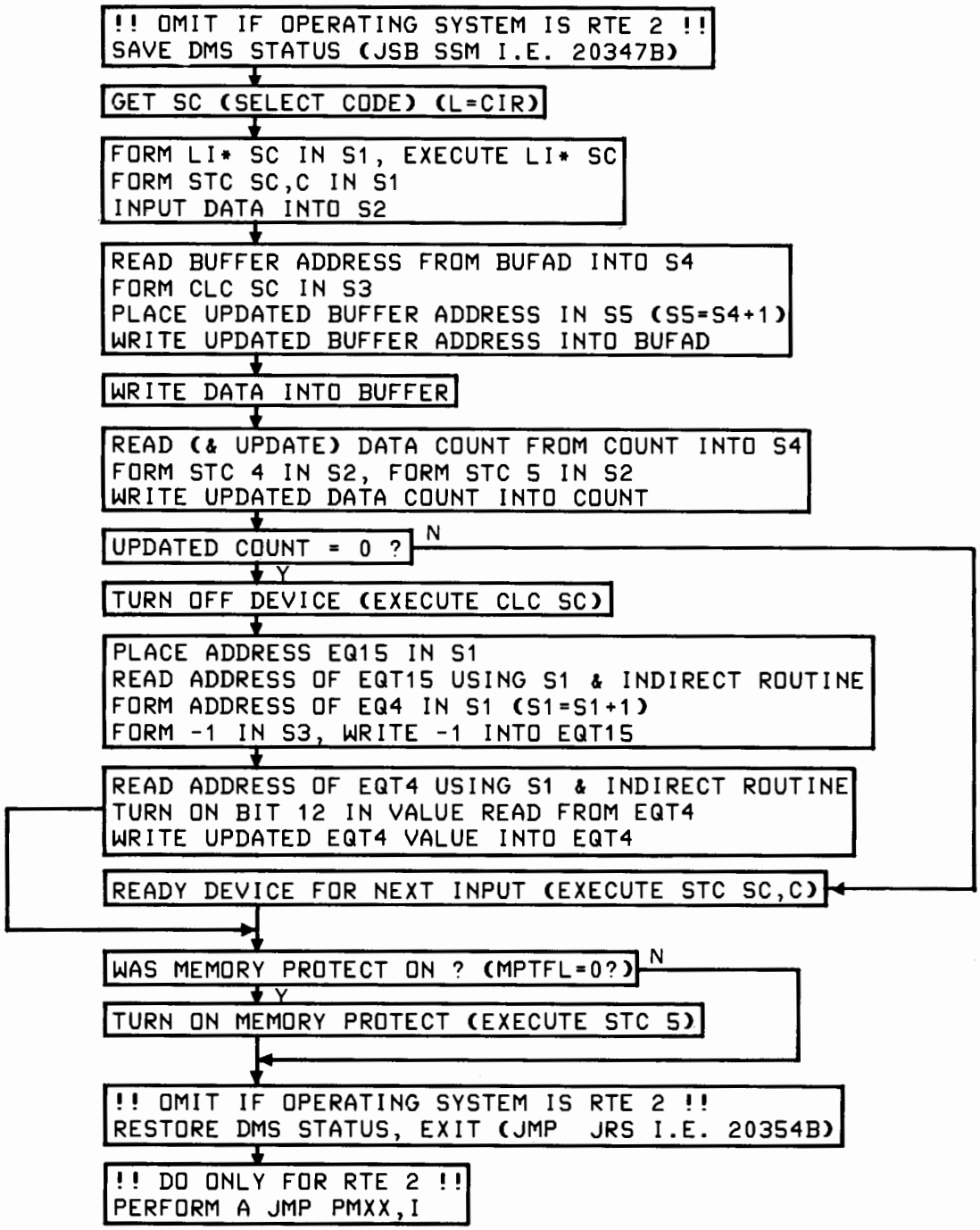


Figure 14-2. Example 4, Microprogrammed Privileged Section Flowchart

EXAMPLE 4: MICROPROGRAMMED DRIVER, CONSOLE RUN SHEET

```
*ON,FMGR
:RU,EDITR                    CREATE MICROPROGRAM SOURCE FILE
SOURCE FILE?
/Δ
EOF
/T;10,15,20,25,30,40         SET TABS FOR MICROINSTRUCTION FORMAT
/ MICMXE,L;;;;;;21MX E-SERIES OR F-SERIES
/ $CODE='M3.1E,REPLACE;;;OBJECT TO DISC
                             USER SELECTED MICROPROGRAM OBJECT FILENAME
  BODY OF
  MICROPROGRAM

/ELC&M3.1E                   USER SELECTED MICROPROGRAM SOURCE FILENAME
  LS FILE 2 33
END OF EDIT
:RU,MICRO,2                  MICROASSEMBLE MICROPROGRAM
 /MICRO: END
:RU,MDEP                     LOAD MICROPROGRAM INTO WCS

COMPUTER TYPE:   1=21MX,2=21MX E-SERIES
TYPE(1 OR 2)?2
$LU,13

LU#        RANGE      STATUS
13   034000--034777    1
$LD,'M3.1E                   FILENAME SPECIFIED IN $CODE STATEMENT
$EX
 $END MDEP                   INPUT DEVICE LU
:RU,MPIO,1,5
                             CONSOLE LU
 GAHDB
                             DATA
:EX
 $END FMGR
```

EXAMPLE 4: DRIVER MICROPROGRAMMED PRIVILEGED SECTION (Sheet 1 of 3)

```
PAGE 0002 RTE MICRO-ASSEMBLER REV.A 760805

                          MICMXE,L                                    21MX E/F-SERIES
                          $CODE='MDRVR,REPLACE                        OBJECT TO DISC
                          ***********************************************
                          *                                             *
                          * SAMPLE PRIVILEGED SECTION MICROPROGRAM FOR DVMXX *
                          *                                             *
                          ***********************************************
                                   ORG  34000B                        105600 => 34000
                          HORI     EQU  6B
                          INDIRECT EQU  251B
                          SSM      EQU  20347B
                          JRS      EQU  20354B
 34000  327 001007              JMP                  34020B     SAVE ENTRY
                                   ALGN                              POINTS
                          ****************************************
                          * !! OMIT IF OPERATING SYSTEM IS RTE 2 !! *
                          * SAVE DMS STATUS (JSB SSM I.E. 20347B)   *
                          ****************************************
 34020  230 036747              READ                          SSM EXPECTS A
 34021  304 016347              JSB                  SSM        READ OF DMSTS
 34022  000 075707                        DEC  P    P          SSM INC'S P 1
                          ******************************        TOO MANY FOR US
                          * GET SC (SELECT CODE)  (L=CIR) *
                          ******************************
 34023  010 024507                        L    CIR        L = SELECT CODE
                          ********************************
                          * FORM LI* SC IN S1, EXECUTE LI* SC *
                          * FORM STC SC,C IN S1              *
                          * INPUT DATA INTO S2               *
                          ********************************
 34024  357 175007              IMM       CMHI S1   376B       S1 = 400 = LI* 0
 34025  010 141007                        IOR  S1   S1         S1 = LI* SC
 34026  010 040606                   IOG       IRCM S1         EXECUTE LI* SC
 34027  353 007023              IMM  L4   CMLO S1   303B       S1=1700=STC 0,C
 34030  010 141007                        IOR  S1   S1         S1 = STC SC,C
 34031  010 013047                             S2   IOI        S2 = DATA
                          ***********************************************
                          * READ BUFFER ADDRESS FROM BUFAD INTO S4       *
                          * FORM CLC SC IN S3                            *
                          * PLACE UPDATED BUFFER ADDRESS IN S5 (S5=S4+1) *
                          * WRITE UPDATED BUFFER ADDRESS INTO BUFAD      *
                          ***********************************************
 34032  227 174707              READ      INC  PNM  P          READ BUF ADDR
 34033  351 107123              IMM  L4   CMLO S3   143B       S3=4700=CLC 0
 34034  010 145107                        IOR  S3   S3         S3 = CLC SC
 34035  010 001147                             S4   TAB        S4 = BUF ADDR
 34036  007 147207                        INC  S5   S4         S5 = NEXT ADDR
 34037  210 050007              WRTE           TAB  S5         UPDATE BUF ADDR
                          *************************
                          * WRITE DATA INTO BUFFER *
                          *************************
 34040  010 046647                             M    S4         M = BUF ADDR
 34041  210 042007              WRTE           TAB  S2         WRITE DATA
```

EXAMPLE 4: DRIVER MICROPROGRAMMED PRIVILEGED SECTION (Sheet 2 of 3)

```
PAGE 0003 RTE MICRO-ASSEMBLER REV.A 760805

                          ************************************************
                          * READ (& UPDATE) DATA COUNT FROM COUNT INTO S4 *
                          * FORM STC 4 IN S2, FORM STC 5 IN S2            *
                          * WRITE UPDATED DATA COUNT INTO COUNT           *
                          ************************************************
34042  227 174707                READ      INC  PNM  P        READ DATA COUNT
34043  350 073062                IMM  L1   CMLO S2   35B      S2 = 704 = STC 4
34044  007 143047                          INC  S2   S2       S2 = 705 = STC 5
34045  007 101147                          INC  S4   TAB      S4 = NEW COUNT
34046  210 046007                WRTE           TAB  S4       WRITE NEW COUNT
                          **********************
                          * UPDATED COUNT = 0 ? *
                          **********************
34047  320 043302                JMP  CNDX ALZ  RJS  STC      NO, STC SC,C
                          ***********************************
                          * TURN OFF DEVICE (EXECUTE CLC SC) *
                          ***********************************
34050  010 044606                     IOG       IRCM S3      EXEC CLC SC
                          ***************************************************
                          * PLACE ADDRESS OF EQ15 IN S1                     *
                          * READ ADDRESS OF EQT15 USING S1 & INDIRECT ROUTINE *
                          * FORM ADDRESS OF EQ4 IN S1 (S1=S1+1)             *
                          * FORM -1 IN S3, WRITE -1 INTO EQT15              *
                          ***************************************************
34051  343 172507                IMM       LOW  L    375B     L = 177775 = -3
34052  004 175007                          SUB  S1   P        S1 = EQ15 ADDR
34053  230 040647                READ           M    S1       GET EQT15 ADDR
34054  300 012477                JSB  IOFF           INDIRECT
34055  007 141007                          INC  S1   S1       S1 = ADDR OF EQ4
34056  343 177107                IMM       LOW  S3   377B     S3 = 177777 = -1
34057  210 044007                WRTE           TAB  S3       S3 EQT15 = -1
                          ************************************************
                          * READ ADDRESS OF EQT4 USING S1 & INDIRECT ROUTINE *
                          * TURN ON BIT 12 IN VALUE READ FROM EQT4           *
                          * WRITE UPDATED EQT4 VALUE INTO EQT4               *
                          ************************************************
34060  230 040647                READ           M    S1       READ EQT4
34061  300 012477                JSB  IOFF           INDIRECT
34062  347 136507                IMM       HIGH L    357B     L = 167777
34063  011 001007                          SONL S1   TAB      TURN ON BIT 12
34064  210 040007                WRTE           TAB  S1       EQT4 BIT 12 = 1
34065  327 003347                JMP                 MPSTAT    CHK MEM. PROT.
                          ***********************************************
                          * READY DEVICE FOR NEXT INPUT (EXECUTE STC SC,C) *
                          ***********************************************
34066  010 040606  STC           IOG       IRCM S1      EXEC STC SC,C
```

EXAMPLE 4: DRIVER MICROPROGRAMMED PRIVILEGED SECTION (Sheet 3 of 3)

```
PAGE 0004 RTE MICRO-ASSEMBLER REV.A 760805

                          ************************************
                          * WAS MEMORY PROTECT ON ? (MPTFL=0?) *
                          ************************************
34067  227 174707  MPSTAT   READ       INC  PNM  P          READ MPTFL
34070  300 012477           JSB  IOFF            INDIRECT
34071  010 000743                ION             TAR        MPTFL = 0 ?
34072  320 043602           JMP  CNDX ALZ  RJS  *+2          NO. LEAVE
                          *                                   MEM. PROT. OFF
                          ******************************************
                          * TURN ON MEMORY PROTECT (EXECUTE STC 5) *
                          ******************************************
34073  010 042606                IOG        IRCM S2        EXEC STC 5
                          ***************************************************
                          * !! OMIT IF OPERATING SYSTEM IS RTE 2 !!         *
                          * RESTORE DMS STATUS, EXIT (JMP JRS I.E. 20354B ) *
                          ***************************************************
34074  227 174707           READ       INC  PNM  P          JRS EXPECTS A
34075  324 016607           JMP                  JRS          READ OF DMSTS
                          **************************
                          * !! DO ONLY FOR RTE 2 !! *
                          * PERFORM A JMP PMXX,I    *
                          **************************
                          *                  INC  P    P          P => DEF PMXX,I
                          *         READ     INC  PNM  P          READ PMXX ADDR
                          *         JSB  IOFF            INDIRECT
                          *         READ MPCK INC  P    M          JMP PMXX,I
                          *         RTN  ION
                                    END

END OF PASS 2: NO ERRORS
```

APPENDIXES

Appendix A
ABBREVIATIONS AND DEFINITIONS

ABBREVIATIONS AND DEFINITIONS

APPENDIX A

An alphabetically arranged listing of abbreviations and definitions used in the manual follows. The listing does not contain definitions of terms such as X-register, S-register, etc., or definitions for languages (FORTRAN, etc.) and other commonly used terms such as K, nS., etc. Pseudo-microinstructions, abbreviations and definitions for micro-orders, and main memory (Assembly language) instructions are not included either. Refer to the computer operating and reference manual or to micro-order lists in this manual for explanations of these mnemonics.

ABBREVIATION	DEFINITION
AAF	A-Addressable Flip-flop
ACM	Association of Computer Manufacturers
ALU	Arithmetic/Logic Unit or ALU field (word type I microinstruction)
ASG	Alter-Skip Group (machine instruction category)
BAF	B-Addressable Flip-flop
BKTBL	Breakpoint table (MDE)
BRCH	Branch micro-order field, word type III or IV microinstruction
BSM	Batch Spool Monitor (RTE subsystem software module)
CIR	Central Interrupt Register
CM	Control memory
CMAR	Control Memory Address Register
CNDX	Condition field, word type II microinstruction
CNTL	Control
CNTR	Counter, either the lower eight bits of the Instruction Register or a micro-order.
COND	Condition field, word type III microinstruction
CPU	Central Processor Unit
CRT	Cathode ray tube (console device)
DCPC	Dual Channel Port Controller (computer accessory)
DMS	Dynamic Mapping System (13305A accessory)
DSPI	Display indicator register or a micro-order
DSPL	Display register or a micro-order
DVR36	Driver 36 for WCS board (12978A and 13197A)
EAG	Extended Arithmetic Group (machine instruction category)
EAU	Extended Arithmetic Unit (machine category)

ABBREVIATION	DEFINITION
EDITR	RTE System Interactive Editor software module
EIG	Extended Instruction Group (machine instruction category)
EOF	End of file
EQT	RTE system equipment table
ESP	Engineering supplement package
EXEC	RTE system call to operating system
FAB	Firmware Accessory Board (13304A 3.5K CM storage accessory)
FF	Flip-flop (single-bit storage element)
FFP	Fast FORTRAN Processor (computer accessory)
FFT	Fast Fourier Transform
FMGR	File Manager (RTE system)
FPP	Hardware Floating Point Processor
HP	Hewlett-Packard
I/O	Input/Output
IBL	Initial Binary Loader
IC	Integrated circuit
IOG	Input-Output Group (machine instruction category)
IR	Instruction Register
KB/S	Kilobytes per second
KP/S	Kilopairs per second
KW/S	Kilowords per second
LED	Light-Emitting Diode (indicators on the computer)
LG	Load and Go (tracks in RTE system)
LOADR	RTE system loader (program name)
LS	Logical Source (tracks in RTE system)
LU	RTE system Logical Unit designator
M	M-register
MDE	Microdebug Editor (microprogramming support software)
MDEP	Name for MDE user scheduled (stand-alone) program
MDES	Name for MDE callable (subroutine) program
MEAR	Memory Address Register (DMS)
MEM	Memory Expansion Module (part of DMS)
MICRO	Program name for RTE Microassembler (microprogramming support software)
MIR	Microinstruction Register

ABBREVIATION	DEFINITION
MJL	Microjump Logic
MOD	Modifier field, word type II microinstruction
MP	Memory Protect
MPP	Multiprogrammable Processor Port
MRG	Memory Reference Group (machine instruction category)
MXREF	Name for RTE Microassembler Cross-Reference Generator (microprogramming support software)
OP	Operation field, word type I and II microinstructions
P	P-register
pROM	Programmable Read-Only Memory (integrated circuits)
PTGEN	Program name for pROM Tape Generator (microprogramming support software)
R-S	Rotate/shift (logic)
RAM	Random Access Memory
ROM	Read-Only Memory (used in control memory, map logic, etc.)
RPL	Remote Program Load Configuration switches
RTE	Real Time Executive (operating system)
RU	RTE system command designation
SC	Select code
SRG	Shift-Rotate Group (machine instruction category)
STR	Store field, word type I and II microinstructions
SYS	System
TTY	Teleprinter (console device)
UCS	User Control Store (13047A 2K CM storage accessory)
UIG	User Instruction Group (machine instruction category)
USR	User
WCS	Writable Control Store (13197A 1K storage accessory)
WCSLT	WCS logical unit table
WLOAD	WCS I/O Utility (library) routine (microprogramming support software)
XFER	Transfer

Appendix B
MICROINSTRUCTION FORMATS

MICROINSTRUCTION FORMATS

APPENDIX B

The four word type formats accepted by the microassembler appear below. The same type information appears at the top of the microprogramming form contained in appendix D.

Word Type 1	LABEL	OP	SPECIAL	ALU	STORE	S-BUS	COMMENTS
Word Type 2	LABEL	"IMM"	SPECIAL	MODIFIER	STORE	OPERAND	COMMENTS
Word Type 3	LABEL	BRANCH	"CNDX"	CONDITION	BRANCH SENSE	ADDRESS	COMMENTS
Word Type 4	LABEL	"JMP" OR "JSB"	MODIFIER/ SPECIAL			ADDRESS	COMMENTS
	FIELD 1 1	FIELD 2 10	FIELD 3 15	FIELD 4 20	FIELD 5 25	FIELD 6 30	FIELD 7 40 72

OBJECT MICROCODE

The HP 1000 E-Series or F-Series object code microinstruction is represented by a nine digit octal number, as follows:

XXX XXXXXX

The left three digits represent bits 23-16 of the microinstruction (the leftmost digit represents bits 23 and 22). Of the remaining six digits, the leftmost represents bit 15 and the other five represent bits 14-0.

Construct the octal representation of an object code microinstruction in the following way. Determine the binary codes of the required micro-orders from appendix C. Form the codes, according to fields, into a 24-bit string. Convert the string to octal by grouping bits.

Example:

Op	Special	ALU	Store	S-bus
ARS	L1	PASS	B	B

Micro-orders

Op	ALU	S-bus	Store	Special
0 0 0 1	1 0 0 0 0	0 0 1 0 0	0 0 1 0 0	1 0 0 1 0

Binary Object Code

23 19 14 9 4

0 3 0 0 1 0 2 2 2

Nine Digit Octal Number

Appendix C
MICRO-ORDER SUMMARY AND SPECIALIZED MICROPROGRAMMING

MICRO-ORDER SUMMARY AND SPECIALIZED MICROPROGRAMMING

APPENDIX C

BINARY FIELD MICRO-ORDER SUMMARY

MICROASSEMBLER → (SOURCE) COLUMN NO. → BITS (ROM) →	OP/ BRANCH 10 23 - 20	MODIFIER/ SPECIAL 15 4 - 0	ALU 20 19 - 15	JMP COND 20 19 - 15	IMMEDIATE MODIFIER 20 19 - 18	STORE 25 9 - 5	BRANCH SENSE 25 14	S-BUS 30 14 - 10
WORD TYPES	I - IV	I - IV	I	III	II	I, II	III	I
Bit Pattern								
00000	*NOP	RTN	DEC	ALZ	LOW	TAB	†RJS	TAB
00001	ARS	§JTAB	OP11	ONES	HIGH	CAB		CAB
00010	CRS	CNDX	OP10	COUT	CMLO	‡MPPA		‡MPPA
00011	LGS	**ION	DBLS	AL0	CMHI	A		A
00100	NRM	**RJ30	OP8	L0		B		B
00101	DIV	**J74	OP7	L15		**IOO		**IOI
00110	LWF	**IOG	ADD	RUN		DSPL		DSPL
00111	MPY	*NOP	OP6	**HOI		DSPI		DSPI
01000	WRTE	SRUN	OP5	CNT4		‡MPPB		‡MPPB
01001	READ	‡MPP2	SUB	IR11		‡MEU		‡MEU
01010	ENV	‡MESP	OP4	‖RUNE		L		**CIR
01011	ENVE	COV	OP3	‖NMLS		CNTR		CNTR
01100	JSB	SOV	ZERO	‡MPP		**IRCM		LDR
01101	JMP	PRST	OP2	CNT8		M		M
01110	IMM	CLFL	OP1	‖NSFP		PNM		**DES
01111	RTN	STFL	INC	AL15		*NOP		*NOP
10000		**SRG2	*PASS	NLDR		S1		S1
10001		**SRG1	IOR	NSTB		S2		S2
10010		L1	SONL	NINC		S3		S3
10011		L4	ONE	NDEC		S4		S4
10100		R1	AND	NRT		S5		S5
10101		DCNT	PASL	NLT		S6		S6
10110		ICNT	XNOR	NSTR		S7		S7
10111		RPT	NSOL	NMDE		S8		S8
11000		‖ASG	SANL	FLAG		S9		S9
11001		‖IAK	XOR	E		S10		S10
11010		‡MPP1	CMPL	NINT		S11		S11
11011		§FTCH	NAND	OVFL		SP		SP
11100		‡INCI	OP13	NSNG		X		X
11101		SHLT	NSAL	**SKPF		Y		Y
11110		‡MPCK	NOR	IR8		P		P
11111		**IOFF	CMPS	MRG		S		S

*Default micro-order.
†If no RJS, bit 14 = 0.
‡Means not normally used by user microprogrammer unless a specific accessory is installed.
§Means included here for completness only; reserved for exclusive use of system microprogrammers.
‖Not normally used by user microprogrammer.
**Use with caution (i.e., be completely familiar with the function.)

SPECIAL USE MICRO-ORDERS

Two micro-orders (FTCH and JTAB) assigned to the word type I Special field are used only in the base set. These two micro-orders are listed in table 4-1 and in the various micro-order summaries only for completeness. They are not to be used in "normal" user microprogramming because of their complex functions and effect on the Save Stack. However, if you are planning to do system emulation, you may have need of the summary information presented below.

FTCH. The FTCH micro-order does the following:

a. Stores the present contents of the M-register into the Memory Protect Violation register if Memory Protect is installed. This is usually the address of the next Assembly language instruction to be executed.

b. Clears the Memory Protect Violation Flag flip-flop and Indirect Counter if Memory Protect is installed.

c. Clears the L-register and the CPU flag.

d. Resets microsubroutine Save Stack address logic.

JTAB. The JTAB micro-order is used to complete the Fetch microroutine and begin the execution operation. JTAB works as follows:

a. If INCI was not specified in the Special field of the previous microinstruction, JTAB calls for the CMAR to be loaded with an execution microroutine address dependent upon the eight most significant bits (15-8) of the IR. These eight bits functions as an address to the Jump Table, the contents of which become the target branch address.

   If INCI was specified in the previous microinstruction, the branch as described above is made only if the condition mapped by bits 19-14 of the microinstruction is met. The condition will be coded with ALU and S-bus field micro-orders, *not* Condition field (word type III) micro-orders. For example, JTAB is used once in the base set at CM location 2. The Condition field is represented by the ALU field (INC) which has the same bit pattern as AL15 in the Condition field. Bit 14 of the microinstruction is one (P is in the S-bus field) so the RJS feature is enabled. Therefore the branch through the Jump Table will only by made if the conditions of AL15 RJS are met. When the specified conditions are met bit 15 of the IR is masked to Look-up table, then the branch through the jump table address in IR bits 15-8 is executed.

b. If the Run flip-flop is reset or an I/O interrupt is pending and not held off by the Interrupt Enable flip-flop (refer to IOFF in the Special field, table 4-1) and INCI was not specified in the previous microinstruction, the operation in the store field is inhibited and a branch to CM location 6 will occur instead of a branch to the address specified by the Jump Table.

c. Inhibits the operation specified in the Store field if a Memory Reference Group instruction is in the IR and bit 15 out of ALU was set during the previous word type I or II microinstruction or, if a JMP, JSB, STA, STB, or ISZ Assembly language instruction is in the IR. Logically:

   $$\text{Inhibit Store} = \text{JTAB}[(IR14 + IR13 + IR12)\ AL15 + IR14 \cdot IR12 \cdot IR11 + IR14 \cdot IR13 \cdot IR12 + IR14 \cdot IR13 \cdot IR11]$$

d. Turns on the Interrupt Enable flip-flop.

e. Initializes the microsubroutine Save Stack address logic.

Because of JTAB's complex functional structure, and intended use (it can be seen only at locations 00001, 00003 and 00305 in the base set), it should *not* be used in normal "user" microprogramming.

MAPPING DETAILS

Section 6 provides information on usable UIG instructions and related CM entry point addresses. An understanding of that information is prerequisite to the material in this appendix. The base set mapping procedure, UIG instruciton decoding (bits 15 through 8), module selection code indexing (bits 8 through 4), and secondary indexing (bits 3 through 0), are explained below. These explanations primarily concern UIG mapping but, some information on the HP reserved areas is also included so that if you plan system emulation the appropriate data can be extracted. It should be noted that it is not intended that the HP 21MX E-Series Computer base set be changed. The base set mapping concept is applicable to any instruction placed in the IR.

UIG DECODING

The base set FETCH microroutine will normally be used to store the UIG instruction in the IR. This procedure occurs during execution of the microinstruction at CM location 00000. (See the base set listing in appendix G for all references to CM base set locations included in this discussion.) Figure C-1 illustrates UIG instruction bit patterns. Note that bits 15 thorugh 9 must have a 101 or 105 (octal) value to fall within this instruction group.

At location 00001 in the base set, a JTAB micro-order causes examination of bits 15 through 8 of the IR and *conditionally* causes this upper byte to be taken as an index (address) to the ROM Jump Tables. For the JTAB conditions, refer to the JTAB explanation in this appendix immediately preceding this mapping discussion. As seen in figure C-1, the upper 8 bits of a UIG instruction (in the IR), when examined by JTAB, will be decoded as a 203, 212 or 213 (octal) value if they fall within the UIG. The applicable value is applied to the Jump Tables as the lower three (octal) digits of the Jump Table address (first two digits, 02, masked off). (See the Jump Table listing at the end of appendix G). The lower bits of the value unloaded from the Jump Tables are applied to the CMAR as the CM location to be branched to in the first step in determining the desired final CM location.

UIG Jump Table addresses 02203 and 02213 (bit 8 of the IR equals 1 in each case) both cause value 000 000107 (octal) to be unloaded from the ROM Jump Tables. (See appendix G.) This, in turn, is used as the CMAR location value 00107 to obtain the next microinstruction. Hence, it can be seen from the Jump Table listing that for UIG instructions beginning 101xxx and 105xxx (xxx equals values as shown in table 6-1), a branch to location MAC1 (00107) in the base set will be made. This means bit 11 (the bit causing the difference between 101 and 105) can be used (as described in paragraph 6-3) to pass A- and B-register information from main memory to all CM locations mapped to by UIG instructions beginning with either code. Note, from table 6-1, that bit 11 is not usable for this purpose when mapping to modules that only have UIG instructions with bits 15 through 9 equal to 105 (octal) available (e.g., user modules 60 and 62).

If UIG instructions 105400 through 105777 are used (02213 applied as an address to the Jump Tables), it can be seen from the base set, Jump Table listings, and figure C-1 that all mapping will be through MAC1 (CM location 00107 in the base set) for this first step. If UIG instructions 105000 through 105377 are used (02212 applied as an address to the Jump Tables) it can be seen that all mapping will be through MACO (CM location 00103 in the base set) for this first step.

MODULE SELECTION

Step 2 in figure C-1 illustrates that module selection is made as the second step (primary map) toward the desired final CM location. The UIG module selection code, composed of UIG instruction bits 8 through 4, is used in determining mapping to a particular CM module. A group of modules (as implied in the preceding paragraph) to be mapped to is determined by examination of bit 8. Examination of bits 7 through 4 of the UIG instruction determines the module to be mapped to within the selected group.

Figure C-2 shows the bit patterns available for all UIG instructions. Note that with the five bits (8 through 4) of the module selection code, 32 combinations are possible. This means 32 module entry points are available. Bit 8 (used to select CM location 00103 or 00107, at labels MAC0 or MAC1) determines whether mapping will be through MACTABL0 or MACTABL1 in the base set Primary Mapping Table. It can be seen (in figure C-2 and the base set listings) that if bit 8 equals 0, MACTABL0 will be used and if bit 8 equals 1, MACTABL1 will be used.

From base set locations 00103 (label MAC0) or 00107 (label MAC1) in the Input-Output Group microroutines, a word type IV branch is made to either MACTABL0 or MACTABL1, respectively, using a J74 micro-order. This micro-order examines bits 7 through 4 of the UIG instruction in the IR to determine the module to be mapped to within the group selected by bit 8 (MACTABL0 or MACT-ABL1).

This discussion is best followed by referring to the base set listing (appendix G) in conjunction with figure C-2. MACTABL1 begins at CM location 00760 and extends through CM location 00777 (16 locations). MACTABL0 begins at CM location 01000 and extends through CM location 01017 (16 locations). Both these (above) are in the base set Primary Mapping Table.

The J74 micro-order (at MAC0 or MAC1) replacement of bits 8 through 5 in the microinstruction branch address field by bits 7 through 4 from the IR completes the second step in mapping (the primary map). With completion of this step, the offset for entry into the Primary Mapping Tables is determined; i.e., the specific control memory module is determined). See figure 4-5, Jump Address Decoding, and the J74 micro-order explanation in table 4-1 for information on branch address field modifications using the J74 micro-order for indexing.

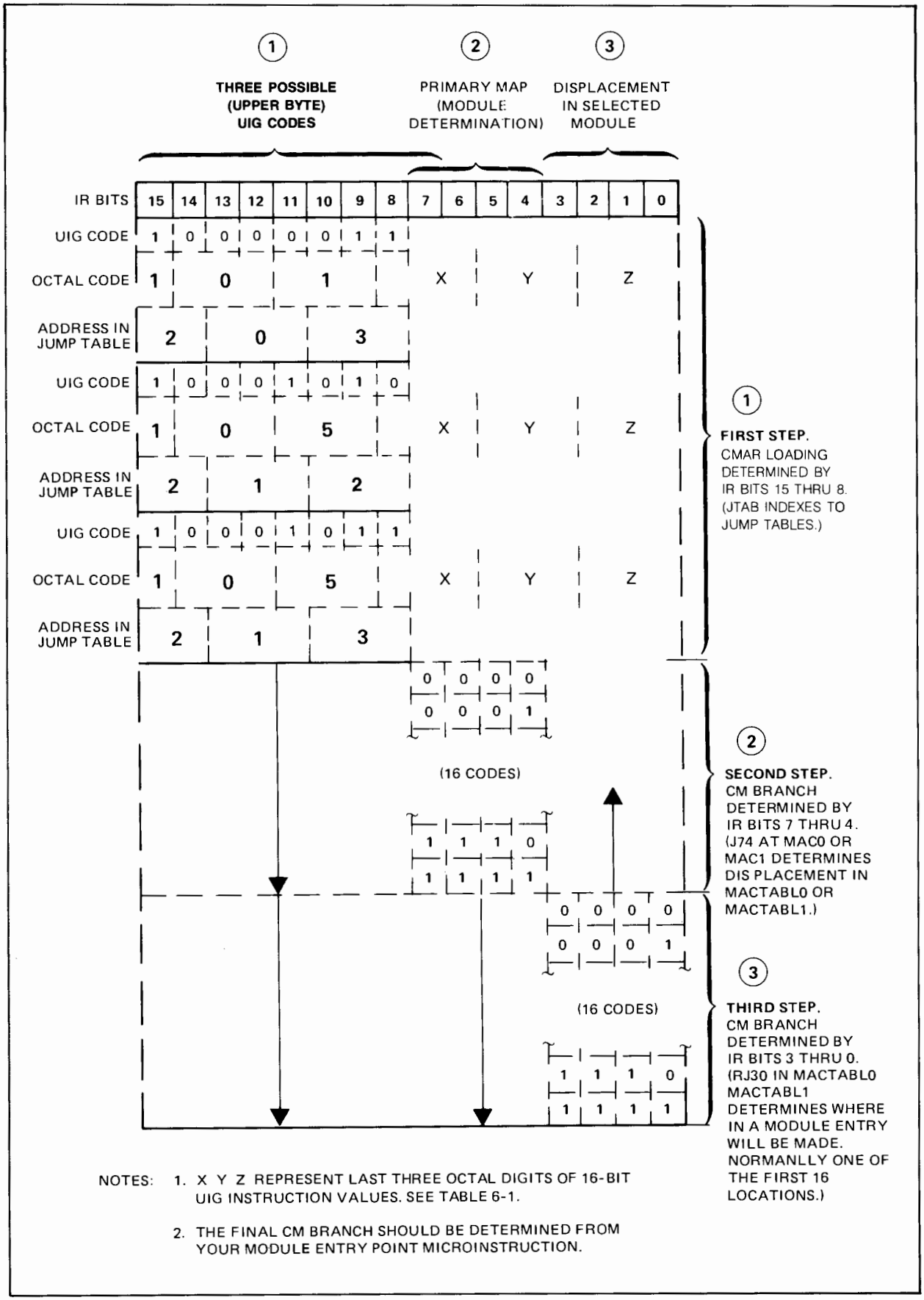
Compare figure C-2 and the base set Primary Mapping Table and you will notice that HP reserved modules 2, 3, 32, and 39 have 2, 6, 2, and 3 entry points (respectively) assigned. CM entry points mapped to are so noted in figure C-2, and note in the base set Primary Mapping Table that modules 3 and 39 do not have branch address modification micro-orders (RJ30) in their microinstructions. Some study of the situation is required if you are going to attempt changes to this system and as mentioned in section 6, the description is beyond the scope of this manual. The discussion for the generally used third step in mapping (secondary) index follows.

SECONDARY INDEX

By examining figure C-2 and the Primary Mapping Table, it can be seen that all modules of the User Instruction Group (except 2, 3, 32 and 39 mentioned above) have a single module selection code assigned. This means that the microinstruction appearing in the Primary Mapping Table for a particular module represents the primary software entry point (step 3 figure C-1) for access to that module. This entry point is expanded to 16 possible entry points per module by the secondary index. That is, as noted in figures C-1 and C-2, (step 3 of mapping to the desired final CM location entry point) examination of bits 3 through 0 of the UIG instruction takes place in MACTABL0 or MACTABL1.

This is accomplished by using the RJ30 micro-order in the Special field for the branch microinstructions (shown in the Primary Mapping Table). RJ30 causes bits 8 through 5 of the word type IV microinstruction branch address field to be replaced by bits 3 through 0 of the IR. RJ30 also begins a read operation from main memory as the branch to the desired module begins (indexed into one of the first 16 locations by bits 3 through 0 of the UIG instruction in the IR).

See the information in table 4-1 (RJ30), figure 4-5, and appendix B on branch address modification and decoding. Also, see the information on microassembler pseudo-microinstructions (e.g., ALGN) in section 8 and the information for the ION and IOG micro-orders (used in word type IV) for branch address field modifications.



7115-19

Figure C-1. UIG Instruction Bit Decoding

USER INSTRUCTION GROUP CODES

INSTRUCTION REGISTER BITS

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		CODE MAPS TO MODULE HP	CODE MAPS TO MODULE USER	CONTROL MEMORY STARTING ADDRESS	COMMENTS
(212) MAC0	1	0	0	0	1	0	1	0	0	0	0	0	W	X	Y	Z	MACTABL0, BIT 8=0	3		01433	FAD *
								0	0	0	0	1						3		01433	FSB *
								0	0	0	1	0						3		01522	FMP *
								0	0	0	1	1						3		01561	FDV *
								0	0	1	0	0						3		01405	FIX *
								0	0	1	0	1						3		01400	FLT *
								0	0	1	1	0							60	36000	USER'S AREA
								0	0	1	1	1							62	37000	USER'S AREA
								0	1	0	0	0						34		21000	FFP
								0	1	0	0	1						35		21400	FFP
								0	1	0	1	0						36		22000	HP RES. (RTE)
								0	1	0	1	1						37		22400	HP RES. (RTE)
								0	1	1	0	0						38		23000	HP RES.
								0	1	1	0	1						40		24000	HP RES.
								0	1	1	1	0						44		26000	HP RES.
	1	0	0	0	1	0	1	0	1	1	1	1						45		26400	HP RES.
(203 or 213) MAC1	1	0	0	0	N	0	1	1	0	0	0	0					MACTABL1, BIT 8=1	39	(Note 5)	23420**	HP RES. *
								1	0	0	0	1						39	(Note 5)	23420**	HP RES. *
								1	0	0	1	0							46	27000	USER'S AREA
								1	0	0	1	1						39	(Note 5)	23400	HP RES. *
								1	0	1	0	0							47	27400	USER'S AREA
								1	0	1	0	1							48	30000	USER'S AREA
								1	0	1	1	0							49	30400	USER'S AREA
								1	0	1	1	1							50	31000	USER'S AREA
								1	1	0	0	0							56	34000	USER'S AREA
								1	1	0	0	1							57	34400	USER'S AREA
								1	1	0	1	0							58	35000	USER'S AREA
								1	1	0	1	1							59	35400	USER'S AREA
								1	1	1	0	0						32	(Note 4)	20000	DMS
								1	1	1	0	1						32	(Note 4)	20020	DMS
								1	1	1	1	0						2		01020	EIG
	1	0	0	0	N	0	1	1	1	1	1	1	W	X	Y	Z		2		01040	EIG

(N=1 OR 0)

JUMP TABLE INDEX — MODULE SELECTION CODE (32 POSSIBLE CODES) — OFFSET IN MODULE (NOTE 1)

NOTES:

1. W, X, Y, Z, ARE LOGIC ONES OR ZEROS MAKING UP A RANGE OF CODES.
2. CONTROL MEMORY STARTING ADDRESSES ARE IN OCTAL.
3. ABBREVIATION RES. MEANS RESERVED. OTHER ABBREVIATIONS IN THE COMMENTS ARE EXPLAINED IN TEXT.
4. MODULE 32 HAS 32 ENTRY POINTS.
5. MODULE 39 HAS 48 ENTRY POINTS HOWEVER 32 ARE MAPPED DIRECTLY TO LOCATION 23420 FOR USE IN ONE OF THE MICROINSTRUCTIONS. THAT IS, ALL 16 COMBINATIONS OF "USER" INSTRUCTIONS MAP DIRECTLY TO THE ENTRY POINT DESIGNATED BY** (NON-INDEXED).
6. *NO EXAMINATION OF BITS 3 THROUGH 0 IN THE PRIMARY MAPPING TABLE FOR E-SERIES BUT ARE USED FOR F-SERIES.

7115-20A

Figure C-2. UIG Instruction Module Mapping

Appendix D
MICROPROGRAMMING FORM

MICROPROGRAMMING FORM

APPENDIX D

(Actual size: 12.5" x 10.5")

HP 21MX SERIES MICROPROGRAMMING FORM

PROGRAMMER	DATE	MICROPROGRAM	MODULE	PAGE OF

LABEL	OP	SPECIAL	ALU	STORE	S-BUS	COMMENTS	Word Type 1
LABEL	"IMM"	SPECIAL	MODIFIER	STORE	OPERAND	COMMENTS	Word Type 2
LABEL	BRANCH	"CNDX"	CONDITION	BRANCH SENSE	ADDRESS	COMMENTS	Word Type 3
LABEL	"JMP" OR "JSB"	MODIFIER/ SPECIAL			ADDRESS	COMMENTS	Word Type 4
1 FIELD 1	10 FIELD 2	15 FIELD 3	20 FIELD 4	25 FIELD 5	30 FIELD 6	40 FIELD 7	80

1 10 15 20 25 30 40 80

Ø = ZERO I or 1 = ONE I = ALPHA I

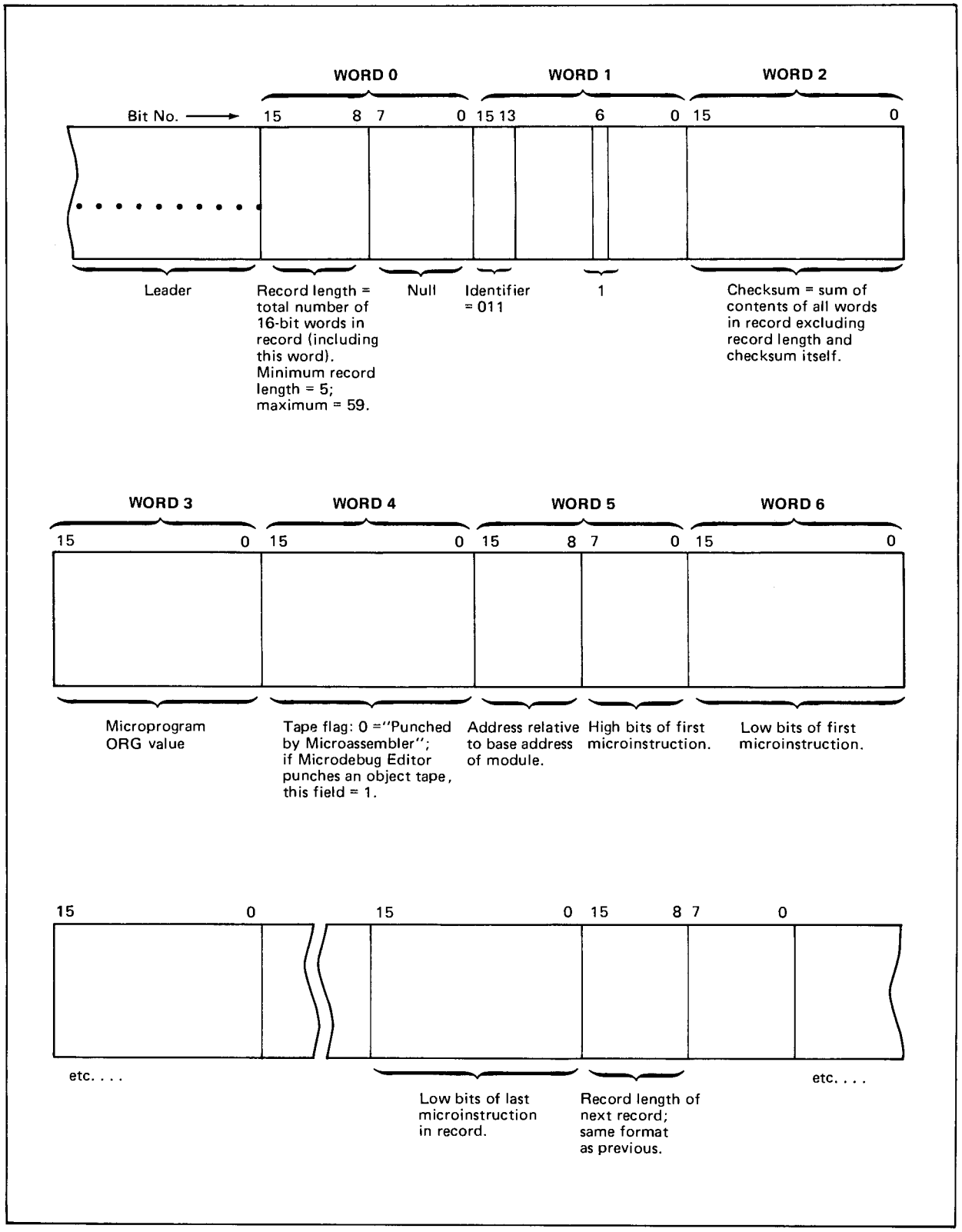
O = ALPHA O 2 = TWO Z = ALPHA Z

5951-9162

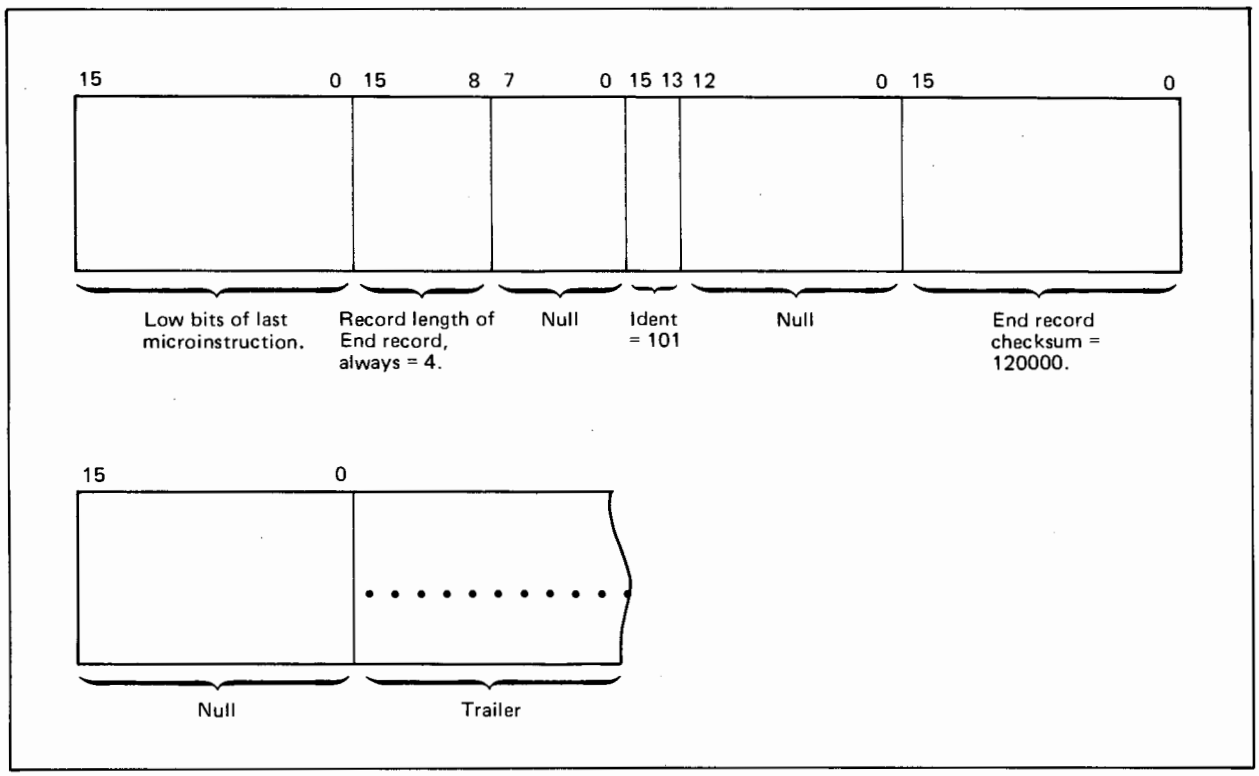
Appendix E
OBJECT TAPE FORMATS

OBJECT TAPE FORMATS

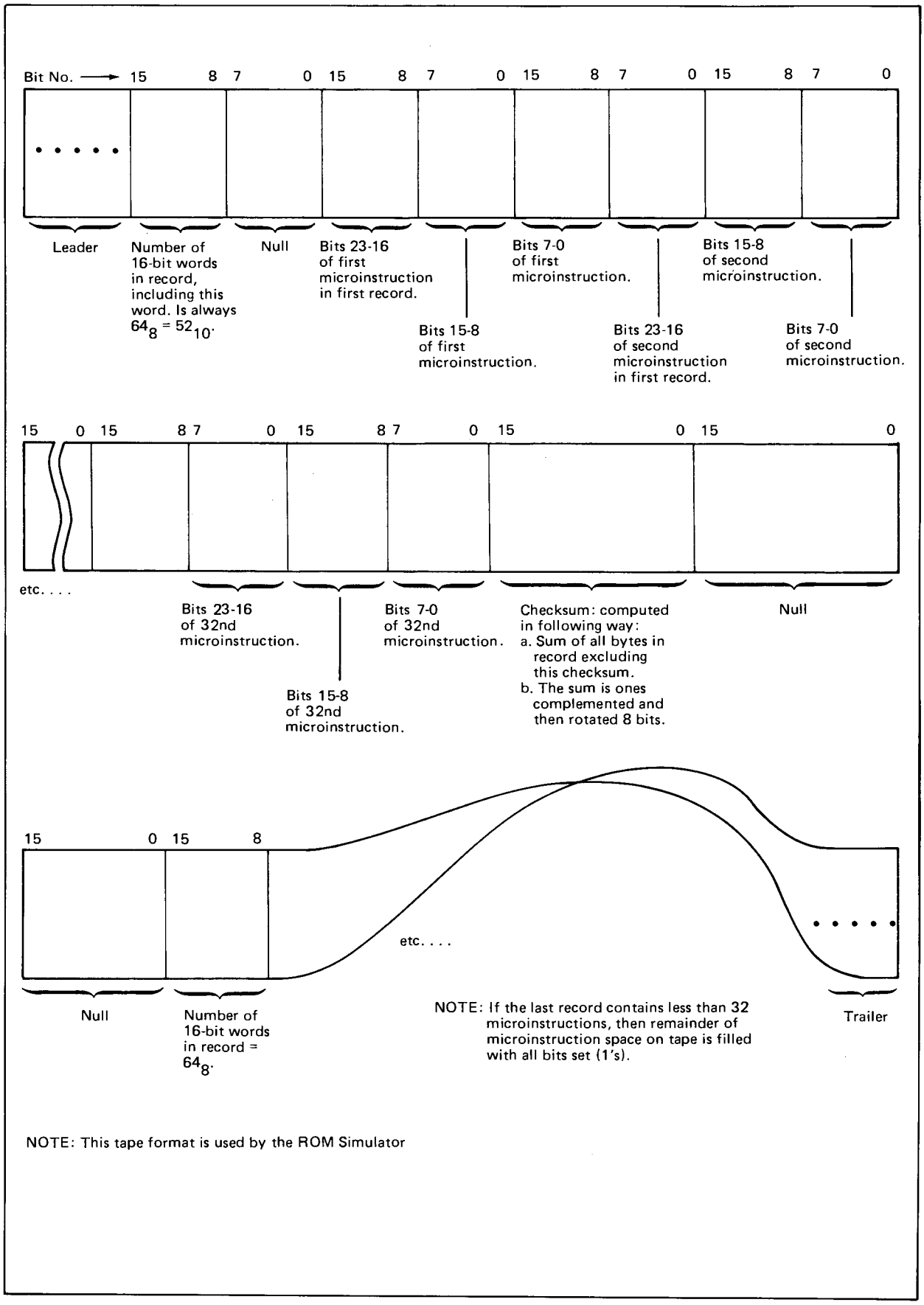
APPENDIX E



Format of Standard Object Tape (Sheet 1 of 2)



Format of Standard Object Tape (Sheet 2 of 2)



Format of Object Tape for the "S" Microassembler Option

Appendix F
HP 1000 M-SERIES-TO-HP 1000 E-/F-SERIES MICRO-ORDER COMPARISON SUMMARY

M-SERIES-TO-E-/F-SERIES MICRO-ORDER COMPARISON SUMMARY

APPENDIX F

This summary includes a comparison of all the HP 21MX Computer micro-orders and all E-Series and F-Series Computer micro-orders. If you already have microprograms prepared for HP 21MX Computers the summary will be helpful for making a conversion. Note that some micro-orders have identical mnemonics and bit patterns. In most instances, however, the bit patterns vary. There is a percentage of the micro-orders that are completely new and also, a percentage of micro-orders that have not propagated from the HP 21MX to the newer Series. You should refer to the "dictionary" section of the micro-orders for each computer document to determine the exact meaning and functions of micro-orders you plan to use.

Micro-Order Comparison Summary

FIELD COLUMN NO. (ROM BITS)	OP/BRANCH 10 23-20		MOD/SPECIAL 15 4-0		ALU 20 19-15		COND 20 19-15		IMM/MOD 20 19-18		STORE 25 9-5		BRANCH SENSE 25 14		S-BUS 30 14-10	
COMPUTER	M-	E-/F-	M-	E-/F-	M-	E-/F-	M-	E-/F-	M-	E-/F-	M-	E-/F-	M-	E-/F-	M-	E-/F-
Corresponding Bit Pattern																
00000	NOP	NOP	IOFF	RTN	INC	DEC	TBZ	ALZ	HIGH	LOW	TAB	TAB		RJS	TAB	TAB
00001	ARS	ARS	SRG2	JTAB	OP1	OP11	ONES	ONES	LOW	HIGH	CAB	CAB	RJS		CAB	CAB
00010	CRS	CRS	L1	CNDX	OP2	OP10	COUT	COUT	CMHI	CMLO	T	MPPA			T	MPDA
00011	LGS	LGS	L4	ION	ZERO	DBLS	AL0	AL0	CMLO	CMHI	L	A			CIR	A
00100	MPY	NRM	R1	RJ30	OP3	OP8	AL15	L0			IOO	B			IOI	B
00101	DIV	DIV	ION	J74	OP4	OP7	NMLS	L15			CNTR	IOO			CNTR	IOI
00110	LWF	LWF	SRG1	IOG	SUB	ADD	CNT8	RUN			DSPL	DSPL			DSPL	DSPL
00111	WRTE	MPY	RES2	NOP	OP5	OP6	FPSP	HOI			DSPI	DSPI			DSPI	DSPI
01000	ASG	WRTE	STFL	SRUN	OP6	OP5	FLAG	CNT4			IR	MPPB			ADR	MPPB
01001	READ	READ	CLFL	MPP2	ADD	SUB	E	IR11			M	MEU			M	MEU
01010	ENV	ENV	FTCH	MESP	OP7	OP4	OVFL	RUNE			B	L			B	CIR
01011	ENVE	ENVE	SOV	COV	OP8	OP3	RUN	NMLS			A	CNTR			A	CNTR
01100	JSB	JSB	COV	SOV	OP9	ZERO	NHOI	MPP			MEU	IRCM			LDR	LDR
01101	JMP	JMP	RPT	PRST	OP10	OP2	SKPF	CNT8			CM	M			RES2	M
01110	IMM	IMM	SRGE	CLFL	OP11	OP1	ASGN	NSFP			PNM	PNM			MEU	DES
01111		RTN	NOP	STFL	DEC	INC	IR2	AL15			NOP	NOP			NOP	NOP
10000			MESP	SRG2	CMPS	PASS	NLDR	NLDR			S1	S1			S1	S1
10001			MPCK	SRG1	NOR	IOR	NSNG	NSTB			S2	S2			S2	S2
10010			IOG	L1	NSAL	SONL	NINC	NINC			S3	S3			S3	S3
10011			ICNT	L4	OP13	ONE	NDEC	NDEC			S4	S4			S4	S4
10100			SHLT	R1	NAND	AND	NRT	NRT			S5	S5			S5	S5
10101			INCI	DCNT	CMPL	PASL	NLT	NLT			S6	S6			S6	S6
10110			RES1	ICNT	XOR	XNOR	NSTR	NSTR			S7	S7			S7	S7
10111			SRUN	RPT	SANL	NSOL	NRST	NMDE			S8	S8			S8	S8
11000			UNCD	ASG	NSOL	SANL	NSTB	FLAG			S9	S9			S9	S9
11001			CNDX	IAK	XNOR	XOR	NSFP	E			S10	S10			S10	S10
11010			JIO	MPP1	PASL	CMPL	INT	NINT			S11	S11			S11	S11
11011			JTAB	FTCH	AND	NAND	SRGL	OVFL			S12	SP			S12	SP
11100			J74	INCI	ONE	OP13	RUNE	NSNG			X	X			X	X
11101			J30	SHLT	SONL	NSAL	NOP	SKPF			Y	Y			Y	Y
11110			RTN	MPCK	IOR	NOR	CNT4	IR8			P	P			P	P
11111			JEAU	IOFF	PASS	CMPS	NMEU	MRG			S	S			S	S

Appendix G
E-SERIES COMPUTER BASE SET MICROPROGRAM LISTING AND F-SERIES JUMP TABLES

E-SERIES COMPUTER BASE SET MICROPROGRAM LISTING AND F-SERIES JUMP TABLES

APPENDIX G

The entire E-Series Computer RTE Microassembler listing for the base set microprogram and F-Series jump tables appear in this appendix. Control memory modules 0 through 3 are used. Information for the ROM Jump Tables is also included at the back of the base set listing. The microprogram listings for the dynamic mapping instructions and the Scientific Instruction Set conclude this appendix.

```
PAGE 0005 RTE MICRO-ASSEMBLER REV.A 760818
0001                    MICMXE,L,C,T
0002                    $CODE=%E0404,REPLACE
0003                            ORG                     0B
0004                    *
0005                    *       21MX E-SERIES BASE SET MICROCODE
0006                    *       -------------------------------
0007                    *       1978-04-04      DATE CODE 1814
0008                    *
0009 00000  230 000633  FETCH   READ FTCH PASS IRCM TAB       IR := T/A/B; M := OP ADR; READ
0010 00001  007 174701          JTAB INC  PNM  P         JMP THRU LUT--M=P CNDL;P=P+1 CNDL
0011                    *
0012 00002  230 000674  MRGIND  READ INCI PASS M    TAB       M := T/A/B; READ
0013 00003  007 174701          JTAB INC  PNM  P         JMP LUT CNDL--M=P CNDL;P=P+1 CNDL
0014 00004  323 140102          JMP  CNDX HOI  RJS  MRGIND    TEST FOR HALT OR INTERRUPT
0015 00005  336 040102          JMP  CNDX NSNG RJS  MRGIND    TEST FOR INSTRUCTION STEP
0016                    *
0017 00006  000 075707  HORI            DEC  P    P         P := P-1
0018 00007  323 053242          JMP  CNDX RUN  RJS  HALT      TEST FOR HALT
0019 00010  010 036771          IAK                           LOAD CIR; ACKNOWLEDGE INTERRUPT
0020 00011  230 024677          READ IOFF PASS M    CIR       M := CIR; READ TRAP CELL
0021 00012  010 033017          STFL PASS S1   M         S1 := M
0022 00013  230 000607          READ     PASS IRCM TAB       IR := T/A/B; M := OP ADR; READ
0023 00014  320 000047          JMP                FETCH+1
PAGE 0006 RTE MICRO-ASSEMBLER REV.A 760818
0025                    *
0026                    *       MEMORY REFERENCE GROUP
0027                    *       ----------------------
0028                    *
0029 00015  230 000507  AND     READ     PASS L    TAB       L := T/A/B; READ
0030 00016  372 006147          RTN      AND  A    A         A := A AND T/A/B
0031                    *
0032 00017  230 000507  AD*     READ     PASS L    TAB       L := T/A/B; READ
0033 00020  263 002040          ENVE RTN ADD  CAB  CAB       A/B := A/B + T/A/B
0034                    *
0035 00021  230 000507  CP*     READ     PASS L    TAB       L := T/A/B; READ
0036 00022  014 102747                   XOR       CAB       COMPARE
0037 00023  360 000042          RTN  CNDX ALZ                TEST IF EQUAL
0038 00024  227 174707          READ     INC  PNM  P         M := P; P := P+1; READ
0039 00025  370 036747          RTN
0040                    *
0041 00026  230 000507  IOR     READ     PASS L    TAB       L := T/A/B; READ
0042 00027  370 106147          RTN      IOR  A    A         A := A IOR T/A/B
0043                    *
0044 00030  007 101007  ISZ              INC  S1   TAB       S1 := T/A/B + 1
0045 00031  210 040036          WRTE MPCK PASS TAB  S1       T/A/B := S1; WRITE
0046 00032  320 041602          JMP  CNDX ALZ  RJS  *+2      TEST IF ZERO
0047 00033  007 175707                   INC  P    P         P := P+1
0048 00034  227 174707          READ     INC  PNM  P         M := P; P := P+1; READ
0049 00035  370 036747          RTN
0050                    *
0051 00036  230 000677  JMP,I    READ IOFF PASS M    TAB       M := T/A/B; READ
0052 00037  307 112442          JSB  CNDX AL15       INDIRECT  TEST FOR MORE INDIRECTS
0053 00040  367 133736  JMP      RTN  MPCK INC  P    M         P := M+1
0054                    *
0055 00041  230 000677  JSB,I    READ IOFF PASS M    TAB       M := T/A/B; READ
0056 00042  307 112442          JSB  CNDX AL15       INDIRECT  TEST FOR MORE INDIRECTS
0057 00043  210 074036  JSB      WRTE MPCK PASS TAB  P         T/A/B := P; WRITE
0058 00044  007 133716               CLFL INC  P    M         P := M+1
0059 00045  227 174707          READ     INC  PNM  P         M := P; P := P+1; READ
0060 00046  370 036747          RTN
0061                    *
0062 00047  230 000047  LD*      READ     PASS CAB  TAB       A/B := T/A/B; READ
0063 00050  370 036747          RTN
0064                    *
0065 00051  230 000507  XOR      READ     PASS L    TAB       L := T/A/B; READ
0066 00052  374 106147          RTN      XOR  A    A         A := A XOR T/A/B
```

```
PAGE 0007 RTE MICRO-ASSEMBLER REV.A 760818
0068                        *
0069                        *         ALTER-SKIP GROUP
0070                        *         ----------------
0071                        *
0072 00053  230 002047  ASGNO*    READ       PASS CAB  CAB       READ
0073 00054  267 102070            ENVE ASG   INC  CAB  CAB       A/B=A/B + 1 CNDL;RTN CNDL;E CNDL
0074 00055  227 174707            READ       INC  PNM  P         M := P; P := P+1; READ
0075 00056  370 036747            RTN
0076                        *
0077 00057  231 136047  ASGCC*    READ       ONE  CAB            A/B := ONES; READ
0078 00060  267 102070            ENVE ASG   INC  CAB  CAB       A/B=A/B + 1 CNDL;RTN CNDL;E CNDL
0079 00061  227 174707            READ       INC  PNM  P         M := P; P := P+1; READ
0080 00062  370 036747            RTN
0081                        *
0082 00063  226 036047  ASGCL*    READ       ZERO CAB            A/B := ZEROS; READ
0083 00064  267 102070            ENVE ASG   INC  CAB  CAB       A/B=A/B + 1 CNDL;RTN CNDL;E CNDL
0084 00065  227 174707            READ       INC  PNM  P         M := P; P := P+1; READ
0085 00066  370 036747            RTN
0086                        *
0087 00067  237 102047  ASGCM*    READ       CMPS CAB  CAB       A/B := CMP A/B
0088 00070  267 102070            ENVE ASG   INC  CAB  CAB       A/B=A/B + 1 CNDL;RTN CNDL;E CNDL
0089 00071  227 174707            READ       INC  PNM  P         M := P; P := P+1; READ
0090 00072  370 036747            RTN
0091                        *
0092                        *         SHIFT-ROTATE GROUP
0093                        *         ------------------
0094                        *
0095 00073  230 002061  SRG       READ SRG1  PASS CAB  CAB       FIRST SHIFT; CLEAR E CNDL; READ
0096 00074  010 002060                SRG2  PASS CAB  CAB       SECOND SHIFT; RTN CNDL
0097 00075  227 174707  RET       READ       INC  PNM  P         M := P; P := P+1; READ
0098 00076  370 036747            RTN
PAGE 0008 RTE MICRO-ASSEMBLER REV.A 760818
0100                        *
0101                        *         INPUT-OUTPUT GROUP
0102                        *         ------------------
0103                        *
0104 00077  320 004006  IOG       JMP  IOG              MI*         T2: SYNCHRONIZE AND JUMP
0105                        *
0106 00100  230 002507  MI*       READ       PASS L     CAB         T3: L := A/B; READ
0107 00101  010 112747                       IOF        IOI         T4:
0108 00102  370 112047            RTN        IOR  CAB   IOI         T5: A/B := A/B IOR I/O
0109 00103  320 040005  MAC0      JMP  J74              MACTABL0
0110                        *
0111 00104  230 012747  LI*       READ       PASS       IOI         T3: READ
0112 00105  010 012747                       PASS       IOI         T4:
0113 00106  370 012047            RTN        PASS CAB   IOI         T5: A/B := I/O
0114 00107  320 037005  MAC1      JMP  J74              MACTABL1
0115                        *
0116 00110  230 002747  OT*       READ       PASS       CAB         T3: READ
0117 00111  010 002247                       PASS IOO   CAB         T4: I/O := A/B
0118 00112  370 002247            RTN        PASS IOO   CAB         T5: I/O := A/B
0119 00113  320 012005  JTBL1000  JMP  J74              EM1000
0120                        *
0121 00114  230 036777  CONTROL   READ IOFF                          T3: READ
0122 00115  336 103642            JMP  CNDX SKPF        RET         T4: TEST FOR SKIP FLAG
0123 00116  363 003642            RTN  CNDX RUN                     T5:
0124 00117  320 000307            JMP                   HORI        T6: TEST FOR HALT INSTRUCTION
```

```
PAGE 0009 RTE MICRO-ASSEMBLER REV.A 760818
0126                        *
0127                        *        EXTENDED ARITHMETIC GROUP
0128                        *        -------------------------
0129                        *
0130 00120  230 036747  DLD      READ                            READ
0131 00121  300 012447           JSB                   INDIRECT
0132 00122  007 133007                    INC  S1   M           S1 := M+1
0133 00123  010 000147                    PASS A    TAB         A := T/A/B
0134 00124  230 040647           READ     PASS M    S1          M := S1; READ
0135 00125  010 000207                    PASS B    TAB         B := T/A/B:= P+1
0136 00126  227 174707           READ     INC  PNM  P           M := P/ P := P+1  READ
0137 00127  370 036747           RTN
0138                        *
0139 00130  230 036747  DST      READ                            READ
0140 00131  300 012447           JSB                   INDIRECT
0141 00132  210 006036           WRTE MPCK PASS TAB    A         T/A/B := A; WRITE
0142 00133  007 133007                    INC  S1   M           S1 := M + 1
0143 00134  010 040647                    PASS M    S1          M := S1
0144 00135  210 002036  ST*      WRTE MPCK PASS TAB    CAB       T/A/B=A/B;WRITE;ENTRY FOR STA,STB
0145 00136  227 174707           READ     INC  PNM  P           M := P; P := P+1; READ
0146 00137  370 036747           RTN
0147                        *
0148 00140  010 036753  MPY           COV
0149 00141  300 012447           JSB                   INDIRECT
0150 00142  257 107007           ENV      CMPS S1   A           S1=MULTIPLICAND; NSIGN IN OVFL
0151 00143  010 000507                    PASS L    TAB         LOAD L WITH MULTIPLIER
0152 00144  006 036227                RPT ZERO B                CLEAR B. INITIATE REPEAT STEP
0153 00145  163 010224           MPY  R1   ADD  B    B         REPEAT MULTIPLY STEP 16 TIMES
0154 00146  315 146502           JSB  CNDX OVFL RJS  *+4       SUBTRACT IF MULTIPLICAND NEGATIVE
0155 00147  227 174713           READ COV  INC  PNM  P         M := P; P := P+1; READ
0156 00150  362 141702           RTN  CNDX L15  RJS            TEST FOR POSITIVE MULTIPLIER
0157 00151  237 140507           READ     CMPS L    S1          PLACE MULTIPLICAND IN L
0158 00152  364 110207           RTN      SUB  B    B           SUBTRACT FOR NEGATIVE MULTIPLIER
PAGE 0010 RTE MICRO-ASSEMBLER REV.A 760818
0160 00153  237 110516  DIV      READ CLFL CMPS L    B         L := NDIVIDENDHI; READ
0161 00154  300 012447           JSB                   INDIRECT
0162 00155  017 101007                    CMPS S1   TAB         S1 := NDIVISOR
0163 00156  014 141047                    XOR  S2   S1          EXPECTED QUOTIENT SIGN IN S2
0164 00157  322 107202           JMP  CNDX L15         DIVS    TEST FOR NEGATIVE DIVIDEND
0165 00160  017 110217                STFL CMPS B    B
0166 00161  017 106147                    CMPS A    A           MAKE
0167 00162  007 106147                    INC  A    A           DIVIDEND
0168 00163  301 010342           JSB  CNDX COUT        RMDR+2  POSITIVE
0169 00164  017 140507  DIVS              CMPS L    S1
0170 00165  327 147342           JMP  CNDX AL15 RJS  *+2       TEST FOR POSITIVE DIVISOR
0171 00166  007 140507                    INC  L    S1          L := ABSOLUTE VALUE OF DIVISOR
0172 00167  004 110754                SOV  SUB       B
0173 00170  327 143642           JMP  CNDX AL15 RJS  RET       TEST FOR DIVISOR TOO SMALL
0174 00171  070 010222           LGS  L1   PASS B    B         PRESHIFT THE DIVIDEND
0175 00172  007 174727                RPT  INC  PNM  P          M := P; P := P+1
0176 00173  124 110222           DIV  L1   SUB  B    B         REPEAT DIVIDE STEP 16 TIMES
0177 00174  010 010224                R1   PASS B    B         REMAINDER := B/2
0178 00175  237 107013           READ COV  CMPS S1   A         S1 := NQUOTIENT
0179 00176  320 110242           JMP  CNDX ONES        RMDR    TEST FOR ZERO QUOTIENT
0180 00177  010 042507                    PASS L    S2
0181 00200  327 150102           JMP  CNDX AL15 RJS  *+2       TEST FOR EXPECTED QUOTIENT SIGN
0182 00201  007 140147                    INC  A    S1          COMPLEMENT QUOTIENT
0183 00202  234 106747           READ     XOR       A           COMPARE QUOTIENT
0184 00203  327 150242           JMP  CNDX AL15 RJS  RMDR       WITH EXPECTED SIGN
0185 00204  230 036754           READ SOV
0186 00205  374 040742  RMDR     RTN  CNDX FLAG RJS            TEST EXPECTED SIGN OF REMAINDER
0187 00206  237 110207           READ     CMPS B    B           BEGIN 2'S COMPLEMENT OF REMAINDER
0188 00207  367 110207           RTN      INC  B    B           COMPLETE TWOS COMPLEMENT
```

```
PAGE 0011 RTE MICRO-ASSEMBLER REV.A 760818
0190 00210  010 036753  ASL            COV
0191 00211  010 036767                 RPT
0192 00212  030 010222           ARS   L1    PASS B    B          ARITHMETIC LEFT SHIFT
0193 00213  230 036740           READ  RTN                        READ
0194                     *
0195 00214  030 010224  ASR      ARS   R1    PASS B    B          ARITHMETIC SHIFT RIGHT
0196 00215  230 036753           READ  COV                        READ
0197 00216  370 036747           RTN
0198                     *
0199 00217  070 010222  LSL      LGS   L1    PASS B    B          LOGICAL LEFT SHIFT
0200 00220  230 036740           READ  RTN                        READ
0201                     *
0202 00221  070 010224  LSR      LGS   R1    PASS B    B          LOGICAL RIGHT SHIFT
0203 00222  230 036740           READ  RTN                        READ
0204                     *
0205 00223  050 010222  RRL      CRS   L1    PASS B    B          ROTATE LEFT
0206 00224  230 036740           READ  RTN                        READ
0207                     *
0208 00225  050 010224  RRR      CRS   R1    PASS B    B          ROTATE RIGHT
0209 00226  230 036740           READ  RTN                        READ
0210                     *
0211 00227  320 013005  JTBL1010 JMP   J74             EM1010
0212                     *
PAGE 0012 RTE MICRO-ASSEMBLER REV.A 760818
0214 00230  323 112742  TIMER    JMP   CNDX HOI       INDIRECT+6 TEST FOR HALT OR INTERRUPT
0215 00231  007 110207                    INC  B       B          INCREMENT B
0216 00232  320 051402           JMP   CNDX ALZ  RJS  *-2        TEST FOR ZERO
0217 00233  230 036740           READ  RTN
0218                     *
0219 00234  323 011542  DIAG     JMP   CNDX RUN       TIMER+3    ABORT TEST IF IN RUN MODE
0220 00235  300 032607           JSB                  MEMLOST+1  TEST CPU, 1 MEGAWORD MEMORY
0221 00236  325 051642           JMP   CNDX RUNE RJS  *-1        LOOP IF SWITCH IS (LOCK+RUNE)
0222 00237  320 013247           JMP                  HALT
0223                     *
0224                     *       EAU/MACTABLE 1 000 000 0
0225                     *       ------------
0226                     *
0227 00240  320 011617  EM1000   JMP   STFL          DIAG       00 00
0228 00241  320 010407           JMP                  ASL        00 01
0229 00242  320 010767           JMP   RPT            LSL        00 10
0230 00243  320 011407           JMP                  TIMER      00 11
0231 00244  320 011167           JMP   RPT            RRL        01 00
0232 00245  300 061107  FPDIAG   JSB                  RETNFP     01 01 MOD 3 TEST POINT
0233 00246  230 036740           READ  RTN                       01 10
0234 00247  230 036740           READ  RTN                       01 11
0235 00250  320 006004           JMP   RJ30           MPY        10 00
0236                     *
0237                     *       UNIVERSAL INDIRECT OPERAND ROUTINE
0238                     *       ---------------------------------
0239                     *
0240 00251  230 000674  INDIRECT READ  INCI PASS M    TAB        M := T/A/B; READ
0241 00252  367 140002           RTN   CNDX AL15 RJS            TEST FOR MORE INDIRECTS
0242 00253  230 036774           READ  INCI
0243 00254  323 152442           JMP   CNDX HOI  RJS  INDIRECT   TEST FOR HALT OR INTERRUPT
0244 00255  336 052442           JMP   CNDX NSNG RJS  INDIRECT   TEST FOR INSTRUCTION STEP
0245 00256  337 100302           JMP   CNDX MRG       HORI       TEST FOR JMP,I OR JSB,I
0246 00257  000 075707                    DEC  P       P          DECREMENT P
0247                     *
0248                     *       EAU/MACTABLE 1 000 001 0
0249                     *       ------------
0250                     *
0251 00260  320 000307  EM1010   JMP                  HORI       HALT OR INTERRUPT PENDING
0252 00261  320 010627           JMP   RPT            ASR        00 01
0253 00262  320 011067           JMP   RPT            LSR        00 10
0254 00263  230 036740           READ  RTN                       00 11
0255 00264  320 011267           JMP   RPT            RRR        01 00
```

```
PAGE 0013 RTE MICRO-ASSEMBLER REV.A 760818
0257                        *
0258                        *        FRONT PANEL ROUTINES
0259                        *        --------------------
0260                        *
0261 00265  334 013342  HALT     JMP  CNDX FLAG      *+2
0262 00266  000 075007                    DEC  S1   P
0263 00267  010 040647                    PASS M    S1
0264 00270  305 172542          JSB  CNDX NMLS RJS  MEMLOST   TEST FOR COLD POWER UP
0265 00271  017 135756               CLFL CMPS S    DES       S := DESCRIPTOR BLOCK
0266 00272  323 015242          JMP  CNDX RUN       RUN       TEST FOR AUTO-RESTART
0267 00273  327 153702          JMP  CNDX AL15 RJS  *+3       TEST FOR SWITCH 15
0268 00274  327 024542          JMP  CNDX NSFP      RPL       TEST FOR NO FRONT PANEL
0269 00275  325 064542          JMP  CNDX RUNE RJS  RPL       TEST LOCK POSTION OF POWER SWITCH
0270 00276  327 021742          JMP  CNDX NSFP      USER      USER FRONT PANEL MODULE
0271 00277  010 015747                    PASS S    DSPL      S := DSPL REGISTER
0272 00300  006 037107                    ZERO S3             CLEAR DMS MAP POINTER
0273 00301  336 054142          JMP  CNDX NSNG RJS  WAIT      TEST FOR INSTRUCTION STEP
0274 00302  343 156347          IMM       LOW  DSPI 367B      MAKE DSPL INDICATOR=T-REGISTER
0275                        *
0276 00303  300 023707  WAIT     JSB                 DSPICODE  BINARY ENCODE OF DSPL INDICATOR
0277 00304  300 022004          JSB  RJ30           UPDATES   UPDATE DSPL REGISTER
0278 00305  330 154242          JMP  CNDX NSTB RJS  *         WAIT FOR BUTTON TO BE RELEASED
0279 00306  006 036774               INCI ZERO
0280 00307  001 136741               JTAB DBLS                 INITIALIZE SAVE STACK
0281 00310  330 114402  IDLE     JMP  CNDX NSTB      *         WAIT FOR BUTTON TO BE PRESSED
0282                        *
0283 00311  300 014547  JSBSCAN  JSB                 SCAN      GO TO SCAN SUBROUTINE
0284 00312  320 014147          JMP                 WAIT

PAGE 0014 RTE MICRO-ASSEMBLER REV.A 760818
0286 00313  007 133047  SCAN              INC  S2   M         S2 := M+1
0287 00314  332 156102          JMP  CNDX NLT  RJS  LEFT      LEFT
0288 00315  332 056602          JMP  CNDX NRT  RJS  RIGHT     RIGHT
0289 00316  331 057442          JMP  CNDX NINC RJS  INCM      INC M
0290 00317  331 157342          JMP  CNDX NDEC RJS  DECM      DEC M
0291 00320  330 064702          JMP  CNDX NLDR RJS  LOADER    IBL/TEST
0292 00321  333 057642          JMP  CNDX NSTR RJS  STORE     STORE
0293 00322  333 155702          JMP  CNDX NMDE RJS  MODE      MODE
0294 00323  336 055302          JMP  CNDX NSNG RJS  INSTP     INSTRUCTION STEP
0295 00324  323 054242          JMP  CNDX RUN  RJS  WAIT+2    PRESET
0296                        *
0297 00325  343 076356  RUN      IMM  CLFL LOW  DSPI 337B      MAKE DSPL INDICATOR=S-REGISTER
0298 00326  314 015702  INSTP    JSB  CNDX FLAG      MODE      TEST FOR INVERSE VIDEO
0299 00327  227 174710          READ SRUN INC  PNM  P         M := P; P := P+1; READ
0300 00330  010 076307                    PASS DSPL S         PLACE S IN DSPL REGISTER
0301 00331  010 001007                    PASS S1   TAB       S1 := T/A/B
0302 00332  230 040633          READ FTCH PASS IRCM S1        IR = S1;M = OPERAND ADDRESS;READ
0303 00333  336 000042          JMP  CNDX NSNG      FETCH+1   TEST FOR NOT SINGLE INSTRUCTION
0304 00334  010 040775               SHLT PASS      S1
0305 00335  320 000047          JMP                 FETCH+1   COMPLETE FETCH
0306                        *
0307 00336  017 117007  MODE              CMPS S1   DSPI      S1 := COMPLEMENTED INDICATOR BITS
0308 00337  334 016042          JMP  CNDX FLAG      *+2
0309 00340  370 040357          RTN  STFL PASS DSPI S1        REVERSE INDICATOR BITS;COMP. FLAG
0310 00341  370 040356          RTN  CLFL PASS DSPI S1        REVERSE INDICATOR BITS;COMP. FLAG
```

```
PAGE 0015 RTE MICRO-ASSEMBLER REV.A 760818
0312                        *
0313 00342  010 017024  LEFT          R1   PASS S1   DSPI      SHIFT DSPL INDICATOR
0314 00343  334 016342          JMP   CNDX FLAG      *+4       TEST FOR REVERSE DISPLAY MODE
0315 00344  321 156302          JMP   CNDX AL0  RJS  *+2       TEST FOR WRAP-AROUND
0316 00345  370 040347          RTN        PASS DSPI S1
0317 00346  343 076340          IMM   RTN  LOW  DSPI 337B      PLACE S-POINTER IN DSPL INDICATOR
0318 00347  321 156442          JMP   CNDX AL0  RJS  *+2       TEST FOR WRAP-AROUND
0319 00350  340 100340          IMM   RTN  LOW  DSPI 040B      PLACE S-POINTER IN DSPL INDICATOR
0320 00351  340 176507          IMM        LOW  L    077B      L := 77
0321 00352  012 041007                     AND  S1   S1        MASK DSPL INDICATOR
0322 00353  370 040347          RTN        PASS DSPI S1
0323 00354  334 017102  RIGHT   JMP   CNDX FLAG      *+6       TEST FOR REVERSE DISPLAY MODE
0324 00355  342 176517          IMM   STFL LOW  L    277B      L := 177677
0325 00356  150 017022          LWF   L1   PASS S1   DSPI      SHIFT DSPL INDICATOR LEFT 1 PLACE
0326 00357  010 140356                CLFL IOR  DSPI S1
0327 00360  360 101002          RTN   CNDX ONES                 TEST FOR NO WRAP-AROUND
0328 00361  343 174340          IMM   RTN  LOW  DSPI 376B      PLACE A-POINTER IN DSPL INDICATOR
0329 00362  010 017022                L1   PASS S1   DSPI
0330 00363  352 176507          IMM        CMLO L    277B      L := 100
0331 00364  012 040347                     AND  DSPI S1        MASK DSPL INDICATOR
0332 00365  360 001002          RTN   CNDX ALZ                  TEST FOR NO WRAP-AROUND
0333 00366  340 002340          IMM   RTN  LOW  DSPI 001B      PLACE X-POINTER IN DSPL INDICATOR
0334                        *
0335 00367  334 017542  DECM    JMP   CNDX FLAG      DECDMS    TEST FOR REVERSE DISPLAY MODE
0336 00370  000 033047                     DEC  S2   M         DECREMENT M
0337 00371  334 017602  INCM    JMP   CNDX FLAG      INCDMS    TEST FOR REVERSE DISPLAY MODE
0338 00372  370 042647          RTN        PASS M    S2
0339                        *
0340 00373  360 045107  DECDMS  RTN        DEC  S3   S3        DECREMENT DMS MAP POINTER
0341 00374  367 145107  INCDMS  RTN        INC  S3   S3        INCREMENT DMS MAP POINTER
0342                        *
0343 00375  300 023707  STORE   JSB                  DSPICODE  BINARY ENCODE OF DSPL INDICATOR
0344 00376  300 020004          JSB   RJ30           STORES    STORE DSPL REGISTER
0345 00377  320 014147          JMP                  WAIT
0346                        *

PAGE 0016 RTE MICRO-ASSEMBLER REV.A 760818
0348                            ALGN
0349                            ORG                  400B
0350 00400  370 015747  STORES  RTN        PASS S    DSPL      S := DSPL REGISTER
0351 00401  370 015707  STOREP  RTN        PASS P    DSPL      P := DSPL REGISTER
0352 00402  320 021607          JMP                  STORET
0353 00403  370 014647  STOREM  RTN        PASS M    DSPL      M := DSPL REGISTER
0354 00404  370 014207  STOREB  RTN        PASS B    DSPL      B := DSPL REGISTER
0355 00405  370 014147  STOREA  RTN        PASS A    DSPL      A := DSPL REGISTER
0356 00406  320 021007  STOREST JMP                  STCPUS
0357 00407  320 020607  STOREF  JMP                  STFENCE
0358 00410  370 014452  STOREMM RTN   MESP PASS MEU  DSPL      DMS MAP DATA := DSPL REGISTER
0359 00411  370 015107  STOREMN RTN        PASS S3   DSPL      DMS MAP NUMBER := DSPL REGISTER
0360 00412  370 015647  STOREY  RTN        PASS Y    DSPL      Y := DSPL REGISTER
0361 00413  370 015607  STOREX  RTN        PASS X    DSPL      X := DSPL REGISTER
0362 00414  344 016507  STFENCE IMM        HIGH L    007B      L := 003777
0363 00415  012 015007                     AND  S1   DSPL      MASK DSPL REGISTER
0364 00416  010 022447                     PASS MEU  MEU
0365 00417  370 040447          RTN        PASS MEU  S1        STORE INTO DMS FENCE
0366 00420  010 033153  STCPUS        COV  PASS S4   M         SAVE M
0367 00421  010 015022                L1   PASS S1   DSPL
0368 00422  327 161202          JMP   CNDX AL15 RJS  *+2       TEST FOR DSPL 14
0369 00423  010 036754                SOV
0370 00424  342 000607          IMM        LOW  IRCM 200B      SET UP ELB INSTRUCTION
0371 00425  010 041021                SRG1 PASS S1   S1        STORE DSPL 14 INTO EXTEND
0372 00426  352 177047          IMM        CMLO S2   277B      S2 := STF 0 INSTRUCTION
0373 00427  010 040747                     PASS      S1
0374 00430  327 121502          JMP   CNDX AL15      *+2       TEST FOR INTERRUPT SYSTEM
0375 00431  357 173047          IMM        CMHI S2   375B      S2 := CLF 0 INSTRUCTION
0376 00432  010 042606                IOG  PASS IRCM S2
0377 00433  370 046647          RTN        PASS M    S4        RESTORE M
0378 00434  210 014007  STORET  WRTE       PASS TAB  DSPL      T := DSPL REGISTER
0379 00435  010 042647                     PASS M    S2        INCREMENT M
0380 00436  320 014247          JMP                  WAIT+2    DO NOT UPDATE DSPL
0381                        *
0382 00437  325 040007  USER    JMP                  25000B    JUMP TO USER FRONT PANEL ROUTINE
```

```
PAGE 0017 RTE MICRO-ASSEMBLER REV.A 760818
0384                                ORG                 440B
0385 00440  370 076307  UPDATES  RTN       PASS DSPL S         DSPL REGISTER := S
0386 00441  370 074307  UPDATEP  RTN       PASS DSPL P         DSPL REGISTER := P
0387 00442  370 000307  UPDATET  RTN       PASS DSPL TAB       DSPL REGISTER := T
0388 00443  370 032307  UPDATEM  RTN       PASS DSPL M         DSPL REGISTER := M
0389 00444  370 010307  UPDATEB  RTN       PASS DSPL B         DSPL REGISTER := B
0390 00445  370 006307  UPDATEA  RTN       PASS DSPL A         DSPL REGISTER := A
0391 00446  320 022707  UPDATEST JMP                   UPDCPUS
0392 00447  320 022607  UPDATEF  JMP                   UPDFENCE
0393 00450  370 022312  UPDATEMM RTN  MESP PASS DSPL MEU       DSPL REGISTER := DMS MAP DATA
0394 00451  370 044307  UPDATEMN RTN       PASS DSPL S3        DSPL REGISTER := DMS MAP NUMBER
0395 00452  370 072307  UPDATEY  RTN       PASS DSPL Y         DSPL REGISTER := Y
0396 00453  370 070307  UPDATEX  RTN       PASS DSPL X         DSPL REGISTER := X
0397 00454  010 022447  UPDFENCE           PASS MEU  MEU
0398 00455  370 022307          RTN       PASS DSPL MEU       DSPL REG=DMS STATUS/FENCE REG
0399 00456  355 176507  UPDCPUS IMM       CMHI L    177B      L := 100000
0400 00457  010 033047                    PASS S2   M         SAVE M
0401 00460  350 177007          IMM       CMLO S1   077B      S1 := 000300  SFS 0
0402 00461  010 040606               IOG  PASS IRCM S1        IR := SFS 0
0403 00462  006 037007                    ZERO S1             INITIALIZE CPU STATUS WORD
0404 00463  336 163242          JMP  CNDX SKPF RJS  *+2       TEST FOR INTERRUPT SYSTEM ON
0405 00464  003 041024               R1   ADD  S1   S1        S1 := 040000
0406 00465  334 163342          JMP  CNDX E    RJS  *+2       TEST FOR EXTEND SET
0407 00466  003 041007                    ADD  S1   S1
0408 00467  010 041024               R1   PASS S1   S1
0409 00470  335 163502          JMP  CNDX OVFL RJS  *+2       TEST FOR OVERFLOW SET
0410 00471  003 041007                    ADD  S1   S1
0411 00472  010 024507                    PASS L    CIR       L := CIR
0412 00473  010 141007                    IOR  S1   S1        MERGE IN CIR
0413 00474  010 042647                    PASS M    S2        RESTORE M
0414 00475  370 040307          RTN       PASS DSPL S1        DSPL := E,O,I, AND CIR
0415                        *
0416 00476  343 164547  DSPICODE IMM      LOW  CNTR 372B      CNTR := 000372
0417 00477  017 117023               L4   CMPS S1   DSPI      S1 := NDSPI SHIFTED LEFT FOUR
0418 00500  334 064202          JMP  CNDX FLAG RJS  *+4       TEST FOR NO REVERSE DISPLAY MODE
0419 00501  340 000547          IMM       LOW  CNTR 000B      CNTR := 000
0420 00502  344 000507          IMM       HIGH L    000B      L := 000377
0421 00503  013 017023               L4   XNOR S1   DSPI
0422 00504  001 141026               ICNT DBLS S1   S1        LEFT SHIFT S1; INCREMENT COUNTER
0423 00505  327 164202          JMP  CNDX AL15 RJS  *-1       TEST FOR INDICATOR BIT
0424 00506  352 000507          IMM       CMLO L    200B      L := 177
0425 00507  012 045107                    AND  S3   S3        MASK DMS MAP POINTER
0426 00510  357 076507          IMM       CMHI L    337B      L := 020000
0427 00511  230 145007          READ      IOR  S1   S3        MERGE DMS CONTROL BIT
0428 00512  370 040447          RTN       PASS MEU  S1        LOAD DMS MAP ADDRESS REGISTER
0429                        *
0430 00513  347 000447  RPL     IMM       HIGH MEU  300B      DISABLE DMS MAPS
0431 00514  300 024707          JSB                 LOADER    GO TO LOADER SUBROUTINE
0432 00515  320 015247          JMP                 RUN       REMOTE PROGRAM LOAD
```

```
PAGE 0018 RTE MICRO-ASSEMBLER REV.A 760818
0434                        *
0435                        *        INITIAL BINARY LOADER
0436                        *        ---------------------
0437                        *
0438 00516  345 177014  LOADER   IMM  SOV  HIGH S1   177B      S1 := 077777
0439 00517  010 076607                     PASS IRCM S         IR = S TO SET UP LOADER SELECTION
0440 00520  343 000507  MEMSIZE  IMM       LOW  L    300B      L  := 177700
0441 00521  012 040707                     AND  PNM  S1        M  := S1; P := S1 AND L
0442 00522  367 101002           RTN  CNDX AL15                TEST FOR NO READ/WRITE CAPABILITY
0443 00523  210 074007           WRTE      PASS TAB  P         WRITE INTO MEMORY
0444 00524  357 136507           IMM       CMHI L    357B      L  := 010000
0445 00525  224 141007           READ      SUB  S1   S1        READ BACK FROM MEMORY
0446 00526  010 074507                     PASS L    P         L  := WRITTEN DATA
0447 00527  014 100747                     XOR       TAB       COMPARE
0448 00530  320 065002           JMP  CNDX ALZ  RJS  MEMSIZE   TEST FOR PRESENT MEMORY
0449 00531  010 075007                     PASS S1   P         S1 := P
0450                        *
0451 00532  350 007063  SELCODE  IMM  L4   CMLO S2   003B      S2 := 007700
0452 00533  010 042507                     PASS L    S2
0453 00534  340 014547           IMM       LOW  CNTR 006B      COUNTER := 6
0454 00535  012 077067                R1   AND  S2   S         MASK SELECT CODE
0455 00536  010 043064                R1   PASS S2   S2        SHIFT SELECT CODE 6 PLACES RIGHT
0456 00537  353 156507           IMM       CMLO L    367B      L  := 000010
0457 00540  004 143071                IAK  SUB  S2   S2        S2 := SELECT CODE -10,SYNC TO T6
0458 00541  367 101042           RTN  CNDX AL15                TEST FOR SELECT CODE LESS THAN 10
0459                        *
0460 00542  010 040647  LOOP               PASS M    S1
0461 00543  010 031023                L4   PASS S1   LDR
0462 00544  010 040526                ICNT PASS L    S1        THE FIRST PART OF THIS LOOP
0463 00545  012 031023                L4   AND  S1   LDR       ROUTINE PACKS EACH FOUR BIT
0464 00546  010 040526                ICNT PASS L    S1        SEGMENT FROM THE SPECIFIED
0465 00547  012 031023                L4   AND  S1   LDR       LOADER ROM INTO A 16-BIT WORD
0466 00550  010 040526                ICNT PASS L    S1
0467 00551  015 131013                COV  NAND S1   LDR       T5 ON FIRST PASS.LDR ->> PRESET

PAGE 0019 RTE MICRO-ASSEMBLER REV.A 760818
0469 00552  354 026526           IMM  ICNT CMHI L    013B      L := 172000
0470 00553  012 041107                     AND  S3   S1
0471 00554  345 166507           IMM       HIGH L    173B      L := 075777
0472 00555  013 044747                     XNOR      S3
0473 00556  320 067402           JMP  CNDX ALZ  RJS  STWORD    TEST FOR I/O INSTRUCTION
0474 00557  350 077122           IMM  L1   CMLO S3   037B      S3 := 000700
0475 00560  010 044507                     PASS L    S3
0476 00561  012 040747                     AND       S1
0477 00562  320 027402           JMP  CNDX ALZ       STWORD    TEST FOR HALT INSTRUCTION
0478 00563  353 016507           IMM       CMLO L    307B      L := 000070
0479 00564  012 040747                     AND       S1
0480 00565  320 027402           JMP  CNDX ALZ       STWORD    TEST FOR SELECT CODE LESS THAN 10
0481 00566  010 042507                     PASS L    S2
0482 00567  003 041007                     ADD  S1   S1        PATCH IN CONFIGURING SELECT CODE
0483 00570  210 040007  STWORD   WRTE      PASS TAB  S1        WRITE WORD INTO MEMORY
0484 00571  007 133007                     INC  S1   M
0485 00572  326 166102           JMP  CNDX CNT8 RJS  LOOP      TEST FOR LOADER COMPLETION
0486 00573  017 175007                     CMPS S1   P         TWOS COMPLEMENT LAST AVAILABLE
0487 00574  007 141007                     INC  S1   S1        WORD OF PROGRAM MEMORY AND
0488 00575  210 040007           WRTE      PASS TAB  S1        STORE INTO LAST LOADER ADDRESS
0489 00576  000 033007                     DEC  S1   M
0490 00577  230 040647           READ      PASS M    S1
0491 00600  010 042507                     PASS L    S2        PATCH SELECT CODE INTO
0492 00601  003 001007                     ADD  S1   TAB       PORT CONTROLLER WORD 1
0493 00602  210 040007           WRTE      PASS TAB  S1        STORE PORT CONTROLLER  WORD 1
0494 00603  300 030747           JSB                 CPTEST    PERFORM QUICK PROCESSOR TEST
```

```
PAGE 0020 RTE MICRO-ASSEMBLER REV.A 760818
0496                        *
0497                        *        FIRMWARE DIAGNOSTICS
0498                        *        --------------------
0499                        *
0500 00604  220 033007  TEST32K  READ       DEC  S1    M          S1 := M - 1; READ MEMORY WORD
0501 00605  360 000642           RTN  CNDX  ALZ                   CHECK FOR TEST COMPLETION
0502 00606  017 101047                      CMPS S2    TAB        S2 := COMPLEMENTED DATA
0503 00607  210 042007           WRTE       PASS TAB   S2         T/A/B := COMPLEMENTED DATA; WRITE
0504 00610  230 042507           READ       PASS L     S2         L := COMPLEMENTED DATA
0505 00611  017 143047                      CMPS S2    S2         S2 := ORIGINAL DATA
0506 00612  014 100747                      XOR        TAB        COMPARE
0507 00613  320 076502           JMP  CNDX  ALZ   RJS  FAILURE    TEST FOR MEMORY FAILURE
0508 00614  210 042007           WRTE       PASS TAB   S2         T/A/B=ORIG. DATA;RESTORE MEMORY
0509 00615  010 040647                      PASS M     S1
0510 00616  320 030207           JMP                   TEST32K
0511                        *
0512 00617  343 053022  CPTEST   IMM  L1    LOW  S1    325B       S1 := 177652
0513 00620  346 124507           IMM        HIGH L     252B       L  := 125377
0514 00621  012 041016                CLFL AND  S1    S1         S1 := 125252
0515 00622  300 031207           JSB                   REGTEST
0516 00623  010 043017                STFL PASS S1    S2         S1 := 052525
0517                        *
0518 00624  017 141054  REGTEST       SOV  CMPS S2    S1         S2 = NS1   THIS ROUTINE LOADS
0519 00625  010 043107                     PASS S3    S2         S3 =  S2   THE SCRATCH REGISTERS
0520 00626  150 045162           LWF  L1    PASS S4    S3         S4 = NS3   WITH ONE OF TWO
0521 00627  150 047224           LWF  R1    PASS S5    S4         S5 = NS4   COMPLEMENTARY DATA
0522 00630  017 151263                L4   CMPS S6    S5         S6 = NS5   PATTERNS.  REGISTERS
0523 00631  157 153322           LWF  L1    CMPS S7    S6         S7 =  S6   WITH 1 BIT DIFFER.
0524 00632  150 055364           LWF  R1    PASS S8    S7         S8 = NS7   IN ADDRESS ARE FILLED
0525 00633  010 057423                L4   PASS S9    S8         S9 =  S8   WITH UNLIKE PATTERNS.
0526 00634  017 161447                     CMPS S10   S9         S10= NS9   THE ROTATE/SHIFT AND
0527 00635  010 063507                     PASS S11   S10        S11=  S10  FLAG LOGIC IS CHECKED.
0528 00636  010 050507                     PASS L     S5         L := OTHER TEST PATTERN
0529 00637  014 156756                CLFL XOR        S8         XOR SAME PATTERN
0530 00640  320 076602           JMP  CNDX  ALZ   RJS  FAILURE+2 TEST FOR NON-ZEROS
0531 00641  013 060747                      XNOR       S9         XNOR SAME PATTERN
0532 00642  320 176602           JMP  CNDX  ONES  RJS  FAILURE+2 TEST FOR NON-ONES
0533 00643  014 154747                      XOR        S7         XOR DIFFERENT PATTERN
0534 00644  320 176602           JMP  CNDX  ONES  RJS  FAILURE+2 TEST FOR NON-ONES
0535 00645  013 062747                      XNOR       S10        XNOR DIFFERENT PATTERN
0536 00646  320 076602           JMP  CNDX  ALZ   RJS  FAILURE+2 TEST FOR NON-ZEROS
0537 00647  003 064747                      ADD        S11        ADD UNLIKE PATTERNS
0538 00650  321 036602           JMP  CNDX  COUT       FAILURE+2 TEST FOR CARRY OUT
0539 00651  320 110642           JMP  CNDX  ONES       ASR+1     TEST FOR NON-ONES
0540 00652  320 036607           JMP                   FAILURE+2
```

```
PAGE 0021 RTE MICRO-ASSEMBLER REV.A 760818
0542 00653  006 037747  MEMLOST            ZERO S                CLEAR DISPLAY ON POWER UP
0543 00654  300 030747          JSB                   CPTEST     TEST CENTRAL PROCESSOR
0544                    *
0545 00655  006 037253  RIPP1MW       COV  ZERO S6               CLEAR S6
0546 00656  010 077207                     PASS S5   S           SAVE S
0547 00657  010 052307                     PASS DSPL S6          CLEAR DISPLAY REGISTER
0548 00660  010 075307                     PASS S7   P           SAVE P
0549 00661  340 100547  DMSLOAD  IMM       LOW  CNTR 040B        COUNTER := 40
0550 00662  357 077047          IMM       CMHI S2   337B        S2 := 020000
0551 00663  345 004447          IMM       HIGH MEU  102B        ENABLE SYSTEM MAP
0552 00664  010 042447                     PASS MEU  S2          CLEAR DMS ADDRESS REGISTER
0553 00665  010 052452               MESP  PASS MEU  S6          LOAD MAP
0554 00666  007 153265               DCNT  INC  S6   S6          INCREMENT MAP ADDRESS
0555 00667  326 173242          JMP  CNDX CNT8 RJS  *-2          TEST FOR ALL MAPS LOADED
0556 00670  353 077747          IMM       CMLO S    337B        PASS LOADER INVALID SC.,SET IR
0557 00671  300 024707          JSB                 LOADER        FIND HOW MUCH MEMORY AVAILABLE
0558 00672  010 015761               SRG1  PASS S    DSPL        RESTORE S. CLEAR EXTEND
0559 00673  010 033107                     PASS S3   M           S3 := TOP OF ENABLED MEMORY
0560 00674  322 134402          JMP  CNDX L15       TESTDMS       TEST FOR PRESENT MEMORY
0561 00675  353 175047          IMM       CMLO S2   376B        BACKGROUND PATTERN := 000001
0562 00676  343 154147          IMM       LOW  A    366B        TEST PATTERN := 177766
0563 00677  300 035107          JSB                 RIPP32K       TEST #1
0564 00700  355 177047          IMM       CMHI S2   177B        BACKGROUND PATTERN := 100000
0565 00701  353 154147          IMM       CMLO A    366B        TEST PATTERN := 000011
0566 00702  300 035107          JSB                 RIPP32K       TEST #2
0567 00703  343 177047          IMM       LOW  S2   377B        BACKGROUND PATTERN := 177777
0568 00704  355 176147          IMM       CMHI A    177B        TEST PATTERN := 100000
0569 00705  300 035107          JSB                 RIPP32K       TEST #3
0570 00706  010 051047                     PASS S2   S5          BACKGROUND PATTERN := S5 (DSPL)
0571 00707  300 035107          JSB                 RIPP32K       TEST #4
0572 00710  010 022447  TESTDMS            PASS MEU  MEU
0573 00711  010 022747                     PASS      MEU         ENABLE MEM STATUS REGISTER
0574 00712  320 135002          JMP  CNDX ONES      *+6          TEST IF DMS IS PRESENT
0575 00713  007 115747                     INC  S    DSPL        S := DISPLAY REGISTER
0576 00714  353 076507          IMM       CMLO L    337B        L := 40
0577 00715  014 176307                     XOR  DSPL S           DISPLAY REGISTER := S
0578 00716  320 073042          JMP  CNDX ALZ  RJS  DMSLOAD     TEST FOR ALL MEMORY TESTED
0579 00717  345 000447          IMM       HIGH MEU  100B        DISABLE DMS MAPS
0580 00720  010 051775               SHLT PASS S    S5          RESTORE S
0581 00721  360 055707          RTN       DEC  P    S7          RESTORE P AND EXIT
PAGE 0022 RTE MICRO-ASSEMBLER REV.A 760818
0583 00722  000 044735  RIPP32K      SHLT DEC  PNM  S3
0584 00723  210 042007          WRTE      PASS TAB  S2          T/A/B := BACKGROUND PATTERN
0585 00724  000 074707                     DEC  PNM  P           M := P; P := P-1
0586 00725  327 175142          JMP  CNDX AL15 RJS  *-2          TEST FOR COMPLETE 32K
0587 00726  352 177147          IMM       CMLO S4   277B        S4 := 000100
0588 00727  010 046715               PRST  PASS PNM  S4          P := S4; M := S4
0589 00730  210 006007  RIPLOOP  WRTE      PASS TAB  A           T/A/B := TEST PATTERN; WRITE
0590 00731  352 174507          IMM       CMLO L    276B        L := 000101
0591 00732  223 075707          READ      ADD  P    P           P := P + 101
0592 00733  010 006515               PRST  PASS L    A           L := TEST PATTERN
0593 00734  014 100747                     XOR       TAB         COMPARE
0594 00735  320 076542          JMP  CNDX ALZ  RJS  FAILURE+1    TEST FOR SUCCESSFUL COMPARE
0595 00736  010 044515               PRST  PASS L    S3          L := TOP OF ENABLED MEMORY
0596 00737  210 042007          WRTE      PASS TAB  S2          T/A/B= BACKGROUND PATTERN RESTORE
0597 00740  004 174647                     SUB  M    P           TEST FOR NON-EXISTENT MEMORY
0598 00741  321 075402          JMP  CNDX COUT RJS  RIPLOOP     TEST FOR RIPPLE PASS COMPLETE
0599 00742  000 047147                     DEC  S4   S4          DECREMENT 32K COUNTER
0600 00743  327 175342          JMP  CNDX AL15 RJS  RIPLOOP-1    TEST FOR ENTIRE 32K TESTED
0601 00744  010 042507                     PASS L    S2          L := BACKGROUND PATTERN
0602 00745  010 044707                     PASS PNM  S3          P := TOP OF ENABLED MEMORY
0603 00746  220 074707  BKGNDCK  READ      DEC  PNM  P           M := P; P := P-1; READ
0604 00747  367 101702          RTN  CNDX AL15                   TEST FOR ENTIRE 32K READ
0605 00750  014 100747                     XOR       TAB         TEST AGAINST EXPECTED BACKGROUND
0606 00751  320 036302          JMP  CNDX ALZ       BKGNDCK      TEST FOR EXPECTED BACKGROUND
0607                    *
0608 00752  010 042147  FAILURE            PASS A    S2          A := EXPECTED DATA
0609 00753  010 000213               COV   PASS B    TAB         B := ACTUAL DATA
0610 00754  340 000375          IMM  SHLT LOW  DSPI 000B         SET ALL DISPLAY INDICATOR BITS
0611 00755  343 176307          IMM       LOW  DSPL 377B         SET ALL DISPLAY REGISTER BITS
0612 00756  327 054242          JMP  CNDX NSFP RJS  WAIT+2       SUSPEND TEST
0613 00757  320 021757          JMP  STFL           USER         FLAG --> TEST REPORTED ERROR
0614                    *
```

```
PAGE 0023 RTE MICRO-ASSEMBLER REV.A 760818
0616                                ORG                 760B
0617                        *
0618                        *       PRIMARY MAPPING TABLE
0619                        *       --------------------
0620                        *
0621 00760  324 161007  MACTABL1 JMP                    23420B    2000 ACCESS SYSTEM
0622 00761  324 161007          JMP                     23420B    2000 ACCESS SYSTEM
0623 00762  325 140004          JMP  RJ30               27000B
0624 00763  324 160004          JMP  RJ30               23400B    2000 ACCESS SYSTEM
0625 00764  325 160004          JMP  RJ30               27400B
0626 00765  326 000004          JMP  RJ30               30000B
0627 00766  326 020004          JMP  RJ30               30400B
0628 00767  326 040004          JMP  RJ30               31000B
0629 00770  327 000004          JMP  RJ30               34000B
0630 00771  327 020004          JMP  RJ30               34400B
0631 00772  327 040004          JMP  RJ30               35000B
0632 00773  327 060004          JMP  RJ30               35400B
0633 00774  324 000004          JMP  RJ30               20000B    DYNAMIC MAPPING SYSTEM
0634 00775  324 001004          JMP  RJ30               20020B    DYNAMIC MAPPING SYSTEM
0635 00776  320 041004          JMP  RJ30               EIG       EXTENDED INSTRUCTION GROUP
0636 00777  320 042004          JMP  RJ30               EIG+20B   EXTENDED INSTRUCTION GROUP
0637                        *
0638 01000  320 061557  MACTABL0 JMP  STFL              FAD       FLOATING POINT ADD
0639 01001  320 061547          JMP                     FSB       FLOATING POINT SUBTRACT
0640 01002  320 065107          JMP                     FMP       FLOATING POINT MULTIPLY
0641 01003  320 067047          JMP                     FDV       FLOATING POINT DIVIDE
0642 01004  320 060257          JMP  STFL               FIX       FLOATING POINT TO INTEGER
0643 01005  320 060007          JMP                     FLT       INTEGER TO FLOATING POINT
0644 01006  327 100004          JMP  RJ30               36000B
0645 01007  327 140004          JMP  RJ30               37000B
0646 01010  324 040004          JMP  RJ30               21000B    FAST FORTRAN
0647 01011  324 060004          JMP  RJ30               21400B    FAST FORTRAN
0648 01012  324 100004          JMP  RJ30               22000B    HP RESERVED
0649 01013  324 120004          JMP  RJ30               22400B    HP RESERVED
0650 01014  324 140004          JMP  RJ30               23000B    HP RESERVED
0651 01015  325 000004          JMP  RJ30               24000B    HP RESERVED
0652 01016  325 100004          JMP  RJ30               26000B    HP RESERVED
0653 01017  325 120004          JMP  RJ30               26400B    HP RESERVED

PAGE 0024 RTE MICRO-ASSEMBLER REV.A 760818
0655                        *
0656                        *       EXTENDED INSTRUCTION GROUP
0657                        *       --------------------------
0658                        *
0659 01020  320 043007  EIG      JMP                    S*X       SAX/SBX
0660 01021  370 003607           RTN       PASS X       CAB       CAX/CBX
0661 01022  320 043307           JMP                    L*X       LAX/LBX
0662 01023  320 043647           JMP                    STX       STX
0663 01024  370 070047           RTN       PASS CAB     X         CXA/CBX
0664 01025  320 044007           JMP                    LDX       LDX
0665 01026  320 044147           JMP                    ADX       ADX
0666 01027  320 044347           JMP                    X*X       XAX/XBX
0667 01030  320 045007           JMP                    S*Y       SAY/SBY
0668 01031  370 003647           RTN       PASS Y       CAB       CAY/CBY
0669 01032  320 045307           JMP                    L*Y       LAY/LBY
0670 01033  320 045607           JMP                    STY       STY
0671 01034  370 072047           RTN       PASS CAB     Y         CYA/CYB
0672 01035  320 045747           JMP                    LDY       LDY
0673 01036  320 046107           JMP                    ADY       ADY
0674 01037  320 046307           JMP                    X*Y       XAY/XBY
0675 01040  320 044507           JMP                    ISX       ISX
0676 01041  320 044647           JMP                    DSX       DSX
0677 01042  320 046747           JMP                    JLY       JLY
0678 01043  320 054107           JMP                    LBT       LBT
0679 01044  320 054407           JMP                    SBT       SBT
0680 01045  320 051507           JMP                    MBT       MBT
0681 01046  320 052147           JMP                    CBT       CBT
0682 01047  320 053107           JMP                    SFB       SFB
0683 01050  320 046447           JMP                    ISY       ISY
0684 01051  320 046607           JMP                    DSY       DSY
0685 01052  320 047147           JMP                    JPY       JPY
0686 01053  320 056707           JMP                    BITS      SBS
0687 01054  320 056707           JMP                    BITS      CBS
0688 01055  320 056707           JMP                    BITS      TBS
0689 01056  320 047407           JMP                    CMW       CMW
0690 01057  320 050747           JMP                    MVW       MVW
```

```
PAGE 0025 RTE MICRO-ASSEMBLER REV.A 760818
0692                        *
0693                        *        INDEX REGISTER GROUP
0694                        *        --------------------
0695                        *
0696 01060  300 012447  S*X      JSB                     INDIRECT
0697 01061  010 070507                   PASS L    X          L := X
0698 01062  003 033007                   ADD  S1   M
0699 01063  010 040647                   PASS M    S1         M := X + T/A/B
0700 01064  210 002036          WRTE MPCK PASS TAB  CAB        T/A/B := A/B; WRITE
0701 01065  320 043547          JMP                 RETURN
0702                        *
0703 01066  300 012447  L*X      JSB                     INDIRECT
0704 01067  010 070507                   PASS L    X          L := X
0705 01070  003 033007                   ADD  S1   M
0706 01071  230 040647          READ     PASS M    S1         M := X + T/A/B; READ
0707 01072  010 000047                   PASS CAB  TAB        A/B := T/A/B
0708 01073  227 174707  RETURN  READ     INC  PNM  P          M := P; P := P+1; READ
0709 01074  370 036747          RTN
0710                        *
0711 01075  300 012447  STX      JSB                     INDIRECT
0712 01076  210 070036          WRTE MPCK PASS TAB  X          T/A/B := X; WRITE
0713 01077  320 043547          JMP                 RETURN
0714                        *
0715 01100  300 012447  LDX      JSB                     INDIRECT
0716 01101  010 001607                   PASS X    TAB        X := T/A/B
0717 01102  227 174700          READ RTN INC  PNM  P          M := P; P := P+1; READ
0718                        *
0719 01103  300 012447  ADX      JSB                     INDIRECT
0720 01104  010 070507                   PASS L    X          L := X
0721 01105  263 001607          ENVE     ADD  X    TAB        X := X + T/A/B
0722 01106  227 174700          READ RTN INC  PNM  P          M := P; P := P+1; READ
0723                        *
0724 01107  230 002507  X*X      READ     PASS L    CAB        L := A/B
0725 01110  010 070047                   PASS CAB  X          A/B := X
0726 01111  372 137607          RTN      PASL X               X := L
0727                        *
0728 01112  227 171607  ISX      READ     INC  X    X          INCREMENT X; READ
0729 01113  360 041602          RTN  CNDX ALZ  RJS            TEST FOR ZERO
0730 01114  227 174700          READ RTN INC  PNM  P          M := P; P := P+1; READ
0731                        *
0732 01115  220 071607  DSX      READ     DEC  X    X          DECREMENT X; READ
0733 01116  360 041602          RTN  CNDX ALZ  RJS            TEST FOR ZERO
0734 01117  227 174700          READ RTN INC  PNM  P          M := P; P := P+1; READ
```

```
PAGE 0026 RTE MICRO-ASSEMBLER REV.A 760818
0736                        *
0737 01120  300 012447  S*Y      JSB                     INDIRECT
0738 01121  010 072507                   PASS L     Y          L := Y
0739 01122  003 033007                   ADD  S1    M
0740 01123  010 040647                   PASS M     S1         M := Y + T/A/B
0741 01124  210 002036           WRTE MPCK PASS TAB  CAB        T/A/B := A/B; WRITE
0742 01125  320 043547           JMP                 RETURN
0743                        *
0744 01126  300 012447  L*Y      JSB                     INDIRECT
0745 01127  010 072507                   PASS L     Y          L := Y
0746 01130  003 033007                   ADD  S1    M
0747 01131  230 040647           READ    PASS M     S1         M := Y + T/A/B; READ
0748 01132  010 000047                   PASS CAB   TAB        A/B := T/A/B
0749 01133  227 174700           READ RTN INC  PNM  P          M := P; P := P+1; READ
0750                        *
0751 01134  300 012447  STY      JSB                     INDIRECT
0752 01135  210 072036           WRTE MPCK PASS TAB  Y          T/A/B := Y; WRITE
0753 01136  320 043547           JMP                 RETURN
0754                        *
0755 01137  300 012447  LDY      JSB                     INDIRECT
0756 01140  010 001647                   PASS Y     TAB        Y := T/A/B
0757 01141  227 174700           READ RTN INC  PNM  P          M := P; P := P+1; READ
0758                        *
0759 01142  300 012447  ADY      JSB                     INDIRECT
0760 01143  010 072507                   PASS L     Y          L := Y
0761 01144  263 001647           ENVE    ADD  Y     TAB        Y := Y + T/A/B
0762 01145  227 174700           READ RTN INC  PNM  P          M := P; P := P+1; READ
0763                        *
0764 01146  230 002507  X*Y      READ    PASS L     CAB        L := A/B
0765 01147  010 072047                   PASS CAB   Y          A/B := Y
0766 01150  372 137647           RTN     PASL Y                Y := L
0767                        *
0768 01151  227 173647  ISY      READ    INC  Y     Y          INCREMENT Y; READ
0769 01152  360 041642           RTN  CNDX ALZ  RJS            TEST FOR ZERO
0770 01153  227 174700           READ RTN INC  PNM  P          M := P; P := P+1; READ
0771                        *
0772 01154  220 073647  DSY      READ    DEC  Y     Y          DECREMENT Y; READ
0773 01155  360 041642           RTN  CNDX ALZ  RJS            TEST FOR ZERO
0774 01156  227 174700           READ RTN INC  PNM  P          M := P; P := P+1; READ
PAGE 0027 RTE MICRO-ASSEMBLER REV.A 760818
0776                        *
0777                        *    JUMP INSTRUCTIONS
0778                        *    ----------------
0779                        *
0780 01157  300 012447  JLY      JSB                     INDIRECT
0781 01160  010 075647                   PASS Y     P          Y := P
0782 01161  010 033707                   PASS P     M          P := M
0783 01162  320 047247           JMP                 JPY+2      DO MP CHECK, COMPLETE JLY
0784                        *
0785 01163  010 072507  JPY              PASS L     Y          L := Y
0786 01164  003 001707                   ADD  P     TAB        P := Y + T/A/B
0787 01165  344 120607           IMM     HIGH IRCM 050B        PREPARE MP FOR 0,1 PROTECTION
0788 01166  227 174707           READ    INC  PNM   P          M := P; P := P+1, READ
0789 01167  370 036776           RTN  MPCK                     CHECK JUMP TARGETS
```

```
PAGE 0028 RTE MICRO-ASSEMBLER REV.A 760818
0791                        *
0792                        *        WORD MANIPULATION INSTRUCTIONS
0793                        *        ------------------------------
0794                        *
0795 01170  300 056007  CMW      JSB                     INITIAL
0796 01171  230 006647  LCMW     READ      PASS M    A          M := WORD ADDRESS OF ARRAY 1
0797 01172  010 010647                     PASS M    B          M := WORD ADDRESS OF ARRAY 2
0798 01173  230 000507           READ      PASS L    TAB        L := ARRAY 1 WORD
0799 01174  007 110207                     INC  B    B          BUMP ARRAY 2 ADDRESS
0800 01175  014 101007                     XOR  S1   TAB
0801 01176  327 110342           JMP  CNDX AL15      NOTEQ      TEST FOR SIMILAR SIGN BITS
0802 01177  014 141007                     XOR  S1   S1
0803 01200  004 140747                     SUB       S1
0804 01201  320 050442           JMP  CNDX ALZ  RJS  NOTEQ+2    TEST FOR WORD COMPARE
0805 01202  007 106147                     INC  A    A          BUMP ARRAY 1 ADDRESS
0806 01203  000 045107                     DEC  S3   S3         INCREMENT WORD COUNT
0807 01204  320 003542           JMP  CNDX ALZ       RETURN     TEST FOR COMPLETE COMPARE
0808 01205  335 007442           JMP  CNDX NINT      LCMW       TEST FOR INTERRUPT PENDING
0809 01206  320 056507           JMP                 INTPEND
0810 01207  322 110542  NOTEQ    JMP  CNDX L15       *+4        TEST FOR WORD 1 NEGATIVE
0811 01210  320 050507           JMP                 *+2        AVOID COUT CHECK FOR XOR
0812 01211  321 010542           JMP  CNDX COUT      *+2        TEST FOR WORD 1 LESS THAN WORD 2
0813 01212  007 175707                     INC  P    P          BUMP P
0814 01213  007 175707                     INC  P    P          BUMP P
0815 01214  000 044507                     DEC  L    S3         L := RESIDUAL STRING COUNT
0816 01215  003 010207                     ADD  B    B          UPDATE B PAST STRING
0817 01216  227 174700           READ RTN  INC  PNM  P          M := P; P := P+1; READ
0818                        *
0819 01217  300 056007  MVW      JSB                     INITIAL
0820 01220  230 006647  LMVW     READ      PASS M    A          M  := SOURCE ADDRESS; READ
0821 01221  007 106147                     INC  A    A          BUMP SOURCE ADDRESS COUNTER
0822 01222  010 010647                     PASS M    B          M  := DESTINATION ADDRESS
0823 01223  010 001047                     PASS S2   TAB        S2 := SOURCE WORD
0824 01224  210 042036           WRTE MPCK PASS TAB  S2         STORE SOURCE WORD TO DESTINATION
0825 01225  007 110207                     INC  B    B          BUMP DESTINATION COUNTER
0826 01226  000 045107                     DEC  S3   S3         DECREMENT WORD COUNTER
0827 01227  320 003542           JMP  CNDX ALZ       RETURN     TEST FOR COMPLETE MOVE
0828 01230  335 011002           JMP  CNDX NINT      LMVW       TEST FOR PENDING INTERRUPT
0829 01231  320 056507           JMP                 INTPEND
```

```
PAGE 0029 RTE MICRO-ASSEMBLER REV.A 760818
0831                        *
0832                        *        BYTE MANIPULATION INSTRUCTIONS
0833                        *        ------------------------------
0834                        *
0835 01232  300 056007  MBT      JSB                     INITIAL
0836 01233  150 007064  LMBT     LWF  R1   PASS S2       A          S2 := FROM WORD ADDRESS
0837 01234  300 055507           JSB                     LDBYTE     JUMP TO BYTE LOADING SUBROUTINE
0838 01235  300 054647           JSB                     STBYTE     JUMP TO BYTE STORING SUBROUTINE
0839 01236  007 106147                     INC  A        A          BUMP FROM ADDRESS
0840 01237  000 045107                     DEC  S3       S3         DECREMENT BYTE COUNT
0841 01240  320 003542           JMP  CNDX ALZ           RETURN     TEST FOR COMPLETE MOVE
0842 01241  335 011542           JMP  CNDX NINT          LMBT       TEST FOR INTERRUPT PENDING
0843 01242  320 056507           JMP                     INTPEND
0844                        *
0845 01243  300 056007  CBT      JSB                     INITIAL
0846 01244  150 007064  LCBT     LWF  R1   PASS S2       A          S2 := WORD ADDRESS
0847 01245  300 055507           JSB                     LDBYTE     JUMP TO BYTE LOADING SUBROUTINE
0848 01246  010 041207                     PASS S5       S1         S5 := BYTE 1
0849 01247  150 011064           LWF  R1   PASS S2       B          S2 := WORD ADDRESS
0850 01250  300 055507           JSB                     LDBYTE     JUMP TO BYTE LOADING SUBROUTINE
0851 01251  007 110207                     INC  B        B          BUMP STRING 2 ADDRESS
0852 01252  010 050507                     PASS L        S5         L := BYTE 1
0853 01253  004 140747                     SUB           S1         SUBTRACT: BYTE 2 - BYTE 1
0854 01254  320 050442           JMP  CNDX ALZ   RJS     NOTEQ+2    TEST FOR BYTE COMPARE
0855 01255  007 106147                     INC  A        A          BUMP STRING 1 ADDRESS
0856 01256  000 045107                     DEC  S3       S3         DECREMENT BYTE COUNT
0857 01257  320 003542           JMP  CNDX ALZ           RETURN     TEST FOR COMPLETE COMPARE
0858 01260  335 012202           JMP  CNDX NINT          LCBT       TEST FOR INTERRUPT PENDING
0859 01261  320 056507           JMP                     INTPEND
0860                        *
0861 01262  344 000507  SFB      IMM       HIGH L        000B       L  := 377B
0862 01263  010 033207                     PASS S5       M          SAVE M
0863 01264  012 007107                     AND  S3       A          S3 := TEST BYTE
0864 01265  014 007163                L4   SANL S4       A
0865 01266  010 047163                L4   PASS S4       S4         S4 := TERMINATION BYTE
0866 01267  150 011064  LSFB     LWF  R1   PASS S2       B          S2 := WORD ADDRESS
0867 01270  300 055507           JSB                     LDBYTE     JUMP TO BYTE LOADING SUBROUTINE
0868 01271  010 040507                     PASS L        S1         L  := RIGHT JUSTIFIED BYTE
0869 01272  014 144747                     XOR           S3         COMPARE TO TEST BYTE
0870 01273  320 014602           JMP  CNDX ALZ           SBT+4      TEST FOR TEST BYTE MATCH
0871 01274  007 110207                     INC  B        B          BUMP STRING ADDRESS
0872 01275  014 146747                     XOR           S4         COMPARE TO TERMINATION BYTE
0873 01276  320 003542           JMP  CNDX ALZ           RETURN     TEST FOR TERMINATION BYTE MATCH
0874 01277  335 013342           JMP  CNDX NINT          LSFB       TEST FOR INTERRUPT PENDING
0875 01300  000 075707                     DEC  P        P          DECREMENT P
0876 01301  320 000307           JMP                     HORI       INTERRUPT PENDING
0877                        *
PAGE 0030 RTE MICRO-ASSEMBLER REV.A 760818
0879 01302  150 011064  LBT      LWF  R1   PASS S2       B          S2 := WORD ADDRESS
0880 01303  010 033107                     PASS S3       M          SAVE M
0881 01304  300 055507           JSB                     LDBYTE     JUMP TO BYTE LOADING SUBROUTINE
0882 01305  230 044647           READ      PASS M        S3         RESTORE M AND READ
0883 01306  007 110207                     INC  B        B          BUMP BYTE ADDRESS
0884 01307  370 040147           RTN       PASS A        S1         A  := RIGHT JUSTIFIED BYTE
0885                        *
0886 01310  344 000507  SBT      IMM       HIGH L        000B       L  := 000377
0887 01311  010 033207                     PASS S5       M
0888 01312  012 007007                     AND  S1       A          S1 := RIGHT JUSTIFIED BYTE; READ
0889 01313  300 054647           JSB                     STBYTE     JUMP TO BYTE STORING SUBROUTINE
0890 01314  230 050640           READ RTN  PASS M        S5         RESTORE M AND READ
```

```
PAGE 0031 RTE MICRO-ASSEMBLER REV.A 760818
0892                        *
0893                        *        COMMON SUBROUTINES
0894                        *        ------------------
0895                        *
0896 01315  150 011064  STBYTE   LWF  R1    PASS S2   B         S2 := WORD ADDRESS
0897 01316  230 042647          READ       PASS M    S2        M  := WORD ADDRESS; READ
0898 01317  334 055202          JMP  CNDX  FLAG RJS  *+5       TEST FOR HIGH ORDER BYTE
0899 01320  014 000507                     SANL L    TAB       L  := BYTE TO BE PRESERVED
0900 01321  010 141007                     IOR  S1   S1        S1 := WORD WITH MERGED BYTES
0901 01322  210 040036          WRTE MPCK  PASS TAB  S1        STORE WORD INTO MEMORY
0902 01323  367 110207          RTN        INC  B    B         BUMP B
0903 01324  012 000507                     AND  L    TAB       L  := BYTE TO BE PRESERVED
0904 01325  010 041023               L4    PASS S1   S1
0905 01326  010 041023               L4    PASS S1   S1
0906 01327  010 141007                     IOR  S1   S1        S1 := WORD WITH MERGED BYAES
0907 01330  210 040036          WRTE MPCK  PASS TAB  S1        STORE WORD IN MEMORY
0908 01331  367 110207          RTN        INC  B    B         BUMP B
0909                        *
0910 01332  230 042647  LDBYTE   READ       PASS M    S2        READ
0911 01333  344 000507          IMM        HIGH L    000B      L  := 000377
0912 01334  334 055702          JMP  CNDX  FLAG RJS  *+2       TEST FOR HIGH ORDER BYTE
0913 01335  372 001007          RTN        AND  S1   TAB       S1 := RIGHT JUSTIFIED BYTE
0914 01336  014 001023               L4    SANL S1   TAB
0915 01337  370 041023          RTN  L4    PASS S1   S1        S1 := RIGHT JUSTIFIED BYTE
0916                        *
0917 01340  230 036747  INITIAL  READ                           READ
0918 01341  300 012447          JSB                    INDIRECT
0919 01342  007 174707                     INC  PNM  P         M := P; P := P+1
0920 01343  230 001107          READ       PASS S3   TAB       S3 := INITIAL COUNT; READ
0921 01344  320 017502          JMP  CNDX  ALZ       GOFETCH   TEST FOR ZERO WORD COUNT
0922 01345  010 000507                     PASS L    TAB
0923 01346  360 000002          RTN  CNDX  ALZ                 TEST FOR RESIDUAL COUNT
0924 01347  006 037047                     ZERO S2
0925 01350  210 042036          WRTE MPCK  PASS TAB  S2        CLEAR WORD 3
0926 01351  372 137107          RTN        PASL S3             S3 := ACTUAL COUNT
0927                        *
0928 01352  000 075707  INTPEND            DEC  P    P         DECREMENT P
0929 01353  000 074707                     DEC  PNM  P         DECREMENT P
0930 01354  210 044007          WRTE       PASS TAB  S3        STORE PRESENT WORD COUNT
0931 01355  320 000307          JMP                    HORI      INTERRUPT PENDING
PAGE 0032 RTE MICRO-ASSEMBLER REV.A 760818
0933                        *
0934                        *        BIT MANIPULATION INSTRUCTIONS
0935                        *        -----------------------------
0936                        *
0937 01356  300 012447  BITS     JSB                    INDIRECT
0938 01357  010 000507                     PASS L    TAB       L := MASK
0939 01360  230 074647          READ       PASS M    P         READ WORD TO BE MODIFIED
0940 01361  300 012447          JSB                    INDIRECT
0941 01362  300 057603          JSB  ION               CBS
0942 01363  210 040036          WRTE MPCK  PASS TAB  S1        STORE WORD BACK INTO MEMORY
0943 01364  007 175707                     INC  P    P         P  := P + 1
0944 01365  227 174700          READ RTN   INC  PNM  P
0945                        *
0946 01366  007 175707  FTBS               INC  P    P         P  := P + 1
0947 01367  007 174707                     INC  PNM  P         M  := P; P := P + 1
0948 01370  234 140747          READ       XOR       S1
0949 01371  320 017542          JMP  CNDX  ALZ       *+2
0950 01372  227 174707  GOFETCH  READ      INC  PNM  P         M := P; P := P+1; READ
0951 01373  320 000007          JMP                    FETCH
0952                        *
0953 01374  374 001007  CBS      RTN       SANL S1   TAB       S1 := WORD WITH BITS CLEARED
0954 01375  012 001007  TBS                AND  S1   TAB       S1 := WORD WITH BITS CLEARED
0955 01376  320 057307          JMP                    FTBS      FINISH TBS
0956 01377  370 101007  SBS      RTN       IOR  S1   TAB       S1 := WORD WITH BITS SET
```

```
PAGE 0033 RTE MICRO-ASSEMBLER REV.A 760818
0958                                ORG                 1400B
0959                        ***********************************************************************
0960                        *
0961                        *       21XE MICRO-CODE
0962                        *       MODULE 03: FLOATING POINT INSTRUCTIONS
0963                        *
0964                        *       REV 1976-04-26-1800        EAS
0965                        ***********************************************************************
0966                        *
0967                        *
0968                        ***********************************************************************
0969                        FLOAT    EQU                    *
0970 01400  010 006213      FLT            COV  PASS B      A
0971 01401  006 036147                     ZERO A                 CLEAR LSB'S TO SHIFT INTO B
0972 01402  353 141147                 IMM     CMLO S4     %360   SET EXPONENT FOR MAX INTEGER
0973 01403  000 075707                         DEC  P      P      BECAUSE PACK BUMPS IT
0974 01404  320 073007                 JMP                 PACK
PAGE 0034 RTE MICRO-ASSEMBLER REV.A 760818
0976                        *
0977                        *       ON ENTRY-- A,B = FLOATING POINT NUMBER
0978                        *                  FLAG = 1
0979                        *
0980                        *       ON EXIT    A = INTEGER    B = CHANGED(USUALLY = A,THOUGH)
0981                        *
0982                        *       USES A,B,S1,S2
0983                        *
0984 01405  340 000513      FIX        IMM  COV  LOW  L     %000   L := 1 111 111 100 000 000
0985 01406  153 111024                 LWF  R1   NSOL S1    B      S1 := - EXP - 1
0986 01407  321 120442                 JMP  CNDX AL0        FIXOK1 RETURN ZERO IF EXP < 0
0987 01410  226 036140                 READ RTN  ZERO A
0988 01411  007 141007      FIXOK1                 INC  S1    S1     S1 := -EXP
0989                        *
0990 01412  012 011047                             AND  S2    B      B := LSB'S
0991 01413  010 006207                             PASS B     A      B := MSB'S
0992 01414  010 042147                             PASS A     S2     A := LO BITS
0993 01415  353 140507                 IMM         CMLO L     %360   L := 15
0994 01416  003 041014                     SOV     ADD  S1    S1     CALCULATE 17 - EXP
0995 01417  320 021242                 JMP  CNDX AL2        RTRNINTG NO SHIFTING IF EXP = 17
0996 01420  327 161142                 JMP  CNDX AL15 RJS   FIXOK2 OVERFLOW IF EXP > 17
0997 01421  011 136164                     R1   ONE  A               SET A TO MAX INTEGER
0998 01422  230 036740      RETNFP     READ RTN                      START INSTRUCTION READ; EXIT
0999                        *
1000 01423  010 040567      FIXOK2         RPT  PASS CNTR S1         COUNTER := #SHIFTS; SET REPEAT FF
1001 01424  030 010224                 ARS  R1   PASS B     B      DO THE SHIFTS
1002                        *
1003                        RTRNINTG EQU                   *
1004 01425  010 006513                     COV  PASS L     A      L := LSB'S FROM SHIFT
1005 01426  010 010147                          PASS A     B      A := INTEGER
1006 01427  327 161102                 JMP  CNDX AL15 RJS   RETNFP WE ARE DONE IF A POSITIVE INTEGER
1007 01430  010 142747                          IOR        S2     ELSE CHECK FOR ROUND NECESSARY
1008 01431  320 021102                 JMP  CNDX ALZ        RETNFP RETURN IF NO BITS DANCING
1009                        *
1010 01432  227 106140                 READ RTN  INC  A     A      ELSE ROUND UP AND RETURN
```

```
PAGE 0035 RTE MICRO-ASSEMBLER REV.A 760818
                        *     F A D  /  F S B -- FLOATING POINT ADD / SUBTRACT
                        *     -----     -----    -------- ----- ---   --------
                        *        ON ENTRY--A,B = FIRST OPERAND
                        *                  P = POINTER TO ADDRESS OF SECOND OPERAND
                        *                  FLAG = 1 MEANS ADD    =0 MEANS SUBTRACT
                        *        ON EXIT-- A,B = (FIRST OPERAND) +(-) (SECOND OPERAND)
                        *
                        *        USES REGISTERS S1,S2,S3,S4,S5,S6
                        *
                        FSB      EQU                 *        ITS THE SAME AS FAD
01433  230 036747       FAD      READ
                        *
01434  300 012447                 JSB                 INDIRECT  GO CLEAR INDIRECTS IF NECESSARY
01435  300 071607                 JSB                 UNPACK    GO UNPACK THE NUMBERS
01436  010 042747                          PASS       S2        IS OP2 = 0?
01437  320 062042                 JMP  CNDX ALZ  RJS  *+2       SKIP IF NOT
01440  342 001147                 IMM       LOW  S4   %200      EXP(D) := -200
                        *
01441  010 010747                          PASS       B         IS OP1 = 0?
01442  320 062202                 JMP  CNDX ALZ  RJS  *+2       SKIP IF NOT
01443  342 001207                 IMM       LOW  S5   %200      EXP(C) := -200
                        *
01444  334 022742                 JMP  CNDX FLAG       DIFR      SKIP AHEAD IF DOING AN ADD
01445  017 143047                          CMPS S2   S2        -ELSE NEGATE OP2
01446  017 145107                          CMPS S3   S3
01447  007 145107                          INC  S3   S3
01450  321 062742                 JMP  CNDX COUT RJS  DIFR      IF NO CARRY OUT, GO PROCEED
01451  007 143047                          INC  S2   S2        -BUMP MSB'S
01452  327 162742                 JMP  CNDX AL15 RJS  DIFR      IF POSITIVE, GO PROCEED
01453  001 142747                          DBLS      S2        -WAS IT  100...0?
01454  320 062742                 JMP  CNDX ALZ  RJS  DIFR
                        *
01455  010 043064                      R1   PASS S2   S2        YES, MAKE IT 010...0
01456  007 147147                          INC  S4   S4        AND ADJUST EXPONENT
```

```
PAGE 0036 RTE MICRO-ASSEMBLER REV.A 760818
                          *
                          *     FIND DIFFERENCE IN EXPONENTS--SMALLER EXPONENT GETS RIGHT-SHIFTED.
                          *
01457  010 046507  DIFR                PASS L    S4        L := EXP(D)
01460  004 151007                      SUB  S1   S5        S1 := EXP(C) - EXP(D)
01461  320 024102            JMP  CNDX ALZ       ADD2      IF 0,DO THE ADDITION (NO SHIFTS)
                          *
01462  327 163302            JMP  CNDX AL15 PJS  RVRS
                          *
                          *     WE NEED TO SHIFT THE SECOND NUMBER.
                          *
01463  017 141007                      CMPS S1   S1
01464  007 141007                      INC  S1   S1        S1 := POSITIVE DIFFERENCE
01465  320 063607            JMP                 SWAMPCHK  GO CHECK IF ONE OF THEM>THE OTHER
                          *
                          *     SWAP THE NUMBERS--WRONG ONE IS IN B,A
                          *
01466  010 042207  RVRS                PASS B    S2        B := S2
01467  010 053047                      PASS S2   S6
01470  010 006507                      PASS L    A
01471  010 044147                      PASS A    S3        A := S3
01472  012 137107                      PASL S3
01473  010 051147                      PASS S4   S5        S4 := LARGER EXPONENT
                          *
                          *     CHECK FOR  ABS(EXP2 - EXP1) > 30 --IF SO, ADDING WILL DO NOTHING
                          *
01474  343 116507  SWAMPCHK IMM        LOW  L    %347
01475  003 040547                      ADD  CNTR S1        -- TEST (S1 - 30(B8))
01476  327 172702            JMP  CNDX AL15 RJS  TOOBIG    BUG OUT EARLY
01477  010 036767  ALIGN          RPT
01500  030 010224            ARS  R1   PASS B    B         ALIGN THE OPERAND FOR ADDING
01501  326 163742            JMP  CNDX CNT8 RJS  ALIGN     IF NOT DONE LOOP
01502  150 042522  ADD2      LWF  L1   PASS L    S2        SET UP TO ADD THE HI BITS
01503  243 010207            ENV       ADD  B    B         ADD THE HIGH BITS
01504  010 044507                      PASS L    S3        PREPARE FOR ADDING THE LO BITS
01505  003 006147                      ADD  A    A         ADD THE LO BITS
01506  321 064442            JMP  CNDX COUT RJS  NOCARY    TEST CARRY OUT FROM LO BITS
01507  345 176507            IMM       HIGH L    %177
01510  247 110207            ENV       INC  B    B         BUMP B IF CARRY OUT OF LO BITS
01511  335 173002  NOCARY    JMP  CNDX OVFL RJS  PACK      IF NO OVERFLOW GO PACK IT UP
01512  334 064702            JMP  CNDX FLAG RJS  OFLOW     IF SIGN POSITIVE HANDLE ODD CASE
01513  345 176507            IMM       HIGH L    %177      SET UP L FOR OVF TEST
01514  014 110747                      XOR       B
01515  320 133002            JMP  CNDX ONES      PACK      IF UNIQUE CASE GO PACK IT UP
                          *
01516  150 010224  OFLOW     LWF  R1   PASS B    B         FULL WORD SHIFT;USE FLAG FOR SIGN
01517  150 006164            LWF  R1   PASS A    A
01520  007 147147                      INC  S4   S4        BUMP THE EXPONENT
01521  320 073007            JMP                 PACK      GO PACK IT UP
```

```
PAGE 0037 RTE MICRO-ASSEMBLER REV.A 760818
1097                        *             F M P -- FLOATING POINT MULTIPLY
1098                        *             -----    -------- ----- --------
1099                        *        ON ENTRY--A,B = C
1100                        *                  P = POINTER TO ADDRESS OF D
1101                        *
1102                        *        ON EXIT--A,B = RESULT
1103                        *
1104                        *        USES REGISTERS A,B,S1,S2,S3,S4,S5,S6
1105                        *
1106 01522  230 036747  FMP      READ
1107 01523  300 012447           JSB                 INDIRECT  GO CLEAR INDIRECTS IF NECESSARY
1108 01524  300 071607           JSB                 UNPACK    GO UNPACK THE NUMBERS
1109                        *
1110                        *        FORM EXP(C)+EXP(D)+1 IN S4; SAVE AS THE EXPONENT OF THE RESULT
1111                        *
1112 01525  007 150507                     INC  L    S5
1113 01526  003 047147                     ADD  S4   S4        S4 = EXP(C) + EXP(D) + 1
1114                        *
1115                        *       CALCULATE   MSB(D)*(LSB(C)/2)
1116                        *
1117 01527  010 006164                R1   PASS A    A         A = LSB(C)/2
1118 01530  010 042507                     PASS L    S2        L = MSB(D)
1119 01531  300 077047           JSB                 MPYX      MSB(D)*(LSB(C)/2)
1120 01532  010 007207                     PASS S5   A         S5 = LSB(TEMP)
1121 01533  010 044164                R1   PASS A    S3        A = LSB(D)/2
1122 01534  010 011107                     PASS S3   B         S3 := MSB(TEMP)
1123                        *
1124                        *        CALCULATE MSB(C)*( LSB(D)/2 )
1125                        *
1126 01535  010 052507                     PASS L    S6        L = MSB(C)
1127 01536  300 077047           JSB                 MPYX      MSB(C)*(LSB(D)/2)
1128                        *
1129                        *        ADD RESULTS TO TEMP1
1130                        *
1131 01537  010 006507                     PASS L    A         L = LSB(RESULT)
1132 01540  003 050747                     ADD       S5
1133 01541  321 066142           JMP  CNDX COUT RJS  *+2       TEST FOR CARRY OUT AND SKIP
1134                        *
1135 01542  007 110207                     INC  B    B         ADD IN THE CARRY BIT
1136 01543  010 010507                     PASS L    B         L = MSB(RESULT)
1137 01544  003 045107                     ADD  S3   S3        S3 = MSB(RESULT)
1138                        *
PAGE 0038 RTE MICRO-ASSEMBLER REV.A 760818
1140                        *        CALCULATE MSB(C)*MSB(D)
1141                        *
1142 01545  010 052507                     PASS L    S6        L = MSB(C)
1143 01546  010 042147                     PASS A    S2        A = MSB(D)
1144 01547  300 077047           JSB                 MPYX      MSB(C)*MSB(D)
1145 01550  010 006164  FMPY7         R1   PASS A    A         A := LSB(RESULT)/2
1146 01551  010 044513                COV  PASS L    S3
1147 01552  243 006162           ENV  L1   ADD  A    A         A := (LSB(RESULT)/2+TEMP1)*2
1148 01553  327 173002           JMP  CNDX AL15 RJS  PACK
1149 01554  335 126742           JMP  CNDX OVFL      FMPY8
1150 01555  000 010207                     DEC  B    B         BORROW FROM MSB'S
1151 01556  320 073007           JMP                 PACK      GO PACK IT UP
1152                        *
1153 01557  007 110207  FMPY8              INC  B    B         CARRY TO MSB'S
1154 01560  320 073007           JMP                 PACK      GO PACK IT UP
```

```
PAGE 0039 RTE MICRO-ASSEMBLER REV.A·760818
                          *            F D V -- FLOATING POINT DIVIDE
                          *            -----    -------- ----- ------
                          *
                          *       ON ENTRY-- A,B = C
                          *                  P = POINTER TO ADDRESS OF D
                          *
                          *       ON EXIT--  A,B = RESULT
                          *
                          *       USES REGISTERS A,B,S1,S2,S3,S4,S5,S6
                          *
01561  230 036747  FDV      READ
01562  300 012447           JSB                 INDIRECT  GO CLEAR INDIRECTS IF NECESSARY
01563  300 071607           JSB                 UNPACK    GO UNPACK THE NUMBERS
                          *
                          *    GET SET TO FORM FIRST QUOTIENT OF THE APPROXIMATION (Q0).
                          *
01564  017 143253                COV  CMPS S6   S2        S6 := NOT(MSB(D))
01565  320 135542           JMP  CNDX ONES      OVERFLOW  CHECK FOR DIVIDE BY ZERO!
01566  327 127402           JMP  CNDX AL15      *+2
01567  007 153054                SOV  INC  S2   S6        S2 := ABS(MSB(D)); OVF := SIGN
01570  000 046507                     DEC  L    S4        L := EXP(D) - 1
01571  004 151147                     SUB  S4   S5        S4 := EXP(C)-EXP(D); CNTR := 1'S
01572  030 010224           ARS  R1   PASS B    B         PRESHIFT TO AVOID OVERFLOW
01573  300 075707           JSB                 DIVX
01574  010 007207                     PASS S5   A         S5 := Q0
01575  010 010747                     PASS      B
01576  321 170002           JMP  CNDX AL0  RJS  *+2
01577  000 010207                     DEC  B    B           FIRST LEFT SHIFT FOR NEXT
01600  006 036147                     ZERO A
01601  300 075707           JSB                 DIVX
01602  010 007247                     PASS S6   A
01603  010 044224  FDIV71        R1   PASS B    S3        B := LSB(D)/4
01604  010 010224                R1   PASS B    B
01605  006 036147                     ZERO A
01606  300 075707           JSB                 DIVX
01607  017 106147                     CMPS A    A
01610  007 106147                     INC  A    A
                          *
01611  010 050507                     PASS L    S5        L  := Q0; COUNTER := ALL ONES
01612  300 077047           JSB                 MPYX
                          *

PAGE 0040 RTE MICRO-ASSEMBLER REV.A 760818
01613  010 011047  FDIV81             PASS S2   B         S2 := MSB(-Q0*Q2)
01614  006 036207                     ZERO B              B := 0
01615  010 052747                     PASS      S6        IF Q1
01616  327 171002           JMP  CNDX AL15 RJS  *+2         NEGATIVE,
01617  011 136207                     ONE  B              B := ONES
01620  010 042747                     PASS      S2        IF (-Q0*Q2)
01621  327 171142           JMP  CNDX AL15 RJS  *+2         NEGATIVE,
01622  000 010207                     DEC  B    B             B := B + (ALL ONES)
01623  001 143062                L1   DBLS S2   S2        REORIENT PRODUCT (*4)
01624  010 042507                     PASS L    S2
01625  003 052147                     ADD  A    S6           ADD TO Q1
01626  321 071402           JMP  CNDX COUT RJS  *+2        IF THERE WAS A CARRY OUT,
01627  007 110207                     INC  B    B           ADD IT TO THE HIGH BITS.
                          *
01630  070 010222           LCS  L1   PASS B    B
01631  010 050507                     PASS L    S5
01632  003 010207                     ADD  B    B         ADD Q0 TO MSB
01633  320 073007           JMP                 PACK      GO PACK IT UP
```

```
PAGE 0041 RTE MICRO-ASSEMBLER REV.A 760818
1217                        *
1218                        *     UNPACK THE NUMBERS:     B  := MSB(C)          S2 := MSB(D)
1219                        *                             A  := LSB(C)          S3 := LSB(D)
1220                        *                             S5 := EXP(C)          S4 := EXP(D)
1221                        *
1222 01634  010 001047  UNPACK              PASS S2   TAB        S2 := MSB(D)
1223                        *
1224 01635  007 133107                      INC  S3   M          S3 := ADDRESS OF LSB(D) + EXP(D)
1225 01636  230 044647          READ        PASS M    S3         READ THE WORD
1226 01637  344 000507          IMM         HIGH L    %0         L := 0 000 000 011 111 111
1227 01640  010 007247                      PASS S6   A          S6 := MSB(C)
1228 01641  010 001107                      PASS S3   TAB        S3 := LSB(D) + EXP(D)
1229 01642  012 045153               COV    AND  S4   S3         S4 := EXP(D)
1230 01643  014 045107                      SANL S3   S3         S3 := LSB(D)
1231 01644  014 010147                      SANL A    B          A := LSB(C)
1232 01645  012 011224               R1     AND  S5   B          S5 := UNPACKED EXP(C)
1233 01646  321 172442          JMP  CNDX   AL0  RJS  *+3        TEST EXP SIGN AND SKIP IF +
1234 01647  342 000507          IMM         LOW  L    %200       L := %177600
1235 01650  003 051207                      ADD  S5   S5         S5 := S5 + %177600
1236 01651  010 052207                      PASS B    S6         B := MSB(C)
1237 01652  010 047164               R1     PASS S4   S4         UNPACK EXP(D)
1238 01653  361 141142          RTN  CNDX   AL0  RJS             TEST EXP SIGN AND EXIT IF +
1239 01654  342 000507          IMM         LOW  L    %200       L := %177600
1240 01655  003 047140               RTN    ADD  S4   S4         S4 := S4 + %177600
1241                        *
PAGE 0042 RTE MICRO-ASSEMBLER REV.A 760818
1243                        *    PACK THE NUMBER
1244                        *
1245                        *         IT IS ASSUMED THAT THE MANTISSA IS UNNORMALIZED AND
1246                        *         CONTAINED IN THE ACCUMULATORS AND THE EXPONENT IN S4
1247                        *
1248                        *
1249 01656  010 042207  TOOBIG               PASS B    S2         ENTER HERE IF SWAMP CHECK IN FAD
1250 01657  010 044147                      PASS A    S3         LOAD THE ACC WITH THE LARGER NUM
1251 01660  010 006513  PACK         COV    PASS L    A
1252 01661  010 110747                      IOR       B          A/B = 0?
1253 01662  320 035642          JMP  CNDX   ALZ       RETNFP2    -RETURN IF SO
1254 01663  343 176547          IMM         LOW  CNTR %377       INIT CNTR FOR 1'S COMP COUNTING
1255 01664  001 110507  NORMLIZ             DBLS L    B          L := LEFT SHIFT B BY ONE BIT
1256 01665  014 110747                      XOR       B          SET UP FOR NORMALIZED TEST
1257 01666  327 133502          JMP  CNDX   AL15      ADJEXP     IF NORMALIZED THEN GO AJUST EXP
1258 01667  010 036767               RPT
1259 01670  106 036762          NRM  L1     ZERO                 NORMALIZE A 32 BIT OPERAND
1260 01671  320 073207          JMP                   NORMLIZ    GO LOOP
1261 01672  007 126507  ADJEXP              INC  L    CNTR       L := -(NUMBER OF SHIFTS REQUIRED)
1262 01673  003 047147                      ADD  S4   S4         S4 := CORRECTED EXPONENT
1263 01674  351 176507  ROUND   IMM         CMLO L    %177       L := +200
1264 01675  010 010747                      PASS      B
1265 01676  327 174002          JMP  CNDX   AL15 RJS  *+2        CHECK SIGN OF B--ADJUST ROUND-OFF
1266 01677  003 036507                      ADD  L                 TO  177  (DECREMENT LATCH)
1267 01700  003 006153               COV    ADD  A    A          ADD 200 (OR 177 IF +) TO LSB'S
1268 01701  321 074602          JMP  CNDX   COUT RJS  ADSBXPNT   -ANY CARRY OUT FROM LSB'S?
1269                        *.   NOTE --  BIT 15 OF THE LATCH MUST (!!) BE ZERO AT THIS POINT.
1270 01702  247 110207          ENV         INC  B    B          ADD CARRY TO MSB'S,
1271 01703  335 174342          JMP  CNDX   OVFL RJS  ADSBNOOV     CHECK FOR OVERFLOW
1272 01704  010 010224               R1     PASS B    B          B := 0100...
1273 01705  007 147153               COV    INC  S4   S4         EXP := EXP + 1
1274 01706  320 074607          JMP                   ADSBXPNT
1275 01707  001 110507  ADSBNOOV            DBLS L    B
1276 01710  014 110747                      XOR       B
1277 01711  327 134602          JMP  CNDX   AL15      ADSBXPNT   CHECK FOR B=11...
1278 01712  070 010222          LGS  L1     PASS B    B          RE-NORMALIZE
1279 01713  000 047147                      DEC  S4   S4
```

```
PAGE 0043 RTE MICRO-ASSEMBLER REV.A 760818
1281 01714  342 000514  ADSBXPNT IMM  SOV  LOW  L    %200      GET OVF SET FOR ERROR; L := -200
1282 01715  004 146747                   SUB       S4        TEST (EXP + 200)
1283 01716  327 135402           JMP  CNDX AL15      UNDERFLO  -IF NEGATIVE, UNDERFLOW
1284 01717  003 046747                   ADD       S4        TEST (EXP - 200)
1285 01720  327 175542           JMP  CNDX AL15 RJS  OVERFLOW  -IF POSITIVE, OVERFLOW
1286 01721  150 046762           LWF  L1   PASS      S4        FLAG := EXPONENT SIGN
1287 01722  154 047162           LWF  L1   SANL S4   S4
1288 01723  340 000507           IMM       LOW  L    0         L := %177400
1289 01724  012 006507                   AND  L    A         L := LSB'S
1290 01725  227 174707           READ    INC  PNM  P         START NEXT INSTRUCTION FETCH
1291 01726  010 010153                COV  PASS A    B         A := MSB'S
1292 01727  010 146200                RTN  IOR  B    S4        B := LSB'S OR EXPONENT
1293                        *
1294 01730  006 036207  UNDERFLO         ZERO B               UNDERFLOW; A,B:=0; OVF := 1
1295 01731  006 036147  RZERO            ZERO A
1296 01732  227 174700           READ RTN  INC  PNM  P         START READ AND EXIT
1297                        *
1298 01733  343 174214  OVERFLOW IMM  SOV  LOW  B    %376      OVERFLOW; A,B := MOST + NUMBER
1299 01734  011 136164  OVER32K       R1   ONE  A
1300 01735  227 174700  RETNFP2  READ RTN  INC  PNM  P         START READ; EXIT

PAGE 0044 RTE MICRO-ASSEMBLER REV.A 760818
1302                        *     MULTIPLY AND DIVIDE UTILITIES FOR FLOATING POINT USE ONLY
1303                        *
1304                        *
1305                        DIVX     EQU                 *
1306 01736  150 010762           LWF  L1   PASS      B         B < 0?  FLAG := SIGN
1307 01737  327 176242           JMP  CNDX AL15 RJS  READY
1308 01740  017 110207                   CMPS B    B         DOUBLE-WORD NEGATE
1309 01741  017 106147                   CMPS A    A
1310 01742  007 106147                   INC  A    A
1311 01743  321 076242           JMP  CNDX COUT RJS  READY
1312 01744  007 110207                   INC  B    B         ADD IN THE CARRY
1313 01745  010 042527  READY         RPT  PASS L    S2        GET THE DIVISOR
1314 01746  124 110222           DIV  L1   SUB  B    B         DO THE DIVIDE STEP 16 TIMES.
1315                        *
1316 01747  010 010224                R1   PASS B    B         -FORM POSITIVE REMAINDER
1317 01750  334 076542           JMP  CNDX FLAG RJS  *+3
1318 01751  017 110207                   CMPS B    B
1319 01752  007 110207                   INC  B    B                   ADD ONE
1320 01753  335 136742           JMP  CNDX OVFL      DIVXFTST
1321 01754  374 076742           RTN  CNDX FLAG RJS
1322 01755  017 106147  COMPLEMT         CMPS A    A
1323 01756  007 106140                RTN  INC  A    A
1324 01757  334 076642  DIVXFTST JMP  CNDX FLAG RJS  COMPLEMT
1325 01760  370 036747           RTN
1326                        *
1327 01761  010 007007  MPYX             PASS S1   A
1328 01762  006 036227                RPT  ZERO B
1329 01763  163 010224           MPY  R1   ADD  B    B
1330 01764  010 040747                   PASS      S1
1331 01765  307 137402           JSB  CNDX AL15      SUBB
1332 01766  362 177402           RTN  CNDX L15  RJS
1333 01767  010 040507                   PASS L    S1
1334 01770  364 110207  SUBB     RTN     SUB  B    B
1335                        *
1336                             ORG                 %1777
1337 01777  320 073007           JMP                 PACK      EXTERNAL ENTRY FOR PACK
```

```
PAGE 0045 RTE MICRO-ASSEMBLER REV.A 760818
1339                                ORG                 2000B
1340                        *
1341                        *       ROM JUMP TABLE
1342                        *       --------------
1343                        *
1344 02000  000 000073              DEF                 SRG          0
1345 02001  000 000073              DEF                 SRG          1
1346 02002  000 000073              DEF                 SRG          2
1347 02003  000 000073              DEF                 SRG          3
1348 02004  000 000053              DEF                 ASGNO*       4
1349 02005  000 000063              DEF                 ASGCL*       5
1350 02006  000 000067              DEF                 ASGCM*       6
1351 02007  000 000057              DEF                 ASGCC*       7
1352 02010  000 000073              DEF                 SRG         10
1353 02011  000 000073              DEF                 SRG         11
1354 02012  000 000073              DEF                 SRG         12
1355 02013  000 000073              DEF                 SRG         13
1356 02014  000 000053              DEF                 ASGNO*      14
1357 02015  000 000063              DEF                 ASGCL*      15
1358 02016  000 000067              DEF                 ASGCM*      16
1359 02017  000 000057              DEF                 ASGCC*      17
1360 02020  000 000015              DEF                 AND         20
1361 02021  000 000015              DEF                 AND         21
1362 02022  000 000015              DEF                 AND         22
1363 02023  000 000015              DEF                 AND         23
1364 02024  000 000015              DEF                 AND         24
1365 02025  000 000015              DEF                 AND         25
1366 02026  000 000015              DEF                 AND         26
1367 02027  000 000015              DEF                 AND         27
1368 02030  000 000043              DEF                 JSB         30
1369 02031  000 000043              DEF                 JSB         31
1370 02032  000 000043              DEF                 JSB         32
1371 02033  000 000043              DEF                 JSB         33
1372 02034  000 000043              DEF                 JSB         34
1373 02035  000 000043              DEF                 JSB         35
1374 02036  000 000043              DEF                 JSB         36
1375 02037  000 000043              DEF                 JSB         37
PAGE 0046 RTE MICRO-ASSEMBLER REV.A 760818
1377 02040  000 000051              DEF                 XOR         40
1378 02041  000 000051              DEF                 XOR         41
1379 02042  000 000051              DEF                 XOR         42
1380 02043  000 000051              DEF                 XOR         43
1381 02044  000 000051              DEF                 XOR         44
1382 02045  000 000051              DEF                 XOR         45
1383 02046  000 000051              DEF                 XOR         46
1384 02047  000 000051              DEF                 XOR         47
1385 02050  000 000040              DEF                 JMP         50
1386 02051  000 000040              DEF                 JMP         51
1387 02052  000 000040              DEF                 JMP         52
1388 02053  000 000040              DEF                 JMP         53
1389 02054  000 000040              DEF                 JMP         54
1390 02055  000 000040              DEF                 JMP         55
1391 02056  000 000040              DEF                 JMP         56
1392 02057  000 000040              DEF                 JMP         57
1393 02060  000 000026              DEF                 IOR         60
1394 02061  000 000026              DEF                 IOR         61
1395 02062  000 000026              DEF                 IOR         62
1396 02063  000 000026              DEF                 IOR         63
1397 02064  000 000026              DEF                 IOR         64
1398 02065  000 000026              DEF                 IOR         65
1399 02066  000 000026              DEF                 IOR         66
1400 02067  000 000026              DEF                 IOR         67
1401 02070  000 000030              DEF                 ISZ         70
1402 02071  000 000030              DEF                 ISZ         71
1403 02072  000 000030              DEF                 ISZ         72
1404 02073  000 000030              DEF                 ISZ         73
1405 02074  000 000030              DEF                 ISZ         74
1406 02075  000 000030              DEF                 ISZ         75
1407 02076  000 000030              DEF                 ISZ         76
1408 02077  000 000030              DEF                 ISZ         77
1409 02100  000 000017              DEF                 AD*        100
1410 02101  000 000017              DEF                 AD*        101
1411 02102  000 000017              DEF                 AD*        102
1412 02103  000 000017              DEF                 AD*        103
1413 02104  000 000017              DEF                 AD*        104
1414 02105  000 000017              DEF                 AD*        105
1415 02106  000 000017              DEF                 AD*        106
1416 02107  000 000017              DEF                 AD*        107
```

```
PAGE 0047 RTE MICRO-ASSEMBLER REV.A 760818
1418 02110  000 000017          DEF                 AD*        110
1419 02111  000 000017          DEF                 AD*        111
1420 02112  000 000017          DEF                 AD*        112
1421 02113  000 000017          DEF                 AD*        113
1422 02114  000 000017          DEF                 AD*        114
1423 02115  000 000017          DEF                 AD*        115
1424 02116  000 000017          DEF                 AD*        116
1425 02117  000 000017          DEF                 AD*        117
1426 02120  000 000021          DEF                 CP*        120
1427 02121  000 000021          DEF                 CP*        121
1428 02122  000 000021          DEF                 CP*        122
1429 02123  000 000021          DEF                 CP*        123
1430 02124  000 000021          DEF                 CP*        124
1431 02125  000 000021          DEF                 CP*        125
1432 02126  000 000021          DEF                 CP*        126
1433 02127  000 000021          DEF                 CP*        127
1434 02130  000 000021          DEF                 CP*        130
1435 02131  000 000021          DEF                 CP*        131
1436 02132  000 000021          DEF                 CP*        132
1437 02133  000 000021          DEF                 CP*        133
1438 02134  000 000021          DEF                 CP*        134
1439 02135  000 000021          DEF                 CP*        135
1440 02136  000 000021          DEF                 CP*        136
1441 02137  000 000021          DEF                 CP*        137
1442 02140  000 000047          DEF                 LD*        140
1443 02141  000 000047          DEF                 LD*        141
1444 02142  000 000047          DEF                 LD*        142
1445 02143  000 000047          DEF                 LD*        143
1446 02144  000 000047          DEF                 LD*        144
1447 02145  000 000047          DEF                 LD*        145
1448 02146  000 000047          DEF                 LD*        146
1449 02147  000 000047          DEF                 LD*        147
1450 02150  000 000047          DEF                 LD*        150
1451 02151  000 000047          DEF                 LD*        151
1452 02152  000 000047          DEF                 LD*        152
1453 02153  000 000047          DEF                 LD*        153
1454 02154  000 000047          DEF                 LD*        154
1455 02155  000 000047          DEF                 LD*        155
1456 02156  000 000047          DEF                 LD*        156
1457 02157  000 000047          DEF                 LD*        157

PAGE 0048 RTE MICRO-ASSEMBLER REV.A 760818
1459 02160  000 000135          DEF                 ST*        160
1460 02161  000 000135          DEF                 ST*        161
1461 02162  000 000135          DEF                 ST*        162
1462 02163  000 000135          DEF                 ST*        163
1463 02164  000 000135          DEF                 ST*        164
1464 02165  000 000135          DEF                 ST*        165
1465 02166  000 000135          DEF                 ST*        166
1466 02167  000 000135          DEF                 ST*        167
1467 02170  000 000135          DEF                 ST*        170
1468 02171  000 000135          DEF                 ST*        171
1469 02172  000 000135          DEF                 ST*        172
1470 02173  000 000135          DEF                 ST*        173
1471 02174  000 000135          DEF                 ST*        174
1472 02175  000 000135          DEF                 ST*        175
1473 02176  000 000135          DEF                 ST*        176
1474 02177  000 000135          DEF                 ST*        177
1475 02200  000 000113          DEF                 JTBL1000   200 ASL,LSL,FRL,MPY
1476 02201  000 000153          DEF                 DIV        201
1477 02202  000 000227          DEF                 JTBL1010   202 ASR,LSR,FRR
1478 02203  000 000107          DEF                 MAC1       203
1479 02204  000 000077          DEF                 IOG        204 HLT,STF,SFC,SF5
1480 02205  000 000077          DEF                 IOG        205 MIA,LIA,OTA,STC
1481 02206  000 000077          DEF                 IOG        206 HLT,CLF
1482 02207  000 000077          DEF                 IOG        207 MIA,LIA,OTA,STC
1483 02210  000 000120          DEF                 DLD        210
1484 02211  000 000130          DEF                 DST        211
1485 02212  000 000103          DEF                 MAC0       212
1486 02213  000 000107          DEF                 MAC1       213
1487 02214  000 000077          DEF                 IOG        214 HLT,STF,SFC,SFS
1488 02215  000 000077          DEF                 IOG        215 MIB,LIB,OTB,CLC
1489 02216  000 000077          DEF                 IOG        216 HLT,CLF
1490 02217  000 000077          DEF                 IOG        217 MIB,LIB,OTB,CLC
1491 02220  000 000002          DEF                 MRGIND     220
1492 02221  000 000002          DEF                 MRGIND     221
1493 02222  000 000002          DEF                 MRGIND     222
1494 02223  000 000002          DEF                 MRGIND     223
1495 02224  000 000002          DEF                 MRGIND     224
1496 02225  000 000002          DEF                 MRGIND     225
1497 02226  000 000002          DEF                 MRGIND     226
1498 02227  000 000002          DEF                 MRGIND     227
```

```
PAGE 0049 RTE MICRO-ASSEMBLER REV.A 760818
1500 02230  000 000041          DEF                 JSB,I     230
1501 02231  000 000041          DEF                 JSB,I     231
1502 02232  000 000041          DEF                 JSB,I     232
1503 02233  000 000041          DEF                 JSB,I     233
1504 02234  000 000041          DEF                 JSB,I     234
1505 02235  000 000041          DEF                 JSB,I     235
1506 02236  000 000041          DEF                 JSB,I     236
1507 02237  000 000041          DEF                 JSB,I     237
1508 02240  000 000002          DEF                 MRGIND    240
1509 02241  000 000002          DEF                 MRGIND    241
1510 02242  000 000002          DEF                 MRGIND    242
1511 02243  000 000002          DEF                 MRGIND    243
1512 02244  000 000002          DEF                 MRGIND    244
1513 02245  000 000002          DEF                 MRGIND    245
1514 02246  000 000002          DEF                 MRGIND    246
1515 02247  000 000002          DEF                 MRGIND    247
1516 02250  000 000036          DEF                 JMP,I     250
1517 02251  000 000036          DEF                 JMP,I     251
1518 02252  000 000036          DEF                 JMP,I     252
1519 02253  000 000036          DEF                 JMP,I     253
1520 02254  000 000036          DEF                 JMP,I     254
1521 02255  000 000036          DEF                 JMP,I     255
1522 02256  000 000036          DEF                 JMP,I     256
1523 02257  000 000036          DEF                 JMP,I     257
1524 02260  000 000002          DEF                 MRGIND
1525 02261  000 000002          DEF                 MRGIND
1526 02262  000 000002          DEF                 MRGIND
1527 02263  000 000002          DEF                 MRGIND
1528 02264  000 000002          DEF                 MRGIND
1529 02265  000 000002          DEF                 MRGIND
1530 02266  000 000002          DEF                 MRGIND
1531 02267  000 000002          DEF                 MRGIND
1532 02270  000 000002          DEF                 MRGIND
1533 02271  000 000002          DEF                 MRGIND
1534 02272  000 000002          DEF                 MRGIND
1535 02273  000 000002          DEF                 MRGIND
1536 02274  000 000002          DEF                 MRGIND
1537 02275  000 000002          DEF                 MRGIND
1538 02276  000 000002          DEF                 MRGIND
1539 02277  000 000002          DEF                 MRGIND

PAGE 0050 RTE MICRO-ASSEMBLER REV.A 760818
1541 02300  000 000002          DEF                 MRGIND
1542 02301  000 000002          DEF                 MRGIND
1543 02302  000 000002          DEF                 MRGIND
1544 02303  000 000002          DEF                 MRGIND
1545 02304  000 000002          DEF                 MRGIND
1546 02305  000 000002          DEF                 MRGIND
1547 02306  000 000002          DEF                 MRGIND
1548 02307  000 000002          DEF                 MRGIND
1549 02310  000 000002          DEF                 MRGIND
1550 02311  000 000002          DEF                 MRGIND
1551 02312  000 000002          DEF                 MRGIND
1552 02313  000 000002          DEF                 MRGIND
1553 02314  000 000002          DEF                 MRGIND
1554 02315  000 000002          DEF                 MRGIND
1555 02316  000 000002          DEF                 MRGIND
1556 02317  000 000002          DEF                 MRGIND
1557 02320  000 000002          DEF                 MRGIND
1558 02321  000 000002          DEF                 MRGIND
1559 02322  000 000002          DEF                 MRGIND
1560 02323  000 000002          DEF                 MRGIND
1561 02324  000 000002          DEF                 MRGIND
1562 02325  000 000002          DEF                 MRGIND
1563 02326  000 000002          DEF                 MRGIND
1564 02327  000 000002          DEF                 MRGIND
1565 02330  000 000002          DEF                 MRGIND
1566 02331  000 000002          DEF                 MRGIND
1567 02332  000 000002          DEF                 MRGIND
1568 02333  000 000002          DEF                 MRGIND
1569 02334  000 000002          DEF                 MRGIND
1570 02335  000 000002          DEF                 MRGIND
1571 02336  000 000002          DEF                 MRGIND
1572 02337  000 000002          DEF                 MRGIND
1573 02340  000 000002          DEF                 MRGIND
1574 02341  000 000002          DEF                 MRGIND
1575 02342  000 000002          DEF                 MRGIND
1576 02343  000 000002          DEF                 MRGIND
1577 02344  000 000002          DEF                 MRGIND
1578 02345  000 000002          DEF                 MRGIND
1579 02346  000 000002          DEF                 MRGIND
1580 02347  000 000002          DEF                 MRGIND
```

```
PAGE 0051 RTE MICRO-ASSEMBLER REV.A 760818
1582 02350  000 000002          DEF                 MRGIND
1583 02351  000 000002          DEF                 MRGIND
1584 02352  000 000002          DEF                 MRGIND
1585 02353  000 000002          DEF                 MRGIND
1586 02354  000 000002          DEF                 MRGIND
1587 02355  000 000002          DEF                 MRGIND
1588 02356  000 000002          DEF                 MRGIND
1589 02357  000 000002          DEF                 MRGIND
1590 02360  000 000002          DEF                 MRGIND
1591 02361  000 000002          DEF                 MRGIND
1592 02362  000 000002          DEF                 MRGIND
1593 02363  000 000002          DEF                 MRGIND
1594 02364  000 000002          DEF                 MRGIND
1595 02365  000 000002          DEF                 MRGIND
1596 02366  000 000002          DEF                 MRGIND
1597 02367  000 000002          DEF                 MRGIND
1598 02370  000 000002          DEF                 MRGIND
1599 02371  000 000002          DEF                 MRGIND
1600 02372  000 000002          DEF                 MRGIND
1601 02373  000 000002          DEF                 MRGIND
1602 02374  000 000002          DEF                 MRGIND
1603 02375  000 000002          DEF                 MRGIND
1604 02376  000 000002          DEF                 MRGIND
1605 02377  000 000002          DEF                 MRGIND
1606                            END
END OF PASS 2: NO ERRORS

PAGE 0052 RTE MICRO CROSS-REFERENCE REV.1813 771212
SYMBOLS=0196  REFERENCES=0505  SOURCE LINES=1606
AD*        0032  1409  1410  1411  1412  1413  1414  1415  1416  1418
           1419  1420  1421  1422  1423  1424  1425
ADD2       1079  1052
ADJEXP     1261  1257
ADSBNOOV   1275  1271
ADSBXPNT   1281  1268  1274  1277
ADX        0719  0665
ADY        0759  0673
ALIGN      1076  1078
AND        0029  1360  1361  1362  1363  1364  1365  1366  1367
ASGCC*     0077  1351  1359
ASGCL*     0082  1349  1357
ASGCM*     0087  1350  1358
ASGNO*     0072  1348  1356
ASL        0190  0228
ASR        0195  0252  0539
BITS       0937  0686  0687  0688
BKGNDCK    0603  0606
CBS        0953  0941
CBT        0845  0681
CMW        0795  0689
COMPLEMT   1322  1324
CONTROL    0121  **NOT REFERENCED**
CP*        0035  1426  1427  1428  1429  1430  1431  1432  1433  1434
           1435  1436  1437  1438  1439  1440  1441
CPTEST     0512  0494  0543
DECDMS     0340  0335
DECM       0335  0290
```

```
PAGE 0053 RTE MICRO CROSS-REFERENCE REV.1813 771212
DIAG       0219  0227
DIFR       1050  1034  1038  1040  1042
DIV        0160  1476
DIVS       0169  0164
DIVX       1305  1179  1185  1190
DIVXFTST   1324  1320
DLD        0130  1483
DMSLOAD    0549  0578
DSPICODE   0416  0276  0343
DST        0139  1484
DSX        0732  0676
DSY        0772  0684
EIG        0659  0635  0636
EM1000     0227  0119
EM1010     0251  0211
FAD        1022  0638
FAILURE    0608  0507  0530  0532  0534  0536  0538  0540  0594
FDIV71     1187  **NOT REFERENCED**
FDIV81     1198  **NOT REFERENCED**
FDV        1166  0641
FETCH      0009  0023  0303  0305  0951
FIX        0984  0642
FIXOK1     0988  0986
FIXOK2     1000  0996
FLOAT      0969  **NOT REFERENCED**
FLT        0970  0643
FMP        1106  0640
FMPY7      1145  **NOT REFERENCED**

PAGE 0054 RTE MICRO CROSS-REFERENCE REV.1813 771212
FMPY8      1153  1149
FPDIAG     0232  **NOT REFERENCED**
FSB        1021  0639
FTBS       0946  0955
GOFETCH    0950  0921
HALT       0261  0018  0222
HORI       0017  0124  0245  0251  0876  0931
IDLE       0281  **NOT REFERENCED**
INCDMS     0341  0337
INCM       0337  0289
INDIRECT   0240  0052  0056  0131  0140  0149  0161  0214  0243  0244
           0696  0703  0711  0715  0719  0737  0744  0751  0755  0759
           0780  0918  0937  0940  1024  1107  1167
INITIAL    0917  0795  0819  0835  0845
INSTP      0298  0294
INTPEND    0928  0809  0829  0843  0859
IOG        0104  1479  1480  1481  1482  1487  1488  1489  1490
IOR        0041  1393  1394  1395  1396  1397  1398  1399  1400
ISX        0728  0675
ISY        0768  0683
ISZ        0044  1401  1402  1403  1404  1405  1406  1407  1408
JLY        0780  0677
JMP        0053  1385  1386  1387  1388  1389  1390  1391  1392
JMP,I      0051  1516  1517  1518  1519  1520  1521  1522  1523
JPY        0785  0685  0783
JSB        0057  1368  1369  1370  1371  1372  1373  1374  1375
JSB,I      0055  1500  1501  1502  1503  1504  1505  1506  1507
JSBSCAN    0283  **NOT REFERENCED**
JTBL1000   0119  1475
```

```
PAGE 0055 RTE MICRO CROSS-REFERENCE REV.1813 771212
JTBL1010  0211  1477
L*X       0703  0661
L*Y       0744  0669
LBT       0879  0678
LCBT      0846  0858
LCMW      0796  0808
LD*       0062  1442  1443  1444  1445  1446  1447  1448  1449  1450
          1451  1452  1453  1454  1455  1456  1457
LDBYTE    0910  0837  0847  0850  0867  0881
LDX       0715  0664
LDY       0755  0672
LEFT      0313  0287
LI*       0111  **NOT REFERENCED**
LMBT      0836  0842
LMVW      0820  0828
LOADER    0438  0291  0431  0557
LOOP      0460  0485
LSFB      0866  0874
LSL       0199  0229
LSR       0202  0253
MAC0      0109  1485
MAC1      0114  1478  1486
MACTABL0  0638  0109
MACTABL1  0621  0114
MBT       0835  0680
MEMLOST   0542  0220  0264
MEMSIZE   0440  0448
MI*       0106  0104
MODE      0307  0293  0298

PAGE 0056 RTE MICRO CROSS-REFERENCE REV.1813 771212
MPY       0148  0235
MPYX      1327  1119  1127  1144  1195
MRGIND    0012  0014  0015  1491  1492  1493  1494  1495  1496  1497
          1498  1508  1509  1510  1511  1512  1513  1514  1515  1524
          1525  1526  1527  1528  1529  1530  1531  1532  1533  1534
          1535  1536  1537  1538  1539  1541  1542  1543  1544  1545
          1546  1547  1548  1549  1550  1551  1552  1553  1554  1555
          1556  1557  1558  1559  1560  1561  1562  1563  1564  1565
          1566  1567  1568  1569  1570  1571  1572  1573  1574  1575
          1576  1577  1578  1579  1580  1582  1583  1584  1585  1586
          1587  1588  1589  1590  1591  1592  1593  1594  1595  1596
          1597  1598  1599  1600  1601  1602  1603  1604  1605
MVW       0819  0690
NOCARY    1086  1083
NORMLIZ   1255  1260
NOTEQ     0810  0801  0804  0854
OFLOW     1092  1087
OT*       0116  **NOT REFERENCED**
OVER32K   1299  **NOT REFERENCED**
OVERFLOW  1298  1173  1285
PACK      1251  0974  1086  1090  1095  1148  1151  1154  1215  1337
READY     1313  1307  1311
REGTEST   0518  0515
RET       0097  0122  0173
RETNFP    0998  0232  1006  1008
RETNFP2   1300  1253
RETURN    0708  0701  0713  0742  0753  0807  0827  0841  0857  0873
RIGHT     0323  0288
RIPLOOP   0589  0598  0600
RIPP1MW   0545  **NOT REFERENCED**
RIPP32K   0583  0563  0566  0569  0571
RMDR      0186  0168  0179  0184
ROUND     1263  **NOT REFERENCED**
```

```
PAGE 0057 RTE MICRO CROSS-REFERENCE REV.1813 771212
RPL        0430  0268  0269
RRL        0205  0231
RRR        0208  0255
RTRNINTG   1003  0995
RUN        0297  0266  0432
RVRS       1064  1054
RZERO      1295  **NOT REFERENCED**
S*X        0696  0659
S*Y        0737  0667
SBS        0956  **NOT REFERENCED**
SBT        0886  0679  0870
SCAN       0286  0283
SELCODE    0451  **NOT REFERENCED**
SFB        0861  0682
SRG        0095  1344  1345  1346  1347  1352  1353  1354  1355
ST*        0144  1459  1460  1461  1462  1463  1464  1465  1466  1467
           1468  1469  1470  1471  1472  1473  1474
STBYTE     0896  0838  0889
STCPUS     0366  0356
STFENCE    0362  0357
STORE      0343  0292
STOREA     0355  **NOT REFERENCED**
STOREB     0354  **NOT REFERENCED**
STOREF     0357  **NOT REFERENCED**
STOREM     0353  **NOT REFERENCED**
STOREMM    0358  **NOT REFERENCED**
STOREMN    0359  **NOT REFERENCED**
STOREP     0351  **NOT REFERENCED**
STORES     0350  0344
PAGE 0058 RTE MICRO CROSS-REFERENCE REV.1813 771212
STOREST    0356  **NOT REFERENCED**
STORET     0378  0352
STOREX     0361  **NOT REFERENCED**
STOREY     0360  **NOT REFERENCED**
STWORD     0483  0473  0477  0480
STX        0711  0662
STY        0751  0670
SUBB       1334  1331
SWAMPCHK   1073  1060
TBS        0954  **NOT REFERENCED**
TEST32K    0500  0510
TESTDMS    0572  0560
TIMER      0214  0219  0230
TOOBIG     1249  1075
UNDERFLO   1294  1283
UNPACK     1222  1025  1108  1168
UPDATEA    0390  **NOT REFERENCED**
UPDATEB    0389  **NOT REFERENCED**
UPDATEF    0392  **NOT REFERENCED**
UPDATEM    0388  **NOT REFERENCED**
UPDATEMM   0393  **NOT REFERENCED**
UPDATEMN   0394  **NOT REFERENCED**
UPDATEP    0386  **NOT REFERENCED**
UPDATES    0385  0277
UPDATEST   0391  **NOT REFERENCED**
UPDATET    0387  **NOT REFERENCED**
UPDATEX    0396  **NOT REFERENCED**
UPDATEY    0395  **NOT REFERENCED**

PAGE 0059 RTE MICRO CROSS-REFERENCE REV.1813 771212
UPDCPUS    0399  0391
UPDFENCE   0397  0392
USER       0382  0270  0613
WAIT       0276  0273  0284  0295  0345  0380  0612
X*X        0724  0666
X*Y        0764  0674
XOR        0065  1377  1378  1379  1380  1381  1382  1383  1384
```

```
PAGE 0024 RTE MICRO-ASSEMBLER REV.A 760818
0703                                ORG                 760B
0704                        *
0705                        *       PRIMARY MAPPING TABLE
0706                        *       --------------------
0707                        *
0708 00760  320 100004  MACTABL1 JMP  RJ30            2000B     105400B, HP RESERVED
0709 00761  320 140004           JMP  RJ30            3000B     105420B, HP RESERVED
0710 00762  325 140004           JMP  RJ30            27000B    105440B, USER RESERVED
0711 00763  321 100004           JMP  RJ30            6000B     105460B, HP RESERVED
0712 00764  325 160004           JMP  RJ30            27400B    105500B, USER RESERVED
0713 00765  326 000004           JMP  RJ30            30000B    105520B, USER RESERVED
0714 00766  326 020004           JMP  RJ30            30400B    105540B, USER RESERVED
0715 00767  326 040004           JMP  RJ30            31000B    105560B, USER RESERVED
0716 00770  327 000004           JMP  RJ30            34000B    105600B, USER RESERVED
0717 00771  327 020004           JMP  RJ30            34400B    105620B, USER RESERVED
0718 00772  327 040004           JMP  RJ30            35000B    105640B, USER RESERVED
0719 00773  327 060004           JMP  RJ30            35400B    105660B, USER RESERVED
0720 00774  324 000004           JMP  RJ30            20000B    DYNAMIC MAPPING SYSTEM
0721 00775  324 001004           JMP  RJ30            20020B    DYNAMIC MAPPING SYSTEM
0722 00776  320 041004           JMP  RJ30            EIG       EXTENDED INSTRUCTION GROUP
0723 00777  320 042004           JMP  RJ30            EIG+20B   EXTENDED INSTRUCTION GROUP
0724                        *
0725                        *
0726                        *
0727                        *
0728                        *
0729                        *

PAGE 0025 RTE MICRO-ASSEMBLER REV.A 760818
0731                        *
0732                        *
0733                        *
0734                        *
0735                        *
0736                        *
0737                        *
0738                        *
0739                        *
0740                        *
0741                        *
0742                        *       21MX F-SERIES BASE SET MICROCODE
0743                        *       MODULES 2,3
0744                        *       -------------------------------
0745                        *
0746                        *
0747                        *
0748                        *

PAGE 0026 RTE MICRO-ASSEMBLER REV.A 760818
0750                        *
0751                                ORG                 1000B     BEGINNING OF MODULE 2
0752                        *
0753 01000  320 060004  MACTABL0 JMP  RJ30            ASMD2345  FLOATING POINT ADD
0754 01001  320 060004           JMP  RJ30            ASMD2345  FLOATING POINT SUBTRACT
0755 01002  320 060004           JMP  RJ30            ASMD2345  FLOATING POINT MULTIPLY
0756 01003  320 060004           JMP  RJ30            ASMD2345  FLOATING POINT DIVIDE
0757 01004  343 130507           IMM       LOW  L     354B      L= COMPL OF 24B
0758 01005  320 061004           JMP  RJ30            XTSD2345  FLOATING POINT TO INTEGER
0759                        *
0760 01006  327 100004           JMP  RJ30            36000B    105140B, USER RESERVED
0761 01007  327 140004           JMP  RJ30            37000B    105160B, USER RESERVED
0762 01010  324 040004           JMP  RJ30            21000B    FAST FORTRAN
0763 01011  324 060004           JMP  RJ30            21400B    FAST FORTRAN
0764 01012  324 100004           JMP  RJ30            22000B    105240B, HP RESERVED
0765 01013  321 000004           JMP  RJ30            4000B     105260B, HP RESERVED
0766 01014  324 140004           JMP  RJ30            23000B    105300B, HP RESERVED
0767 01015  325 000004           JMP  RJ30            24000B    105320B, SCI. INSTRUCTION SET
0768 01016  322 000004           JMP  RJ30            10000B    105340B, HP RESERVED
0769 01017  325 040004           JMP  RJ30            25000B    105360B, SCI. INSTRUCTION SET
```

```
PAGE 0004 RTE MICRO-ASSEMBLER REV.A 760818
0004                          ORG                 %20000
0005                    *
0006                    *
0007                    *
0008                    *
0009                    *
0010                    *
0011                    ***********************************************************************
0012                    *                                                                     *
0013                    *       MEMORY EXPANSION UNIT MACRO INSTRUCTIONS                      *
0014                    *       ------ --------- ---- ----- -----------                       *
0015                    *       1977-12-20-1430
0016                    ******************************************************
0017                    *
0018                    INDIRECT EQU                %251
0019                    HORI     EQU                %006
0020                    *
0021                    ******************************************************
0022                    *
0023                    *   REGISTER ASSIGNMENTS
0024                    *
0025                    *     S3 :: P-REGISTER
0026                    *     S4 :: MEM CONTROL WORD; MEM ADDRESS REGISTER
0027                    *     S5 :: WORDS AND MAP DATA IN LOOP EXECUTION; MASKS AND CONSTANTS
0028                    *     S6 :: GENERAL PURPOSE SCRATCH
0029                    *
0030                    ***********************************************************************
PAGE 0005 RTE MICRO-ASSEMBLER REV.A 760818
0032                    *
0033                    *
0034                    *
0035                    *
0036                    *
0037                    ***********************************************************************
0038                    *      ENTRY JUMP TABLE
0039                    ***********************************************************************
0040                    *
0041                    *       MACRO JUMP POINT AND MNEMONIC    BINARY CODE
0042                    *
0043 20000  324 002047  JTABL    JMP                 XMM       1000X011110X0000
0044 20001  377 102047           RTN       CMPS CAB  CAB       QUICK SELF TEST
0045 20002  324 010207           JMP                 MBI       1000X01111000010
0046 20003  324 010147           JMP                 MBF       1000X01111000011
0047 20004  324 011047           JMP                 MBW       1000X01111000100
0048 20005  324 012707           JMP                 MWI       1000X01111000101
0049 20006  324 012647           JMP                 MWF       1000X01111000110
0050 20007  324 013547           JMP                 MWW       1000X01111000111
0051 20010  324 014507           JMP                 SY*       1000X01111001000
0052 20011  324 015047           JMP                 US*       1000X01111001001
0053 20012  324 014607           JMP                 PA*       1000X01111001010
0054 20013  324 014747           JMP                 PB*       1000X01111001011
0055 20014  324 016247           JMP                 SSM       1000X01111001100
0056 20015  324 016507           JMP                 JRS       1000X01111001101
0057 20016  370 036747           RTN                           1000X01111001110
0058 20017  370 036747           RTN                           1000X01111001111
0059 20020  324 002047           JMP                 XMM       1000X011110X0000
0060 20021  324 002057           JMP  STFL           XMM       1000X01111010001
0061 20022  324 004607           JMP                 XM*       1000X01111010010
0062 20023  370 036747           RTN                           1000X01111010011
0063 20024  324 005547           JMP                 XL*       1000X01111010100
0064 20025  324 006007           JMP                 XS*       1000X01111010101
0065 20026  324 006247           JMP                 XC*       1000X01111010110
0066 20027  324 006647           JMP                 LF*       1000X01111010111
0067 20030  010 022447  RS*                PASS MEU  MEU       1000X01111011000
0068 20031  230 022040  RV*      READ RTN  PASS CAB  MEU       1000X01111011001
0069 20032  324 007047           JMP                 DJP       1000X01111011010
0070 20033  324 007647           JMP                 DJS       1000X01111011011
0071 20034  324 007147           JMP                 SJP       1000X01111011100
0072 20035  324 007547           JMP                 SJS       1000X01111011101
0073 20036  324 007247           JMP                 UJP       1000X01111011110
0074                    *
0075 20037  345 007165  UJS      IMM  DCNT HIGH S4   %103      S4:=USER, CNTR:=16B
0076 20040  324 007704           JMP  RJ30           JS*       START READ ON DEF.
```

```
0077                    ***********************************************************************
PAGE 0006 RTE MICRO-ASSEMBLER REV.A 760818
0079                    *
0080                    *
0081                    *
0082                    *
0083                    *
0084                    ***********************************************************************
0085 20041  010 033107  XMM                 PASS S3   M         S3 := M; SAVE M
0086 20042  230 070547          READ        PASS CNTR X          CNTR :=  COUNT
0087 20043  360 001602          RTN  CNDX ALZ                    TEST FOR ZERO COUNT
0088 20044  342 000507          IMM         LOW  L    %200       L  := 1111111110000000
0089 20045  234 007147          READ        SANL S4   A          MASK LOW 7 BITS OF A-REG
0090 20046  347 076507          IMM         HIGH L    %337       L  := 1101111111111111
0091 20047  011 047147                      SONL S4   S4         ADD CONTROL BIT (13)
0092 20050  010 046447                      PASS MEU  S4         MEM ADDR REG := S4
0093 20051  010 011707                      PASS P    B          P  := B(TABLE ADDRESS)
0094 20052  010 070747                      PASS      X          SET ALU FLAGS FROM X
0095 20053  334 003402          JMP  CNDX FLAG       XMS        TEST FOR XMS INSTRUCTION
0096 20054  327 104142          JMP  CNDX AL15       READMAP    TEST FOR NEGATIVE COUNT
0097 20055  227 174725  MELOOP1 READ DCNT INC  PNM   P          READ NEXT WORD; P := P+1
0098                    *
0099 20056  230 001207          READ        PASS S5   TAB        S5 := MAP DATA - DUMMY READ
0100 20057  007 106147                      INC  A    A          A := A+1
0101 20060  010 050452                 MESP PASS MEU  S5         MAP REG := DATA
0102 20061  000 071607                      DEC  X    X          X := X-1
0103 20062  320 003302          JMP  CNDX ALZ        XMM.RTN    IF DONE THEN BUG OUT
0104 20063  324 042642          JMP  CNDX CNT4 RJS   MELOOP1    LOOP FOR 16X
0105 20064  335 002642          JMP  CNDX NINT       MELOOP1    TEST FOR NO INTERRUPT
0106 20065  000 045107  XMM.EXIT            DEC  S3   S3           ELSE SERVICE INTERRUPT
0107                    *
0108 20066  010 074207  XMM.RTN             PASS B    P          RESET B-REG
0109 20067  227 144700  P.RTN   READ RTN    INC  PNM  S3         P  := NEXT INSTRUCTION; START REA
0110                    *
PAGE 0007 RTE MICRO-ASSEMBLER REV.A 760818
0112                    *
0113                    *
0114                    *
0115                    *
0116                    *
0117                    ***********************************************************************
0118 20070  327 103342  XMS     JMP  CNDX AL15       P.RTN      TEST FOR X<0 ... NOP
0119 20071  230 036747  MELOOP2 READ                             FOR DCPC
0120 20072  007 106147                      INC  A    A          A := A+1
0121 20073  010 010452                 MESP PASS MEU  B          MAP REG := DATA
0122 20074  007 110225                 DCNT INC  B    B          B  := B + 1; INC CNTR
0123 20075  000 071607                      DEC  X    X          X := X-1
0124 20076  320 003342          JMP  CNDX ALZ        P.RTN      IF DONE THEN BUG OUT
0125 20077  324 043442          JMP  CNDX CNT4 RJS   MELOOP2    LOOP FOR 16X
0126 20100  335 003442          JMP  CNDX NINT       MELOOP2    TEST FOR NO INTERRUPT
0127 20101  000 045107                      DEC  S3   S3         RESET P REGISTER FOR RESTART
0128 20102  227 144700          READ RTN    INC  PNM  S3            SERVICE INTERRUPT
0129                    *
0130                    READMAP EQU                  *
0131 20103  227 174726  MELOOP3 READ ICNT INC  PNM  P          P := P+1 - DUMMY READ
0132 20104  007 106147                      INC  A    A          A := A+1
0133 20105  010 023212                 MESP PASS S5   MEU        S5 := MAP REG
0134 20106  210 050036          WRTE MPCK PASS TAB   S5         WRITE DATA INTO TABLE
0135 20107  007 171607                      INC  X    X              X := X-1
0136 20110  320 003302          JMP  CNDX ALZ        XMM.RTN        IF DONE THEN BUG OUT
0137 20111  324 044142          JMP  CNDX CNT4 RJS   MELOOP3    LOOP FOR 16X
0138 20112  335 004142          JMP  CNDX NINT       MELOOP3    TEST FOR NO INTERRUPT
0139 20113  324 003247          JMP                  XMM.EXIT   ELSE SERVICE INTERRUPT
```

```
PAGE 0008 RTE MICRO-ASSEMBLER REV.A 760818
0141                    *
0142                    *
0143                    *
0144                    *
0145                    *
0146                    ***********************************************************************
0147 20114  357 077147  XM*      IMM        CMHI S4   %337      S4 := 0010000000000000
0148 20115  150 002762           LWF  L1    PASS      CAB       T-BUS := A/B; FLAG := A/B(15)
0149 20116  321 145042  PA.PB    JMP  CNDX  AL0  RJS  SY.US     TEST FOR PORT.A MAP
0150 20117  341 176507           IMM        LOW  L    %177      L  := 1111111101111111
0151 20120  231 047147           READ       SONL S4   S4        S4 := 0010000010000000
0152 20121  334 045202  SY.US    JMP  CNDX  FLAG RJS  XFER      TEST FOR SYSTEM MAP
0153 20122  343 076507           IMM        LOW  L    %337      L  := 1111111111011111
0154 20123  231 047147           READ       SONL S4   S4        S4 := 00100000X0100000
0155 20124  010 046447  XFER                PASS MEU  S4        MEM ADDR REG := S4(7-0)
0156 20125  340 100547           IMM        LOW  CNTR %40       CNTR := 32
0157 20126  230 036765  XFERLOOP READ DCNT                      DUMMY READ
0158 20127  010 036747                      PASS                FOR MEB DELETE WITH DUMMY READ
0159 20130  010 022452                MESP PASS MEU  MEU        MEM PORT REG := MEM PROG REG
0160 20131  326 145302           JMP  CNDX  CNT8 RJS  XFERLOOP IF NOT DONE THEN LOOP
0161 20132  230 036740  RTN*     READ RTN                        RETURN
0162                    ***********************************************************************
PAGE 0009 RTE MICRO-ASSEMBLER REV.A 760818
0164                    *
0165                    *
0166                    *
0167                    *
0168                    *
0169                    ***********************************************************************
0170 20133  300 012447  XL*      JSB                   INDIRECT  GET OPERAND ADDR FROM INSTR + 1
0171 20134  010 036752                MESP                        SWITCH MAP STATE
0172 20135  230 036747           READ                             START CROSS LOAD START CROSS LOAD
0173 20136  010 000052                MESP PASS CAB  TAB          CAB := DATA, RESET MAPS
0174 20137  227 174700           READ RTN  INC  PNM  P           RETURN TO FETCH
0175                    ***********************************************************************
0176 20140  300 012447  XS*      JSB                   INDIRECT  GET OPERAND ADDR FROM INSTR + 1
0177 20141  010 036752                MESP                        SWITCH MAP STATE
0178 20142  210 002036           WRTE MPCK PASS TAB  CAB
0179 20143  010 022447                     PASS MEU  MEU         RESET MAP STATE
0180 20144  227 174700           READ RTN  INC  PNM  P           START NEXT INST READ - EXIT
0181                    ***********************************************************************
0182 20145  300 012447  XC*      JSB                   INDIRECT  GET OPERAND ADDR FROM INSTR + 1
0183 20146  010 002512                MESP PASS L    CAB         L  := A/B; SET ALTERNATE MAP
0184 20147  230 036747           READ                             READ REAL OPERAND
0185 20150  010 001012                MESP PASS S1   TAB         S1 := DATA, RESET MAPS
0186 20151  227 174707           READ       INC  PNM  P          START READ FOR NEXT INST.
0187 20152  014 140747                      XOR       S1         COMPARE DATA
0188 20153  360 001002           RTN  CNDX  ALZ                  PTN-DON'T SKIP IF EQUAL
0189 20154  227 174700           READ RTN  INC  PNM  P           P  := INSTR + 2; RETURN
0190                    ***********************************************************************
0191 20155  344 016507  LF*      IMM        HIGH L    %007      L  := 0000011111111111
0192 20156  232 003147           READ       AND  S4   CAB       S4 := A/B(10-0) BEWARE THE READ
0193 20157  010 022447                      PASS MEU  MEU       SEND "FENCE" DIRECTIVE
0194 20160  370 046447           RTN        PASS MEU  S4        MEM FENCE := S4
```

```
0195                    ***********************************************************************
PAGE 0010 RTE MICRO-ASSEMBLER REV.A 760818
0197                    *
0198                    *
0199                    *
0200                    *
0201                    *
0202                    ***********************************************************************
0203 20161  345 001147  DJP      IMM        HIGH S4   %100      S4 := 0100000011111111
0204 20162  324 007307           JMP                  JP*
0205                    *
0206 20163  345 005147  SJP      IMM        HIGH S4   %102      S4 := 0100001011111111
0207 20164  324 007307           JMP                  JP*
0208                    *
0209 20165  345 007147  UJP      IMM        HIGH S4   %103      S4 := 0100001111111111
0210 20166  230 036747  JP*      READ
0211 20167  304 017377           JSE  IOFF            OPGET     GET OPERAND ADDR FROM INSTR + 1
0212 20170  010 046447  JMPSTAT            PASS MEU  S4        MEM STATUS IS SET HERE
0213 20171  227 133736           READ MPCK INC  P     M         CHECK TARGET ; START INST READ
0214 20172  370 036747           RTN                            RETURN
0215                    ***********************************************************************
0216 20173  345 005166  SJS      IMM  ICNT HIGH S4   %102      S4 := 0100001011111111
0217 20174  324 007704           JMP  RJ30            JS*       START READ ON DEF.
0218                    *
0219 20175  345 001144  DJS      IMM  RJ30 HIGH S4   %100      S4 := 0100000011111111
0220                    *
0221                             ORG                  20176B    !!!DO NOT MOVE--INDEXED ENTRY FRO
0222 20176  304 017377  JS*      JSB  IOFF            OPGET     GET OPERAND ADDR FROM INSTR + 1
0223 20177  010 046447                      PASS MEU  S4        MEM STATUS IS SET HERE
0224 20200  210 074036           WRTE MPCK PASS TAB  P         WRITE RETURN ADDR AT TARGET
0225 20201  007 133707                      INC  P    M         P := TARGET ADDRESS
0226 20202  227 174700  JS*EXIT  READ RTN  INC  PNM  P         P  := TARGET + 1
```

```
0227                    ************************************************************************
PAGE 0011 RTE MICRO-ASSEMBLER REV.A 760818
0229                    *
0230                    *
0231                    *
0232                    *
0233                    *
0234                    ************************************************************************
0235 20203  010 036752  MBF          MESP                        SET ALTERNATE MAP
0236 20204  304 012407  MBI      JSB                  BYTEADJ     ADJUST FOR FULL WORD PROCESSING
0237 20205  304 013007           JSB                  X.LOOP-1    MOVE BYTES IN PAIRS
0238 20206  010 070747                    PASS        X           ALU FLAGS := X CONDITIONS
0239 20207  320 052142           JMP  CNDX ALZ  RJS   B.RESET     TEST FOR INTERRUPTED MOVE
0240 20210  334 052202           JMP  CNDX FLAG RJS   B.RESET+1   TEST FOR NO ODD BYTE
0241 20211  344 000507           IMM       HIGH L     %000        L := 0000000011111111
0242 20212  230 026747           READ      PASS       CNTR        AL0 := IR(0); START DCPC READ
0243 20213  321 150642           JMP  CNDX AL0  RJS   *+2         TEST FOR MBI INSTRUCTION
0244 20214  010 036752               MESP                         SET ALTERNATE MAP
0245 20215  230 006647           READ      PASS M     A           M  := SOURCE ADDRESS, RESET MAPS
0246 20216  010 006162               L1    PASS A     A           FORM BYTE ADDRESS IN A
0247 20217  014 001152               MESP  SANL S4    TAB         S4 := AAAAAAAA00000000
0248 20220  324 011507           JMP                  MB*
0249                    ************************************************************************
0250 20221  344 000512  MBW      IMM  MESP HIGH L     %000        SET THE OPPOSITE MAP L := BYTE MA
0251 20222  304 012407           JSB                  BYTEADJ     ADJUST FOR FULLWORD PROCESSING
0252 20223  304 013647           JSB                  W.LOOP-1    MOVE BYTES IN PAIRS
0253 20224  010 070752               MESP  PASS       X           ALU := X; SELECT ALTERNATE MAP
0254 20225  320 052142           JMP  CNDX ALZ  RJS   B.RESET     TEST FOR INTERRUPTED MOVE
0255 20226  334 052202           JMP  CNDX FLAG RJS   B.RESET+1   TEST FOR NO ODD BYTE
0256 20227  230 006647           READ      PASS M     A           M  := SOURCE ADDRESS
0257 20230  010 006162               L1    PASS A     A           FORM BYTE ADDRESS IN A
0258 20231  014 001147                     SANL S4    TAB         S4 := AAAAAAAA00000000
0259                    *
0260 20232  230 010647  MB*      READ      PASS M     B           M  := DESTINATION ADDRESS
0261 20233  010 010222               L1    PASS B     B           FORM BYTE ADDRESS IN B
0262 20234  012 000507                     AND  L     TAB         L  := 00000000BBBBBBBB
0263 20235  010 147147                     IOR  S4    S4          S4 := AAAAAAAABBBBBBBB
0264 20236  210 046036           WRTE MPCK PASS TAB   S4          WRITE DATA INTO DESTINATION
0265 20237  007 106147                     INC  A     A           A  := A + 1
0266 20240  010 022447                     PASS MEU   MEU         RESET SELECTED MAP
0267 20241  007 110207                     INC  B     B           B  := B + 1
0268 20242  227 144700           READ RTN  INC  PNM   S3
0269                    ************************************************************************
0270 20243  150 071622  B.RESET  LWF  L1   PASS X     X           RESET X IN BYTES
0271 20244  010 022447                     PASS MEU   MEU         RESET SELECTED MAP
0272 20245  227 144707           READ      INC  PNM   S3          EXIT
0273 20246  010 006162               L1    PASS A     A           RESET A FOR EVEN BYTE ADDRESS
0274 20247  370 010222           RTN  L1   PASS B     B           RESET B FOR EVEN BYTE ADDRESS
0275                    ************************************************************************
0276 20250  010 033116  BYTEADJ      CLFL PASS S3     M           SAVE M FOR NEXT INST FETCH
0277 20251  010 006164               R1   PASS A      A           A := SOURCE WORD ADDRESS
0278 20252  010 010224               R1   PASS B      B           B := DESTINATION WORD ADDRESS
0279 20253  150 071624           LWF  R1   PASS X     X           X := WORD COUNT. FLAG := ODD BYTE
0280 20254  370 070747           RTN       PASS       X           SET ALU FLAGS FOR TESTING X
```

```
PAGE 0012 RTE MICRO-ASSEMBLER REV.A 760818
0282                        *
0283                        *
0284                        *
0285                        *
0286                        *
0287                        ***********************************************************************
0288 20255  010 036752  MWF            MESP                        FLIP THE MAP SO IT WILL COME OUT
0289 20256  010 033107  MWI                  PASS S3   M           SAVE M FOR NEXT INST
0290 20257  010 070747                       PASS      X           ALU FLAGS := X CONDITIONS
0291 20260  320 014402            JMP  CNDX ALZ       MW*         TEST FOR X=0
0292 20261  230 006647  X.LOOP    READ       PASS M    A           READ SOURCE WORD
0293 20262  007 106147                       INC  A    A           INCR. SOURCE ADDR.; SWITCH MAPS
0294 20263  010 010652            MESP PASS M    B           M.P. CHECK,M := DEST ADDR
0295 20264  010 001147                       PASS S4   TAB         S4 := DATA
0296 20265  210 046036            WRTE MPCK PASS TAB  S4          WRITE DATA INTO DESTINATION
0297 20266  007 110207                       INC  B    B           INCREMENT DESTINATION ADDRESS
0298 20267  000 071612            MESP DEC  X    X           DECREMENT COUNT; SWITCH MAPS
0299 20270  320 014402            JMP  CNDX ALZ       MW*         TEST IF MOVE COMPLETE
0300 20271  335 013042            JMP  CNDX NINT      X.LOOP      TEST FOR NO INTERRUPT
0301 20272  324 014347            JMP                 MWINT
0302                        ***********************************************************************
0303 20273  010 033112  MWW            MESP PASS S3   M           SAVE M FOR NEXT INST FETCH
0304 20274  010 070747                       PASS      X           T-BUS := X
0305 20275  320 014402            JMP  CNDX ALZ       MW*         TEST FOR X=0
0306 20276  230 006647  W.LOOP    READ       PASS M    A           READ SOURCE WORD
0307 20277  007 106147                       INC  A    A           INCREMENT SOURCE ADDRESS
0308 20300  010 001147                       PASS S4   TAB         S4 := DATA
0309 20301  010 010647                       PASS M    B           M.P.CHECK; M := DEST ADDRESS
0310 20302  210 046036            WRTE MPCK PASS TAB  S4          WRITE DATA INTO DESTINATION
0311 20303  007 110207                       INC  B    B           INCREMENT DESTINATION ADDRESS
0312 20304  000 071607                       DEC  X    X           DECREMENT COUNT
0313 20305  320 014402            JMP  CNDX ALZ       MW*         TEST IF MOVE COMPLETE
0314 20306  335 013702            JMP  CNDX NINT      W.LOOP      TEST FOR NO INTERRUPT
0315 20307  000 045107  MWINT                DEC  S3   S3          SET P COUNTER FOR INTERUPT EXIT
0316 20310  010 022447  MW*                  PASS MEU  MEU         RESET SELECTED MAP; RETURN
0317 20311  227 144700            READ RTN  INC  PNM  S3          START INST FETCH; EXIT
0318                        ***********************************************************************
PAGE 0013 RTE MICRO-ASSEMBLER REV.A 760818
0320                        *
0321                        *
0322                        *
0323                        *
0324                        *
0325                        ***********************************************************************
0326 20312  357 077147  SY*       IMM       CMHI S4   %337        S4 := 0010000000000000
0327 20313  324 015207            JMP                 MAPMOVE
0328                        ***********************************************************************
0329 20314  355 175164  PA*       IMM  R1   CMHI S4   %176        S4 := 0100000010000000
0330 20315  010 047164                 R1   PASS S4   S4          S4 := 0010000001000000
0331 20316  324 015207            JMP                 MAPMOVE
0332                        ***********************************************************************
0333 20317  342 077147  PB*       IMM       LOW  S4   %237        S4 := 1111111110011111
0334 20320  324 015107            JMP                 US*+1       L  := 1101111111111111
0335                        *                                     S4 := 0010000001100000
0336                        ***********************************************************************
0337 20321  343 077147  US*       IMM       LOW  S4   %337        S4 := 1111111111011111
0338 20322  347 076507            IMM       HIGH L    %337        L  := 1101111111111111
0339 20323  014 147147                      XOR  S4   S4          S4 := 0010000000100000
0340 20324  230 033107  MAPMOVE   READ       PASS S3   M           S3 := M - DUMMY READ
0341 20325  010 046447                       PASS MEU  S4          MEM ADDR REG := S4
0342 20326  340 100547            IMM       LOW  CNTR 32                := 32
0343 20327  010 003707                       PASS P    CAB         P  := A/B
0344 20330  327 115742            JMP  CNDX AL15      MELOOP5     AL15=1 => READ MAPS
0345                        *
0346 20331  227 174725  MELOOP4   READ DCNT INC  PNM  P           READ NEXT WORD; P := P + 1
0347 20332  230 001207            READ       PASS S5   TAB         S5 := MAP DATA - DUMMY READ
0348 20333  010 074047                       PASS CAB  P           A OR B := P
0349 20334  010 050452            MESP PASS MEU  S5          MAP REG := DATA
0350 20335  326 155442            JMP  CNDX CNT8 RJS  MELOOP4     LOOP FOR 32X
0351 20336  227 144700            READ RTN  INC  PNM  S3          P  := INSTR + 1
0352                        *
0353 20337  227 174725  MELOOP5   READ DCNT INC  PNM  P           DEC CNTR P := P + 1  -DUMMY READ
0354 20340  010 074047                       PASS CAB  P           A OR B := P
0355 20341  010 023212            MESP PASS S5   MEU         S5 := MAP DATA
0356 20342  210 050036            WRTE MPCK PASS TAB  S5          WRITE DATA INTO TABLE
0357 20343  326 155742            JMP  CNDX CNT8 RJS  MELOOP5     LOOP FOR 32X
0358 20344  227 144700            READ RTN  INC  PNM  S3          P  := INSTR + 1
```

```
0359                    ***********************************************************************
PAGE 0014 RTE MICRO-ASSEMBLER REV.A 760818
0361                    *
0362                    *
0363                    *
0364                    *
0365                    *
0366                    ***********************************************************************
0367 20345  300 012447  SSM      JSB                     INDIRECT  GET OPERAND ADDR FROM INSTR + 1
0368 20346  010 022447                     PASS MEU  MEU        SEND "STATUS" DIRECTIVE
0369 20347  010 023007                     PASS S1   MEU        WRITE STATUS WORD INTO MEMORY
0370 20350  210 040036           WRTE MPCK PASS TAB  S1
0371 20351  324 010107           JMP                     JS*EXIT
0372                    ***********************************************************************
0373 20352  230 036747  JRS      READ
0374 20353  304 017377           JSB  IOFF               OPGET     GET STATUS WORD FROM ^.INSTR+1
0375 20354  150 001222           LWF  L1   PASS S5       TAB       FLAG := STAT(15); S5(15) := STAT(
0376 20355  345 007147           IMM       HIGH S4       %103      S4 := 0100001111111111
0377 20356  230 074647           READ      PASS M        P         SET M FOR SECOND OPERAND
0378 20357  304 017377           JSB  IOFF               OPGET     GET TARGET ADDR FROM INSTR+2
0379 20360  334 017102  ON.OFF   JMP  CNDX FLAG          SY.USR    TEST IF MEM WAS ON
0380 20361  345 003147           IMM       HIGH S4       %101      IF OFF, S4 := 0100000111111111
0381 20362  230 050747  SY.USR   READ      PASS          S5        AL15 := STAT(14) -DUMMY READ
0382 20363  327 107402           JMP  CNDX AL15          JMPSTAT   TEST STAT(14) FOR USER SELECTED
0383 20364  345 004507           IMM       HIGH L        %102      IF SYS, L  := 0100001011111111
0384 20365  232 047147           READ      AND  S4       S4          THEN  S4 := 0100000X011111111
0385 20366  324 007407           JMP                     JMPSTAT   SET STATUS OF MEM; ALSO SET P
0386                    ***********************************************************************
0387 20367  340 006547  OPGET    IMM       LOW  CNTR     003B      SET COUNTER FOR MAXIMUM INDIRECTS
0388 20370  230 000665           READ DCNT PASS M        TAB       M := T/A/B  DEC INDIRECT CNTR
0389 20371  367 140002           RTN  CNDX AL15 RJS                RETURN IF INDIRECT RESOLVED
0390 20372  326 157402           JMP  CNDX CNTR RJS      *-2       CONTINUE IF IND. LEVEL <= 3
0391 20373  230 036743           READ ION                          RE-ENABLE INTERRUPT RECOGNITION
0392 20374  323 157342           JMP  CNDX HOI  RJS      OPGET     TEST FOR HALT OR INTERRUPT
0393 20375  320 012747           JMP                     INDIRECT+6 PROCESS PENDING INTERRUPT
0394                    ***********************************************************************
0395                             END
END OF PASS 2: NO ERRORS

PAGE 0015 RTE MICRO CROSS-REFERENCE REV.1813 771212
SYMBOLS=0059  REFERENCES=0081  SOURCE LINES=0395
B.RESET    0270  0239  0240  0254  0255
BYTEADJ    0276  0236  0251
DJP        0203  0069
DJS        0219  0070
HORI       0019  **NOT REFERENCED**
INDIRECT   0018  0170  0176  0182  0367  0393
JMPSTAT    0212  0382  0385
JP*        0210  0204  0207
JRS        0373  0056
JS*        0222  0076  0217
JS*EXIT    0226  0371
JTABL      0043  **NOT REFERENCED**
LF*        0191  0066
MAPMOVE    0340  0327  0331
MB*        0260  0248
MBF        0235  0046
MBI        0236  0045
MBW        0250  0047
MELOOP1    0097  0104  0105
MELOOP2    0119  0125  0126
MELOOP3    0131  0137  0138
MELOOP4    0346  0350
MELOOP5    0353  0344  0357
MW*        0316  0291  0299  0305  0313
MWF        0288  0049
MWI        0289  0048
MWINT      0315  0301
```

```
PAGE 0016 RTE MICRO CROSS-REFERENCE REV.1813 771212
MWW        0303  0050
ON.OFF     0379  **NOT REFERENCED**
OPGET      0387  0211  0222  0374  0378  0392
P.RTN      0109  0118  0124
PA*        0329  0053
PA.PB      0149  **NOT REFERENCED**
PB*        0333  0054
READMAP    0130  0096
RS*        0067  **NOT REFERENCED**
RTN*       0161  **NOT REFERENCED**
RV*        0068  **NOT REFERENCED**
SJP        0206  0071
SJS        0216  0072
SSM        0367  0055
SY*        0326  0051
SY.US      0152  0149
SY.USR     0381  0379
UJP        0209  0073
UJS        0075  **NOT REFERENCED**
US*        0337  0052  0334
W.LOOP     0306  0252  0314
X.LOOP     0292  0237  0300
XC*        0182  0065
XFER       0155  0152
XFERLOOP   0157  0160
XL*        0170  0063
XM*        0147  0061
XMM        0085  0043  0059  0060
PAGE 0017 RTE MICRO CROSS-REFERENCE REV.1813 771212
XMM.EXIT   0106  0139
XMM.RTN    0108  0103  0136
XMS        0118  0095
XS*        0176  0064
```

```
PAGE 0001 RTE MICRO-ASSEMBLER REV.A 760818
0001  MICMXE,L,C
END OF PASS 1: NO ERRORS
PAGE 0002 RTE MICRO-ASSEMBLER REV.A 760818
0001                        MICMXE,L,C
0002                        *CODE='SIS::32,REPLACE
0003                        ************************************************************
0004                        *
0005                        *
0006                        *  SCIENTIFIC INSTRUCTION SET MICROCODE
0007                        *  FOR 21MX - E AND F SERIES COMPUTERS
0008                        *
0009                        *  JAN. 24, 1978
0010                        *
0011                        *
0012                        ************************************************************
0013                        FETCH    EQU                 00000B
0014                        HORI     EQU                 00006B
0015                                 ORG                 24000B
0016 24000  325 012047               JMP                 TAN
0017 24001  325 041007               JMP                 SQRT
0018 24002  325 023047               JMP                 ALOG
0019 24003  325 027647               JMP                 ATAN
0020 24004  325 015647               JMP                 COS
0021 24005  325 015747               JMP                 SIN
0022 24006  325 044447               JMP                 EXP
0023 24007  325 027207               JMP                 ALOGT
0024 24010  325 050147               JMP                 TANH
0025 24011  230 036740               READ RTN
0026 24012  230 036740               READ RTN
0027 24013  230 036740               READ RTN
0028 24014  230 036740               READ RTN
0029 24015  230 036740               READ RTN
0030 24016  230 036740               READ RTN
0031 24017  325 035347               JMP                 SELFTEST
```

```
PAGE 0003 RTE MICRO-ASSEMBLER REV.A 760818
0033                        ************************************************************
0034                        *
0035                        *  SUBROUTINE FLUN
0036                        *
0037                        *  ENTER:  B = LOW PART OF FLOATING POINT NUMBER
0038                        *  RETURN: A = EXPONENT  B = LOW MANTISSA
0039                        *
0040 24020  340 000516  FLUN     IMM  CLFL LOW  L    000B      L = 177400B
0041 24021  154 010164          LWF  R1   SANL A    B         GET EXPONENT IN A
0042 24022  012 010207                    AND  B    B         PUT MANTISSA IN B
0043 24023  374 040202          RTN  CNDX FLAG RJS            RETURN IF EXP POSITIVE
0044 24024  342 000507          IMM       LOW  L    200B      L = 177600B
0045 24025  370 106147          RTN       IOR  A    A         RETURN, EXTEND SIGN BIT
0046                        *
0047                        ************************************************************
0048                        *
0049                        *  SUBROUTINE PWR2
0050                        *
0051                        *  ENTER:  FLOATING POINT NUMBER IN BOX
0052                        *          INTEGER IN S11
0053                        *  RETURN: (A B) = X*2**S11, P = P + 1
0054                        *
0055 24026  306 043242  PWR2     JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0056 24027  010 020172               MPP1 PASS A    MPPB      GET A FROM BOX
0057 24030  320 002642          JMP  CNDX ALZ       PDONE     EXIT IF X = 0
0058 24031  010 020232               MPP1 PASS B    MPPB      GET B FROM BOX
0059 24032  340 000516          IMM  CLFL LOW  L    000B      L  = 177400B
0060 24033  154 011024          LWF  R1   SANL S1   B         S1 = EXPONENT
0061 24034  334 041742          JMP  CNDX FLAG RJS  PNEXT1    JUMP IF EXP POSTIVE
0062 24035  342 000507          IMM       LOW  L    200B      L  = 177600B
0063 24036  010 141007                    IOR  S1   S1        EXTEND EXP SIGN BIT
0064 24037  010 040507  PNEXT1            PASS L    S1        PUT EXPONENT IN L
0065 24040  143 064762          LWF  L1   ADD       S11       SET NEW EXP SIGN BIT
0066 24041  143 065022          LWF  L1   ADD  S1   S11       S1 = NEW EXP
0067 24042  340 000507          IMM       LOW  L    000B      L  = 177400B
0068                        *
0069                        *  CHECK FOR UNDER/OVERFLOW
0070                        *
0071 24043  012 040753               COV  AND       S1
0072 24044  320 002402          JMP  CNDX ALZ       PNEXT2    JUMP IF NO OVERFLOW
0073 24045  011 040747                    SONL       S1
0074 24046  320 102402          JMP  CNDX ONES      PNEXT2    JUMP IF NO UNDERFLOW
0075 24047  370 036754          RTN  SOV                      OVER/UNDERFLOW RETURN
0076                        *
0077 24050  014 041007  PNEXT2            SANL S1   S1        S1 = 000EXP
0078 24051  012 010507                    AND  L    B         L  = MAN000
0079 24052  010 140207                    IOR  B    S1        B  = MANEXP
0080 24053  227 174707          READ      INC  PNM  P         START READ
0081 24054  370 036747          RTN                           RETURN
0082                        *
0083 24055  010 020232  PDONE         MPP1 PASS B    MPPB      GET B
0084 24056  227 174707          READ      INC  PNM  P         START READ
0085 24057  370 036747          RTN                           RETURN
0086                        *
```

```
0087                        ************************************************************
PAGE 0004 RTE MICRO-ASSEMBLER REV.A 760818
0089                        ************************************************************
0090                        *
0091                        *  SUBROUTINE FMPY
0092                        *
0093                        *  STARTS BOX MULTIPLY ON ACCUMULATOR
0094                        *  AND REGISTERS (S2 S3)
0095                        *
0096                        *
0097 24060  306 043242  FMPY     JSB  CNDX MPP  RJS  WAIT1      WAIT FOR BOX
0098 24061  340 110607           IMM       LOW  IRCM 044B       BOX MPY OPCODE
0099 24062  010 036751                MPP2                       START BOX
0100 24063  010 042432                MPP1 PASS MPPB S2          SEND OP1 = (S2 S3)
0101 24064  370 044432           RTN  MPP1 PASS MPPB S3          RETURN
0102                        *
0103                        *
0104                        ************************************************************
0105                        *
0106                        *
0107                        *  SUBROUTINE WAIT1
0108                        *
0109                        *  WAITS FOR BOX TO COMPLETE EXECUTION
0110                        *  LOOPS FOR COUNT OF 32, THEN ABORTS
0111                        *  ON ABORT:  A SET TO ALL ONES
0112                        *             B RESTORED
0113                        *             GENERATE MP INT
0114                        *
0115                        *
0116 24065  340 100547  WAIT1    IMM       LOW  CNTR 040B       CNTR=32
0117 24066  323 104102           JMP  CNDX HOI       INTRT1    CHECK FOR INTERRUPTS
0118 24067  366 004102  LOOP1    RTN  CNDX MPP                 RETURN WHEN DONE
0119 24070  010 036765                DCNT                      DECREMENT CNTR
0120 24071  366 000742           RTN  CNDX MPP                 RETURN WHEN DONE
0121 24072  326 143342           JMP  CNDX CNT8 RJS  LOOP1     ELSE LOOP1 32 TIMES
0122 24073  011 136154                SOV  ONE  A               SET A = 177777B
0123 24074  355 165047           IMM       CMHI S2   172B      SET IRCM = MIA 00
0124 24075  010 042607                     PASS IRCM S2
0125 24076  010 052206                IOG  PASS B    S6         RESTORE B, MP INT
0126 24077  000 075707                     DEC  P    P          SET P = ERROR ADDR
0127 24100  227 174707           READ      INC  PNM  P          START READ
0128 24101  320 000007           JMP                 FETCH      RETURN
0129                        *
0130                        ************************************************************
0131                        *
0132                        *  INTERRUPT ROUTINE
0133                        *
0134                        *  RESTORES A,B,P AND RETURNS
0135                        *
0136                        *
0137 24102  336 043342  INTRT1   JMP  CNDX NSNG RJS  LOOP1      RTN IF SINGLE STEP
0138 24103  000 075732                MPP1 DEC  P    P          RESET BOX, SET ADDR
0139 24104  010 050147                     PASS A    S5         RESTORE A
0140 24105  010 052207                     PASS B    S6         RESTORE B
0141 24106  320 000307           JMP                 HORI
0142                        *
```

```
0143                    ************************************************************
PAGE 0005 RTE MICRO-ASSEMBLER REV.A 760818
0145                    ************************************************************
0146                    *
0147                    *  SUBROUTINE FSUB2
0148                    *
0149                    *  STARTS BOX SUB ON ACC AND (S2 S3)
0150                    *
0151 24107  306 043242  FSUB2    JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0152 24110  340 060607           IMM       LOW  IRCM 030B     BOX SUB OPCODE
0153 24111  010 036751                MPP2                      START BOX
0154 24112  010 042432                MPP1 PASS MPPB S2         SEND OP1 = (S2 S3)
0155 24113  370 044432           RTN  MPP1 PASS MPPB S3         RETURN
0156                    *
0157                    ************************************************************
0158                    *
0159                    *  SUBROUTINE FADD
0160                    *
0161                    *  STARTS BOX ADD ON ACC AND (S2 S3)
0162                    *
0163 24114  306 043242  FADD     JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0164 24115  340 020607           IMM       LOW  IRCM 010B     BOX ADD OPCODE
0165 24116  010 036751                MPP2                      START BOX
0166 24117  010 042432                MPP1 PASS MPPB S2         SEND OP1 = (S2 S3)
0167 24120  370 044432           RTN  MPP1 PASS MPPB S3         RETURN
0168                    *
0169                    ************************************************************
0170                    *
0171                    *  SUBROUTINE FDIV7
0172                    *
0173                    *  STARTS BOX DIV ON (S7 S8) AND ACC
0174                    *
0175 24121  306 043242  FDIV7    JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0176 24122  340 150607           IMM       LOW  IRCM 064B     BOX DIV OPCODE
0177 24123  010 036751                MPP2                      START BOX
0178 24124  010 054432                MPP1 PASS MPPB S7         SEND OP1 = (S7 S8)
0179 24125  370 056432           RTN  MPP1 PASS MPPB S8         RETURN
0180                    *
0181                    ************************************************************
0182                    *
0183                    *  SUBROUTINE XSQ
0184                    *
0185                    *  SAVES ACC IN (S7 S8) AND STARTS ACC*ACC
0186                    *
0187 24126  306 043242  XSQ      JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0188 24127  340 100607           IMM       LOW  IRCM 040B     BOX MPY OPCODE
0189 24130  010 021332                MPP1 PASS S7   MPPB       SAVE IN (S7 S8)
0190 24131  010 021372                MPP1 PASS S8   MPPB
0191 24132  010 036751                MPP2                      START BOX
0192 24133  010 054432                MPP1 PASS MPPB S7         SEND OP1 = (S7 S8)
0193 24134  010 056432                MPP1 PASS MPPB S8
0194 24135  010 054432                MPP1 PASS MPPB S7         SEND OP2 = (S7 S8)
0195 24136  370 056432           RTN  MPP1 PASS MPPB S8         RETURN
0196                    *
```

```
0197                    ************************************************************
PAGE 0006 RTE MICRO-ASSEMBLER REV.A 760818
0199                    ************************************************************
0200                    *
0201                    *  SUBROUTINE FADA2
0202                    *
0203                    *  STARTS BOX ADD ON (A B) AND (S2 S3)
0204                    *
0205 24137  306 043242  FADA2    JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0206 24140  340 000607           IMM       LOW  IRCM 000B      BOX ADD OPCODE
0207 24141  010 036751  BAB23         MPP2                     START BOX
0208 24142  010 006432                MPP1 PASS MPPB A         SEND OP1 = (A B)
0209 24143  010 010432                MPP1 PASS MPPB B
0210 24144  010 042432                MPP1 PASS MPPB S2        SEND OP2 = (S2 S3)
0211 24145  370 044432           RTN  MPP1 PASS MPPB S3        RETURN
0212                    *
0213                    ************************************************************
0214                    *
0215                    *  SUBROUTINE FDAVC2
0216                    *
0217                    *  STARTS BOX DIV ON ACC AND (S2 S3)
0218                    *
0219 24146  306 043242  FDVAC2   JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0220 24147  340 150607           IMM       LOW  IRCM 064B      BOX DIV OPCODE
0221 24150  010 036751                MPP2                     START BOX
0222 24151  010 042432                MPP1 PASS MPPB S2        SEND OP1 = (S2 S3)
0223 24152  370 044432           RTN  MPP1 PASS MPPB S3        RETURN
0224                    *
0225                    ************************************************************
0226                    *
0227                    *  SUBROUTINE VMPY
0228                    *
0229                    *  STARTS BOX MPY ON ACC AND (S9 S10)
0230                    *
0231 24153  306 043242  VMPY     JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0232 24154  340 110607           IMM       LOW  IRCM 044B      BOX MPY OPCODE
0233 24155  010 036751                MPP2                     START BOX
0234 24156  010 060432                MPP1 PASS MPPB S9        SEND OP1 = (S9 S10)
0235 24157  370 062432           RTN  MPP1 PASS MPPB S10       RETURN
0236                    *
0237                    ************************************************************
0238                    *
0239                    *  SUBROUTINE FSUB
0240                    *
0241                    *  STARTS SUB ON (S2 S3) AND ACC
0242                    *
0243 24160  006 037047  NEGATE                ZERO S2          SET (S2 S3) = 0
0244 24161  006 037107                        ZERO S3
0245                    *
0246                    *
0247 24162  306 043242  FSUB     JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0248 24163  340 050607           IMM       LOW  IRCM 024B      BOX SUB OPCODE
0249 24164  010 036751                MPP2                     START BOX
0250 24165  010 042432                MPP1 PASS MPPB S2        SEND OP1 = (S2 S3)
0251 24166  370 044432           RTN  MPP1 PASS MPPB S3
0252                    *
```

```
0253                    ***************************************************************
PAGE 0007 RTE MICRO-ASSEMBLER REV.A 760818
0255                    ***************************************************************
0256                    *
0257                    *  SUBROUTINE REDUCE
0258                    *
0259 24167  345 043047  FOPI     IMM       HIGH S2   121B      SET (S2 S3 S4) TO 4/PI
0260 24170  341 170507           IMM       LOW  L    174B
0261 24171  012 043047                     AND  S2   S2
0262 24172  347 003107           IMM       HIGH S3   301B
0263 24173  342 156507           IMM       LOW  L    267B
0264 24174  012 045107                     AND  S3   S3
0265 24175  344 117147           IMM       HIGH S4   047B
0266 24176  340 004507           IMM       LOW  L    002B
0267 24177  012 047147                     AND  S4   S4
0268                    *
0269                    *
0270 24200  340 102607  REDUCE   IMM       LOW  IRCM 041B      BOX MPY OPCODE
0271 24201  340 000511           IMM  MPP2 LOW  L    000B      SET L = 177400B
0272 24202  010 042432               MPP1 PASS MPPB S2        SEND OP1 = (S2 S3 S4)
0273 24203  010 044432               MPP1 PASS MPPB S3
0274 24204  010 046432               MPP1 PASS MPPB S4
0275 24205  010 006432               MPP1 PASS MPPB A         SEND OP2 = (A B)
0276 24206  012 011307                    AND  S7   B         SET S7 = MAN(B)
0277 24207  010 054432               MPP1 PASS MPPB S7
0278 24210  014 011347                    SANL S8   B         SET S8 = EXPO(B)
0279 24211  010 056432               MPP1 PASS MPPB S8
0280 24212  305 003247           JSB                WAIT1     WAIT FOR BOX
0281 24213  010 021332               MPP1 PASS S7   MPPB      SAVE IN (S7 S8 S9)
0282 24214  010 021372               MPP1 PASS S8   MPPB
0283 24215  010 021432               MPP1 PASS S9   MPPB
0284 24216  341 022607           IMM       LOW  IRCM 111B      BOX FIX OPCODE
0285 24217  353 172511           IMM  MPP2 CMLO L    375B      SET L = 000002B
0286 24220  305 003247           JSB                WAIT1     WAIT FOR BOX
0287 24221  010 020172               MPP1 PASS A    MPPB      SAVE INTEGER IN A
0288 24222  150 006164           LWF  R1   PASS A    A         SET FLAG = BIT0
0289 24223  010 006162                L1   PASS A    A         SET BIT0 = 0
0290 24224  334 051402           JMP  CNDX FLAG RJS  RNXT
0291 24225  010 006747                     PASS      A         TEST A
0292 24226  327 111402           JMP  CNDX AL15      RNXT      JUMP IF A < 0
0293 24227  243 006147           ENV       ADD  A    A         SET A = A+2
0294 24230  341 042607  RNXT     IMM       LOW  IRCM 121B      BOX FLOAT OPCODE
0295 24231  010 036751               MPP2                      START BOX
0296 24232  010 006432               MPP1 PASS MPPB A         SEND OPA = A
0297 24233  305 003247           JSB                WAIT1     WAIT FOR BOX
0298 24234  340 052607           IMM       LOW  IRCM 025B      BOX SUBTRACT OPCODE
0299 24235  010 036751               MPP2                      START BOX
0300 24236  010 054432               MPP1 PASS MPPB S7        SEND OP1 = (S7 S8 S9)
0301 24237  010 056432               MPP1 PASS MPPB S8
0302 24240  370 060432           RTN  MPP1 PASS MPPB S9
0303                    *
0304                    *
```

```
0305                    ************************************************************
PAGE 0008 RTE MICRO-ASSEMBLER REV.A 760818
0307                    ************************************************************
0308                    *
0309                    * TANGENT ROUTINE
0310                    *
0311                    *
0312 24241  010 007213  TAN          COV  PASS S5   A        SAVE A,B FOR
0313 24242  010 011247               PASS S6   B           INTRT ROUTINE
0314 24243  305 007347          JSB                FOPI     X = 4X/PI, REDUCE
0315 24244  335 115242          JMP  CNDX OVFL     TANERR   REDUCE ERROR
0316 24245  010 006164               R1   PASS A    A        SET FLAG = BIT1(N)
0317 24246  150 006764          LWF  R1   PASS     A
0318 24247  305 005307          JSB                XSQ      GET X, SQUARE
0319 24250  346 177047          IMM       HIGH S2  277B     SET (S2 S3) = C4 =
0320 24251  343 146507          IMM       LOW  L   363B        -4.0030956
0321 24252  012 043047                    AND  S2  S2
0322 24253  345 045107          IMM       HIGH S3  122B
0323 24254  340 014507          IMM       LOW  L   006B
0324 24255  012 045107                    AND  S3  S3
0325 24256  306 043242          JSB  CNDX MPP  RJS WAIT1    WAIT FOR BOX
0326 24257  010 021432               MPP1 PASS S9   MPPB     SAVE IN XSQ (S9 S10)
0327 24260  010 021472               MPP1 PASS S10  MPPB
0328 24261  305 004647          JSB                FADD+1   Z = Z + C4
0329 24262  346 141047          IMM       HIGH S2  260B     SET (S2 S3) = C3 =
0330 24263  340 016507          IMM       LOW  L   007B        -1279.5424
0331 24264  012 043047                    AND  S2  S2
0332 24265  345 045107          IMM       HIGH S3  122B
0333 24266  340 054507          IMM       LOW  L   026B
0334 24267  012 045107                    AND  S3  S3
0335 24270  305 006307          JSB                FDVAC2   Z = C3/Z
0336 24271  010 061047                    PASS S2  S9       SET (S2 S3) = XSQ
0337 24272  010 063107                    PASS S3  S10
0338 24273  305 004607          JSB                FADD     Z = Z + XSQ
0339 24274  345 003047          IMM       HIGH S2  101B     SET (S2 S3) = C2 =
0340 24275  341 150507          IMM       LOW  L   164B        .0019974806
0341 24276  012 043047                    AND  S2  S2
0342 24277  354 053107          IMM       CMHI S3  025B
0343 24300  353 142507          IMM       CMLO L   361B
0344 24301  017 045107                    NOR  S3  S3
0345 24302  305 003007          JSB                FMPY     Z = C2 * Z
0346 24303  345 027047          IMM       HIGH S2  113B     SET (S2 S3) = C1 =
0347 24304  340 164507          IMM       LOW  L   072B        .146926953
0348 24305  012 043047                    AND  S2  S2
0349 24306  354 005107          IMM       CMHI S3  002B
0350 24307  353 172507          IMM       CMLO L   375B
0351 24310  017 045107                    NOR  S3  S3
0352 24311  305 004607          JSB                FADD     Z = Z + C1
0353 24312  010 055047                    PASS S2  S7       SET (S2 S3) = X
0354 24313  010 057107                    PASS S3  S8
0355 24314  305 003007          JSB                FMPY     Z = X * Z
0356 24315  334 055042          JMP  CNDX FLAG RJS EXIT1

PAGE 0009 RTE MICRO-ASSEMBLER REV.A 760818
0358 24316  355 177047          IMM       CMHI S2  177B     SET (S2 S3) = -1.0
0359 24317  006 037107                    ZERO S3
0360 24320  305 006307          JSB                FDVAC2   Z = -1/Z
0361 24321  305 003247  EXIT1   JSB                WAIT1    WAIT FOR BOX
0362 24322  227 174707          READ      INC  PNM P        START READ
0363 24323  010 020172               MPP1 PASS A    MPPB     PUT ANSWER IN (A B)
0364 24324  370 020232          RTN  MPP1 PASS B    MPPB     RETURN
0365                    *
0366                    *
0367 24325  340 162514  TANERR  IMM  SOV  LOW  L   071B     SET (A B) TO
0368 24326  344 140147          IMM       HIGH A   060B        ASCII 09OR
0369 24327  012 006147                    AND  A   A
0370 24330  341 044507          IMM       LOW  L   122B
0371 24331  345 036207          IMM       HIGH B   117B
0372 24332  000 075732  ERRET        MPP1 DEC  P   P        SET P = ERROR ADDR
0373 24333  227 174707          READ      INC  PNM P        START READ
0374 24334  372 010207          RTN       AND  B   B        ERROR RETURN
0375                    *
0376                    *
```

```
0377                    ************************************************************
PAGE 0010 RTE MICRO-ASSEMBLER REV.A 760818
0379                    ************************************************************
0380                    *
0381                    *
0382                    *     SINE ROUTINE
0383                    *
0384                    *
0385 24335  353 173007  COS      IMM        CMLO S1   375B      SET J=2
0386 24336  325 016007           JMP                  SIN+1
0387                    *
0388                    *
0389 24337  006 037007  SIN                 ZERO S1              SET J=0
0390 24340  010 007213                 COV  PASS S5   A         SAVE A,B FOR
0391 24341  010 011247                      PASS S6   B            INTRT ROUTINE
0392 24342  305 007347           JSB                  FOPI      X = 4X/PI, REDUCE
0393 24343  335 122542           JMP  CNDX  OVFL      SINERR    REDUCE ERROR
0394 24344  010 040507                      PASS L    S1        SET L=J
0395 24345  003 006164                 R1   ADD  A    A         N = (N+J)/2
0396 24346  150 007024           LWF  R1    PASS S1   A         SET FLAG = BIT1(N)
0397 24347  305 005307           JSB                  XSQ       GET X, SQUARE
0398 24350  334 060242           JMP  CNDX  FLAG RJS  SINAG     SIN OR COS ?
0399                    *
0400 24351  343 127107  COSAG    IMM        LOW  S3   353B      SET (S2 S3) = CC4 =
0401 24352  344 012507           IMM        HIGH L    005B         -.00031957
0402 24353  012 045107                      AND  S3   S3
0403 24354  346 131047           IMM        HIGH S2   254B
0404 24355  340 164507           IMM        LOW  L    072B
0405 24356  012 043047                      AND  S2   S2
0406 24357  306 043242           JSB  CNDX  MPP  RJS  WAIT1     WAIT FOR BOX
0407 24360  010 021432                 MPP1 PASS S9   MPPB      SAVE VSQR IN (S9 S10)
0408 24361  010 021472                 MPP1 PASS S10  MPPB
0409 24362  305 003047           JSB                  FMPY+1    Z = Z*CC4
0410 24363  345 000507           IMM        HIGH L    100B      SET (S2 S3) = CC3 =
0411 24364  343 133047           IMM        LOW  S2   355B         .015851077
0412 24365  012 043047                      AND  S2   S2
0413 24366  354 036507           IMM        CMHI L    017B
0414 24367  353 157107           IMM        CMLO S3   367B
0415 24370  017 045107                      NOR  S3   S3
0416 24371  305 004607           JSB                  FADD      Z = Z+CC3
0417 24372  305 006547           JSB                  VMPY      Z = Z*VSQR
0418 24373  356 142507           IMM        CMHI L    261B      SET (S2 S3) = CC2 =
0419 24374  350 027047           IMM        CMLO S2   013B         -.30842483
0420 24375  017 043047                      NOR  S2   S2
0421 24376  344 045107           IMM        HIGH S3   022B
0422 24377  305 004607           JSB                  FADD      Z = Z+CC2
0423 24400  305 006547           JSB                  VMPY      Z = Z*VSQR
0424 24401  356 177047           IMM        CMHI S2   277B      SET (S2 S3) = CC1 =
0425 24402  353 173107           IMM        CMLO S3   375B         1.0
0426 24403  305 004607           JSB                  FADD      Z = Z+CC1
0427 24404  325 022347           JMP                  CKSGN     TEST SIGN OF ANSWER
```

```
PAGE 0011 RTE MICRO-ASSEMBLER REV.A 760818
0429 24405  342 067047  SINAG    IMM        LOW  S2   233B      SET (S2 S3) = C4 =
0430 24406  346 150507           IMM        HIGH L    264B         -.000035950439
0431 24407  012 043047                      AND  S2   S2
0432 24410  355 002507           IMM        CMHI L    101B
0433 24411  353 113107           IMM        CMLO S3   345B
0434 24412  017 045107                      NOR  S3   S3
0435 24413  306 043242           JSB  CNDX MPP  RJS   WAIT1     WAIT FOR BOX
0436 24414  010 021432                MPP1 PASS S9    MPPB      SAVE VSQR IN (S9 S10)
0437 24415  010 021472                MPP1 PASS S10   MPPB
0438 24416  305 003047           JSB                  FMPY+1    Z = VSQR*C4
0439 24417  345 042507           IMM        HIGH L    121B      SET (S2 S3) = C3 =
0440 24420  342 057047           IMM        LOW  S2   227B         .002490001
0441 24421  012 043047                      AND  S2   S2
0442 24422  356 110507           IMM        CMHI L    244B
0443 24423  353 143107           IMM        CMLO S3   361B
0444 24424  017 045107                      NOR  S3   S3
0445 24425  305 004607           JSB                  FADD      Z = Z+C3
0446 24426  305 006547           JSB                  VMPY      Z = Z*VSQR
0447 24427  346 132507           IMM        HIGH L    255B      SET (S2 S3) = C2 =
0448 24430  341 043047           IMM        LOW  S2   121B         -.0807454325
0449 24431  012 043047                      AND  S2   S2
0450 24432  354 044507           IMM        CMHI L    022B
0451 24433  353 167107           IMM        CMLO S3   373B
0452 24434  017 045107                      NOR  S3   S3
0453 24435  305 004607           JSB                  FADD      Z = Z+C2
0454 24436  305 006547           JSB                  VMPY      Z = Z*VSQR
0455 24437  355 110507           IMM        CMHI L    144B      SET (S2 S3) = C1 =
0456 24440  352 017047           IMM        CMLO S2   207B         .78539816
0457 24441  017 043047                      NOR  S2   S2
0458 24442  354 045107           IMM        CMHI S3   022B
0459 24443  305 004607           JSB                  FADD      Z = Z+C1
0460 24444  010 055047                      PASS S2   S7        SET (S2 S3) = V
0461 24445  010 057107                      PASS S3   S8
0462 24446  305 003007           JSB                  FMPY      Z = Z * V
0463                     *
0464 24447  010 040747  CKSGN               PASS      S1        TEST SIGN
0465 24450  321 155042           JMP  CNDX AL0  RJS   EXIT1     EXIT IF BIT2(N) = 1
0466 24451  305 007007           JSB                  NEGATE    Z = -Z
0467 24452  325 015047           JMP                  EXIT1     EXIT ROUTINE
0468                     *
0469                     *
0470 24453  344 140514  SINERR   IMM  SOV  HIGH L     060B      SET (A B) TO
0471 24454  340 152147           IMM       LOW  A     065B         ASCII 050R
0472 24455  012 006147                     AND  A     A
0473 24456  345 036507           IMM       HIGH L     117B
0474 24457  341 044207           IMM       LOW  B     122B
0475 24460  325 015507           JMP                  ERRET     ERROR RETURN
0476                     *
0477                     *
```

```
0478                        ************************************************************
PAGE 0012 RTE MICRO-ASSEMBLER REV.A 760818
0480                        ************************************************************
0481                        *
0482                        *     NATURAL LOGARITHM ROUTINE
0483                        *
0484 24461  010 011253  ALOG          COV  PASS S6   B          SAVE A,B FOR
0485 24462  010 007207                     PASS S5   A              INTRT ROUTINE
0486 24463  320 026702          JMP  CNDX ALZ       LOGERR    ERROR IF X = 0
0487 24464  327 126702          JMP  CNDX AL15      LOGERR    ERROR IF X < 0
0488 24465  305 001007          JSB                 FLUN      UNPACK MANT AND EXP
0489 24466  357 062507          IMM       CMHI L    331B      TEST FOR BITS 14-7
0490 24467  003 050747                    ADD       S5           OF X > 264B
0491 24470  327 123602          JMP  CNDX AL15      LGNXT     JUMP IF GREATER
0492 24471  000 006147                    DEC  A    A         DECREMENT EXPO(X)
0493 24472  007 110207                    INC  B    B         SET EXPO(MAN(X)) = 1
0494 24473  007 110207                    INC  B    B
0495 24474  341 040607  LGNXT   IMM       LOW  IRCM 120B      BOX FLOAT OPCODE
0496 24475  356 177051          IMM  MPP2 CMHI S2   277B      SET (S2 S3) = 1.0
0497 24476  010 006432               MPP1 PASS MPPB A         SEND OP1 = EXPO(X)
0498 24477  010 050147                    PASS A    S5        SET (A B) = MAN(X)
0499 24500  353 173107          IMM       CMLO S3   375B
0500 24501  306 043242          JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0501 24502  010 021032               MPP1 PASS S1   MPPB      SAVE CHAR IN (S1 S11)
0502 24503  010 021532               MPP1 PASS S11  MPPB
0503 24504  340 040607          IMM       LOW  IRCM 020B      BOX SUB OPCODE
0504 24505  305 006047          JSB                 BAB23     Z = MAN(X) - 1.0
0505 24506  305 003247          JSB                 WAIT1     WAIT FOR BOX
0506 24507  010 021332               MPP1 PASS S7   MPPB      SAVE IN Y (S7 S8)
0507 24510  010 021372               MPP1 PASS S8   MPPB
0508 24511  305 006007          JSB                 FADA2+1   Z = MAN(X) + 1.0
0509 24512  305 005047          JSB                 FDIV7     Z = Z/Y
0510 24513  305 003247          JSB                 WAIT1     WAIT FOR BOX
0511 24514  010 021432               MPP1 PASS S9   MPPB      SAVE IN W (S9 S10)
0512 24515  010 021472               MPP1 PASS S10  MPPB
0513 24516  340 130607          IMM       LOW  IRCM 054B      BOX Z*Z OPCODE
0514 24517  345 125051          IMM  MPP2 HIGH S2   152B      START BOX
0515 24520  340 020507          IMM       LOW  L    010B      SET (S2 S3) = C =
0516 24521  012 043047                    AND  S2   S2           1.6567626301
0517 24522  345 115107          IMM       HIGH S3   146B
0518 24523  340 004507          IMM       LOW  L    002B
0519 24524  012 045107                    AND  S3   S3
0520 24525  305 004347          JSB                 FSUB2     Z = Z-C
0521 24526  346 126507          IMM       HIGH L    253B      SET (S2 S3) = MB =
0522 24527  342 015047          IMM       LOW  S2   206B         -2.6398577035
0523 24530  012 043047                    AND  S2   S2
0524 24531  345 023107          IMM       HIGH S3   111B
0525 24532  340 010507          IMM       LOW  L    004B
0526 24533  012 045107                    AND  S3   S3
0527 24534  305 006307          JSB                 FDVAC2    Z = MB/Z
0528 24535  345 044507          IMM       HIGH L    122B      SET (S2 S3) = A =
0529 24536  342 141047          IMM       LOW  S2   260B         1.2920070987
0530 24537  012 043047                    AND  S2   S2
0531 24540  344 177107          IMM       HIGH S3   077B
0532 24541  340 004507          IMM       LOW  L    002B
0533 24542  012 045107                    AND  S3   S3
```

```
PAGE 0013 RTE MICRO-ASSEMBLER REV.A 760818
0535 24543  305 004607          JSB                 FADD      Z = Z+A
0536 24544  305 006547          JSB                 VMPY      Z = Z*W
0537 24545  010 041047                   PASS S2    S1        SET (S2 S3) = CHAR
0538 24546  010 065107                   PASS S3    S11
0539 24547  305 004607          JSB                 FADD      Z = Z+CHAR
0540 24550  355 060507          IMM      CMHI L     130B      SET (S2 S3) = LE2 =
0541 24551  352 163047          IMM      CMLO S2    271B         .6931471806
0542 24552  017 043047                   NOR  S2    S2
0543 24553  357 147116          IMM  CLFL CMHI S3   363B
0544 24554  305 003007          JSB                 FMPY      Z = Z*LE2
0545 24555  325 015047          JMP                 EXIT1     EXIT ROUTINE
0546                     *
0547                     *
0548 24556  344 140514  LOGERR  IMM  SOV  HIGH L     060B      SET (A B) TO
0549 24557  340 144157          IMM  STFL LOW  A     062B         ASCII 02UN
0550 24560  012 006147                   AND  A     A
0551 24561  345 052507          IMM      HIGH L     125B
0552 24562  341 034207          IMM      LOW  B     116B
0553 24563  325 015507          JMP                 ERRET     ERROR RETURN
0554                     *
0555                     *
0556                     ************************************************************
0557                     *
0558                     *
0559                     *  COMMON LOG ROUTINE
0560                     *
0561 24564  305 023047  ALOGT   JSB                 ALOG      COMPUTE LN(X)
0562 24565  374 023042          RTN  CNDX FLAG                ERROR RETURN
0563                     *
0564 24566  000 075707                   DEC  P     P         SET RETURN ADDRESS
0565 24567  355 136507          IMM      CMHI L     157B      SET (S2 S3) = LOG(E) =
0566 24570  350 133047          IMM      CMLO S2    055B         .43429228
0567 24571  017 043047                   NOR  S2    S2
0568 24572  347 131107          IMM      HIGH S3    354B
0569 24573  305 003047          JSB                 FMPY+1
0570 24574  325 015047          JMP                 EXIT1     EXIT ROUTINE
0571                     *
0572                     *
0573                     ************************************************************
```

```
PAGE 0014 RTE MICRO-ASSEMBLER REV.A 760818
0575                        ***********************************************************
0576                        *
0577                        *      ARCTANGENT ROUTINE
0578                        *
0579                        *
0580 24575  340 000607  ATAN     IMM        LOW  IRCM 000B      BOX ADD OPCODE
0581 24576  150 010764           LWF  R1    PASS      B         SET FLAG = BIT0(B)
0582 24577  010 011253                COV   PASS S6   B         SAVE A,B FOR
0583 24600  010 007207                      PASS S5   A            INTRT ROUTINE
0584 24601  320 035202           JMP  CNDX ALZ       ZERO1     RETURN ZERO IF X = 0
0585 24602  334 030242           JMP  CNDX FLAG      POS       JUMP IF ABS(X) < .5
0586 24603  327 170242           JMP  CNDX AL15 RJS  POS       JUMP IF X > 0
0587 24604  340 040607           IMM        LOW  IRCM 020B      BOX SUBTRACT OPCODE
0588 24605  006 037051  POS           MPP2 ZERO S2             SET S2 = 0
0589 24606  010 042432                MPP1 PASS MPPB S2        SEND OP1 = 0
0590 24607  010 042432                MPP1 PASS MPPB S2
0591 24610  010 006432                MPP1 PASS MPPB A         SEND OP2 = (A B)
0592 24611  010 010432                MPP1 PASS MPPB B
0593 24612  334 032042           JMP  CNDX FLAG      ATAN3     JUMP IF ABS(X) < .5
0594 24613  345 111007           IMM        HIGH S1   144B      SET (S1 S11) = PI/4
0595 24614  342 016507           IMM        LOW  L    207B
0596 24615  012 041007                      AND  S1   S1
0597 24616  354 045507           IMM        CMHI S11  022B
0598 24617  356 177047           IMM        CMHI S2   277B      SET (S2 S3) = 1.0
0599 24620  353 173107           IMM        CMLO S3   375B
0600 24621  350 006507           IMM        CMLO L    003B      SET L = 000374B
0601 24622  012 010747                      AND       B         TEST EXPO(X)
0602 24623  320 031402           JMP  CNDX ALZ       ATAN1     JUMP IF ABS(X) < 2
0603 24624  007 165507                      INC  S11  S11       SET (S1 S11) = PI/2
0604 24625  007 165507                      INC  S11  S11
0605 24626  305 006307           JSB                  FDVAC2    X = 1/X
0606 24627  325 032047           JMP                  ATAN3     CONTINUE
0607 24630  306 043242  ATAN1    JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0608 24631  010 020172                MPP1 PASS A    MPPB      SAVE X IN (A B)
0609 24632  010 020232                MPP1 PASS B    MPPB
0610 24633  305 007147           JSB                  FSUB+1    Z = 1 - X
0611 24634  305 003247           JSB                  WAIT1     WAIT FOR BOX
0612 24635  010 021332                MPP1 PASS S7   MPPB      SAVE (1-X) IN (S7 S8)
0613 24636  010 021372                MPP1 PASS S8   MPPB
0614 24637  305 006007           JSB                  FADA2+1   Z = 1 + X
0615 24640  305 005047           JSB                  FDIV7     Z = (1-X)/(1+X)
0616 24641  305 003247  ATAN3    JSB                  WAIT1     WAIT FOR BOX
0617 24642  010 021332                MPP1 PASS S7   MPPB      SAVE NEW X IN (S7 S8)
0618 24643  010 021372                MPP1 PASS S8   MPPB
0619 24644  340 130613           IMM  COV  LOW  IRCM 054B      BOX Z*Z OPCODE
0620 24645  345 001051           IMM  MPP2 HIGH S2   100B      SET (S2 S3) = C4 =
0621 24646  342 136507           IMM        LOW  L    257B         2.0214656
0622 24647  012 043047                      AND  S2   S2
0623 24650  347 063107           IMM        HIGH S3   331B
0624 24651  340 010507           IMM        LOW  L    004B
0625 24652  012 045107                      AND  S3   S3
```

```
PAGE 0015 RTE MICRO-ASSEMBLER REV.A 760818
0627 24653  306 043242          JSB  CNDX MPP  RJS  WAIT1     WAIT FOR BOX
0628 24654  010 021432               MPP1 PASS S9   MPPB      SAVE XSQ IN (S9 S10)
0629 24655  010 021472               MPP1 PASS S10  MPPB
0630 24656  305 004647          JSB                 FADD+1    Z = Z + C4
0631 24657  346 151047          IMM       HIGH S2   264B      SET (S2 S3) = C3 =
0632 24660  340 144507          IMM       LOW  L    062B         -4.7376165
0633 24661  012 043047                    AND  S2   S2
0634 24662  346 157107          IMM       HIGH S3   267B
0635 24663  340 014507          IMM       LOW  L    006B
0636 24664  012 045107                    AND  S3   S3
0637 24665  305 006307          JSB                 FDVAC2    Z = C3/Z
0638 24666  010 061047                    PASS S2   S9        SET (S2 S3) = XSQ
0639 24667  010 063107                    PASS S3   S10
0640 24670  305 004607          JSB                 FADD      Z = Z + XSQ
0641 24671  345 037047          IMM       HIGH S2   117B      SET (S2 S3) = C2 =
0642 24672  340 016507          IMM       LOW  L    007B         .154357652
0643 24673  012 043047                    AND  S2   S2
0644 24674  357 157107          IMM       CMHI S3   367B
0645 24675  353 172507          IMM       CMLO L    375B
0646 24676  017 045107                    NOR  S3   S3
0647 24677  305 003007          JSB                 FMPY      Z = C2 * Z
0648 24700  345 057047          IMM       HIGH S2   127B      SET (S2 S3) = C1 =
0649 24701  340 116507          IMM       LOW  L    047B         1.3617611
0650 24702  012 043047                    AND  S2   S2
0651 24703  344 061107          IMM       HIGH S3   030B
0652 24704  340 004507          IMM       LOW  L    002B
0653 24705  012 045107                    AND  S3   S3
0654 24706  305 004607          JSB                 FADD      Z = C1 + Z
0655 24707  305 005047          JSB                 FDIV7     Z = X/Z
0656 24710  334 034742          JMP  CNDX FLAG      EXIT2     EXIT IF ABS(X) < .5
0657 24711  010 041047                    PASS S2   S1        SET (S2 S3) = PI/N
0658 24712  010 065107                    PASS S3   S11
0659 24713  305 004347          JSB                 FSUB2     Z = Z - PI/N
0660 24714  010 050747                    PASS      S5        TEST MAN(X)
0661 24715  327 134742          JMP  CNDX AL15      EXIT2     EXIT IF X < 0
0662 24716  305 007007          JSB                 NEGATE    Z = - Z
0663                      *
0664 24717  305 003247  EXIT2   JSB                 WAIT1     WAIT FOR BOX
0665 24720  000 075707                    DEC  P    P         SET RETURN ADDR
0666 24721  227 174707          READ      INC  PNM  P         START READ
0667 24722  010 020172               MPP1 PASS A    MPPB      SAVE ANS IN (A B)
0668 24723  370 020232          RTN  MPP1 PASS B    MPPB      RETURN
0669                      *
0670                      *
0671 24724  000 075707  ZERO1             DEC  P    P         SET RETURN ADDR
0672 24725  227 174707          READ      INC  PNM  P         START READ
0673 24726  370 036747          RTN                           RETURN
0674                      *
0675                      *
0676                      ************************************************************
PAGE 0016 RTE MICRO-ASSEMBLER REV.A 760818
0678                      ************************************************************
0679                      *
0680                      *
0681                      *  SELF-TEST ROUTINE
0682                      *
0683                      *
0684 24727  376 000742  SELFTEST RTN  CNDX NSNG                RETURN IF NOT SNGL STEP
0685 24730  343 076347          IMM       LOW  DSPI 337B      TURN ON "S" DSPI LED
0686 24731  355 167747          IMM       CMHI S    173B      SET S = 102001B
0687 24732  007 177747                    INC  S    S
0688 24733  010 076307                    PASS DSPL S         SEND TO PANEL
0689 24734  356 176147          IMM       CMHI A    277B      SET (A B) = 4.0
0690 24735  353 162207          IMM       CMLO B    371B
0691 24736  305 041007          JSB                 SQRT      CALCULATE SQRT(4)
0692 24737  007 177747                    INC  S    S         SET S = 102002B
0693 24740  335 136442          JMP  CNDX OVFL      DISPLAY   JUMP IF OVERFLOW
0694 24741  356 176507          IMM       CMHI L    277B      CHECK ANSWER
0695 24742  014 106747                    XOR       A
0696 24743  320 076442          JMP  CNDX ALZ  RJS  DISPLAY   JUMP IF A WRONG
0697 24744  353 166507          IMM       CMLO L    373B
0698 24745  014 110747                    XOR       B
0699 24746  320 076442          JMP  CNDX ALZ  RJS  DISPLAY   JUMP IF B WRONG
0700 24747  353 000507          IMM       CMLO L    300B
0701 24750  010 177747                    IOR  S    S         SET S = 102077B
0702 24751  370 076307  DISPLAY RTN       PASS DSPL S         SEND TO PANEL, RETURN
0703                      *
0704                      *
```

```
0705                    ************************************************************
PAGE 0017 RTE MICRO-ASSEMBLER REV.A 760818
0707                    ************************************************************
0708                    *
0709                    *
0710                    *        RESERVED ENTRY POINTS
0711                    *
0712                    *
0713                             ORG                 25000B
0714 25000  230 036740           READ RTN
0715 25001  230 036740           READ RTN
0716 25002  230 036740           READ RTN
0717 25003  230 036740           READ RTN
0718 25004  230 036740           READ RTN
0719 25005  230 036740           READ RTN
0720 25006  230 036740           READ RTN
0721 25007  230 036740           READ RTN
0722 25010  230 036740           READ RTN
0723 25011  230 036740           READ RTN
0724 25012  230 036740           READ RTN
0725 25013  230 036740           READ RTN
0726 25014  230 036740           READ RTN
0727 25015  230 036740           READ RTN
0728 25016  230 036740           READ RTN
0729 25017  230 036740           READ RTN
0730                    *
0731                    *
0732                    ************************************************************
PAGE 0018 RTE MICRO-ASSEMBLER REV.A 760818
0734                    ************************************************************
0735                    *
0736                    *     SQUARE ROOT ROUTINE
0737                    *
0738 25020  010 011253  SQRT          COV  PASS S6   B          SAVE A,B FOR
0739 25021  010 007207                     PASS S5   A             INTRT ROUTINE
0740 25022  320 013442          JMP  CNDX AL2        EXIT3      EXIT IF X = 0
0741 25023  327 104142          JMP  CNDX AL15       SQRERR     ERROR IF X < 0
0742 25024  305 001007          JSB                  FLUN       UNPACK EXP AND MAN
0743 25025  150 006762          LWF  L1   PASS       A          ARITHMETIC SHIFT
0744 25026  150 007524          LWF  R1   PASS S11   A             A-REG RIGHT
0745 25027  321 102202          JMP  CNDX AL0        ODD        JUMP IF EXP ODD
0746                    *
0747 25030  010 050147  EVEN               PASS A    S5         RESTORE A
0748 25031  345 027047          IMM        HIGH S2   113B       SET (S2 S3) = A2 =
0749 25032  342 024507          IMM        LOW  L    212B          .5901621
0750 25033  012 043047                     AND  S2   S2
0751 25034  356 041107          IMM        CMHI S3   220B
0752 25035  340 100607          IMM        LOW  IRCM 040B       BOX MPY OPCODE
0753 25036  305 006047          JSB                  BAB23      Z = F*A2
0754 25037  345 125047          IMM        HIGH S2   152B       SET (S2 S3) = B2 =
0755 25040  343 050507          IMM        LOW  L    324B          .4173076
0756 25041  012 043047                     AND  S2   S2
0757 25042  346 131107          IMM        HIGH S3   254B
0758 25043  325 043047          JMP                  BOTH       CONTINUE
0759                    *
0760 25044  010 050147  ODD                PASS A    S5         RESTORE A
0761 25045  345 125047          IMM        HIGH S2   152B       SET (S2 S3) = A1 =
0762 25046  343 050507          IMM        LOW  L    324B          .8346152
0763 25047  012 043047                     AND  S2   S2
0764 25050  355 045107          IMM        CMHI S3   122B
0765 25051  340 100607          IMM        LOW  IRCM 040B       BOX MPY OPCODE
0766 25052  305 006047          JSB                  BAB23      Z = F*A1
0767 25053  345 027047          IMM        HIGH S2   113B       SET (S2 S3) = B1 =
0768 25054  342 024507          IMM        LOW  L    212B          .5901621
0769 25055  012 043047                     AND  S2   S2
0770 25056  356 041107          IMM        CMHI S3   220B
0771 25057  007 110207                     INC  B    B          SET F = 2 * F
0772 25060  007 110207                     INC  B    B
0773                    *
0774 25061  305 004607  BOTH    JSB                  FADD       Z = Z+B
0775 25062  010 007307                     PASS S7   A          SET (S7 S8) = F =
0776 25063  010 011347                     PASS S8   B             MAN(X)
0777 25064  305 003247          JSB                  WAIT1      WAIT FOR BOX
0778 25065  010 021072               MPP1 PASS S2   MPPB       SAVE IN P (S2 S3)
0779 25066  010 021132               MPP1 PASS S3   MPPB
0780 25067  305 005107          JSB                  FDIV7+1    Z = F/P
0781 25070  353 172507          IMM        CMLO L    375B       SET L = 2
0782 25071  003 057347                     ADD  S8   S8         SET F = 4 * F
0783 25072  003 057347                     ADD  S8   S8
```

```
PAGE 0019 RTE MICRO-ASSEMBLER REV.A 760818
0785 25073  305 004607          JSB                 FADD      Z = Z+P
0786 25074  305 003247          JSB                 WAIT1     WAIT FOR BOX
0787 25075  010 021072               MPP1 PASS S2   MPPB      SAVE IN P (S2 S3)
0788 25076  010 021132               MPP1 PASS S3   MPPB
0789 25077  305 005107          JSB                 FDIV7+1   Z = F/P
0790 25100  305 004607          JSB                 FADD      Z = Z+P
0791 25101  004 165507                   SUB  S11   S11       DEC EXPONENT BY 2
0792 25102  325 001307          JMP                 PWR2      Z = Z*2**S11, RETURN
0793                     *
0794                     *
0795 25103  344 140514  SQRERR  IMM  SOV  HIGH L    060B      SET (A B) TO
0796 25104  340 146147          IMM       LOW  A    063B         ASCII 03UN
0797 25105  012 006147                    AND  A    A
0798 25106  345 052507          IMM       HIGH L    125B
0799 25107  341 034207          IMM       LOW  B    116B
0800 25110  325 015507          JMP                 ERRET     ERROR RETURN
0801                     *
0802                     *
0803                     ************************************************************
PAGE 0020 RTE MICRO-ASSEMBLER REV.A 760818
0805                     ************************************************************
0806                     *
0807                     *     EXPONENTIATION ROUTINE
0808                     *
0809 25111  010 007213  EXP           COV  PASS S5   A         SAVE A,B
0810 25112  010 011247                     PASS S6   B            FOR INTRT ROUTINE
0811 25113  345 070507          IMM       HIGH L    134B      SET (S2 S3 S4) =
0812 25114  341 053047          IMM       LOW  S2   125B         2/LE2
0813 25115  012 043047                    AND  S2   S2
0814 25116  344 073107          IMM       HIGH S3   035B
0815 25117  342 050507          IMM       LOW  L    224B
0816 25120  012 045107                    AND  S3   S3
0817 25121  346 135147          IMM       HIGH S4   256B
0818 25122  340 010507          IMM       LOW  L    004B
0819 25123  012 047147                    AND  S4   S4
0820 25124  305 010007          JSB                 REDUCE
0821 25125  150 006762          LWF  L1   PASS      A         ARITH RT SHIFT
0822 25126  150 006164          LWF  R1   PASS A    A
0823 25127  007 107507                    INC  S11  A         INC, SAVE IN S11
0824                     *
0825                     *  PERFORM BOUNDS CHECKS
0826                     *
0827 25130  010 050747                    PASS      S5        TEST X
0828 25131  327 145702          JMP  CNDX AL15 RJS  POSCHK    JUMP IF X > 0
0829                     *
0830 25132  351 176507  NEGCHK  IMM       CMLO L    177B      SET L = 128
0831 25133  003 064747                    ADD       S11       INT < -128 ?
0832 25134  327 107502          JMP  CNDX AL15      ZERO2     YES, ANS = 0
0833 25135  325 046107          JMP                 EXPNXT    NO, CONTINUE
0834                     *
0835 25136  335 107642  POSCHK  JMP  CNDX OVFL      EXPERR    REDUCE ERROR
0836 25137  342 000507          IMM       LOW  L    200B      SET L = -128
0837 25140  003 006747                    ADD       A         INT > 128?
0838 25141  327 147642          JMP  CNDX AL15 RJS  EXPERR    YES, ERROR
0839                     *
0840 25142  305 005307  EXPNXT  JSB                 XSQ       GET X, SQUARE
0841 25143  356 023047          IMM       CMHI S2   211B      SET (S2 S3) = C2 =
0842 25144  355 143107          IMM       CMHI S3   161B         .05761803
0843 25145  353 162507          IMM       CMLO L    371B
0844 25146  017 045107                    NOR  S3   S3
0845 25147  305 003007          JSB                 FMPY      Z = C2 * Z
0846 25150  010 055047                    PASS S2   S7        SET (S2 S3) = F
0847 25151  010 057107                    PASS S3   S8
0848 25152  305 004347          JSB                 FSUB2     Z = Z - F
0849 25153  345 071047          IMM       HIGH S2   134B      SET (S2 S3) = C1 =
0850 25154  341 052507          IMM       LOW  L    125B         5.7708162
0851 25155  012 043047                    AND  S2   S2
0852 25156  345 007107          IMM       HIGH S3   103B
0853 25157  340 014507          IMM       LOW  L    006B
0854 25160  012 045107                    AND  S3   S3
0855 25161  305 004607          JSB                 FADD      Z = C1 + Z
```

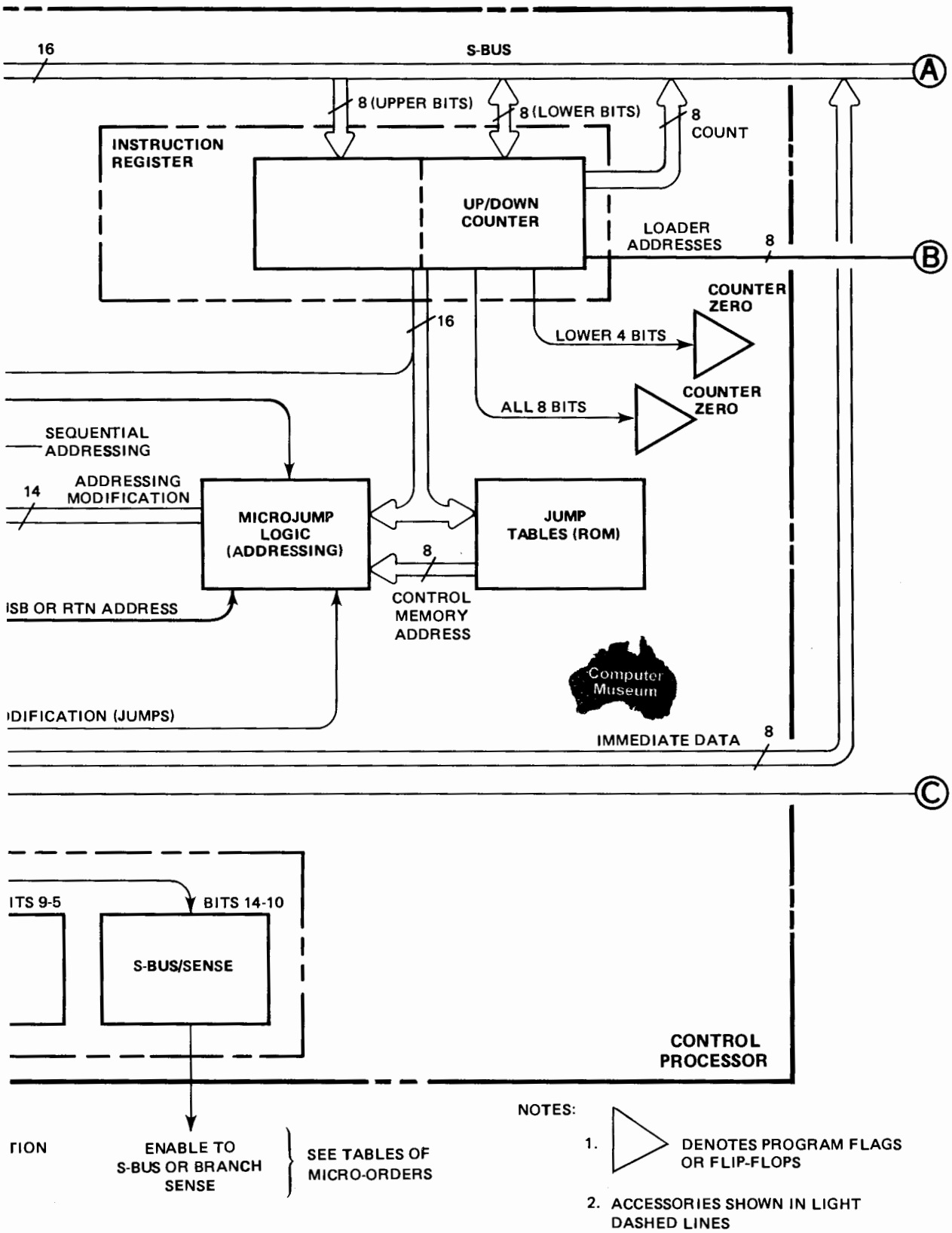
```
PAGE 0021 RTE MICRO-ASSEMBLER REV.A 760818
0857 25162  305 005047          JSB                     FDIV7     Z = F/Z
0858 25163  356 177047          IMM       CMHI S2       277B      SET (S2 S3) = .5
0859 25164  006 037107                    ZERO S3
0860 25165  305 004607          JSB                     FADD      Z = Z + .5
0861 25166  305 001307          JSB                     PWR2      Z = Z*2**S11
0862 25167  375 141302          RTN  CNDX OVFL RJS                RETURN IF PWR2 OK
0863 25170  010 050747                    PASS          S5        TEST X
0864 25171  327 147642          JMP  CNDX AL15 RJS      EXPERR    ERROR IF X > 0
0865                     *
0866                     *
0867 25172  227 174732  ZERO2   READ MPP1 INC  PNM       P         START READ
0868 25173  006 036153               COV  ZERO A                  SET ANS = 0
0869 25174  366 036207          RTN       ZERO B                  RETURN
0870                     *
0871                     *
0872 25175  344 140514  EXPERR  IMM  SOV  HIGH L        060B      SET (A B) TO
0873 25176  340 156147          IMM       LOW  A        067B         ASCII 07OF
0874 25177  012 006147                    AND  A        A
0875 25200  345 036507          IMM       HIGH L        117B
0876 25201  341 014207          IMM       LOW  B        106B
0877 25202  325 015507          JMP                     ERRET     ERROR RETURN
0878                     *
0879                     *

0880                     ************************************************************
PAGE 0022 RTE MICRO-ASSEMBLER REV.A 760818
0882                     ************************************************************
0883                     *
0884                     *  TANH ROUTINE
0885                     *
0886 25203  010 007213  TANH          COV  PASS S5        A         SAVE A,B FOR
0887 25204  010 011247                    PASS S6        B            INTRT ROUTINE
0888                     *
0889                     *  PERFORM BOUNDS CHECKS
0890                     *
0891 25205  321 111342          JMP  CNDX AL0           TANH1     JUMP IF ABS(X) < .5
0892 25206  350 016507          IMM       CMLO L        007B      SET L = 000370B
0893 25207  012 010747                    AND           B
0894 25210  320 053202          JMP  CNDX ALZ  RJS      TANH2     JUMP IF ABS(X) > 8
0895                     *
0896 25211  007 110207                    INC  B        B         SET X = 2*X
0897 25212  007 110207                    INC  B        B
0898 25213  305 044547          JSB                     EXP+2     GET EXP(2*X)
0899 25214  000 075707                    DEC  P        P         SET RETURN ADDRESS
0900 25215  356 177047          IMM       CMHI S2       277B      SET (S2 S3) = 1.0
0901 25216  353 173107          IMM       CMLO S3       375B
0902 25217  340 040607          IMM       LOW  IRCM     020B      BOX SUB OPCODE
0903 25220  305 006047          JSB                     BAB23     Z = EXP(2*X) - 1
0904 25221  305 003247          JSB                     WAIT1     WAIT FOR BOX
0905 25222  010 021332               MPP1 PASS S7       MPPB      SAVE IN (S7 S8)
0906 25223  010 021372               MPP1 PASS S8       MPPB
0907 25224  305 006007          JSB                     FADA2+1   Z = EXP(2*X) + 1
0908 25225  305 005047          JSB                     FDIV7     Z = TANH(X)
0909 25226  325 034747          JMP                     EXIT2     EXIT ROUTINE
0910                     *
0911 25227  340 100607  TANH1   IMM       LOW  IRCM     040B      BOX MPY OPCODE
0912 25230  010 007047                    PASS S2       A         SET (S2 S3) = X
0913 25231  010 011107                    PASS S3       B
0914 25232  305 006047          JSB                     BAB23     Z = X * X
0915 25233  345 041047          IMM       HIGH S2       120B      SET (S2 S3) = C3 =
0916 25234  340 112507          IMM       LOW  L        045B         2.5045337
0917 25235  012 043047                    AND  S2       S2
0918 25236  344 111107          IMM       HIGH S3       044B
0919 25237  340 010507          IMM       LOW  L        004B
0920 25240  012 045107                    AND  S3       S3
0921 25241  305 004607          JSB                     FADD      Z = Z + C3
0922 25242  345 005307          IMM       HIGH S7       102B      SET (S7 S8) = C2 =
0923 25243  343 116507          IMM       LOW  L        347B         2.0907609
0924 25244  012 055307                    AND  S7       S7
0925 25245  346 007347          IMM       HIGH S8       203B
0926 25246  340 010507          IMM       LOW  L        004B
0927 25247  012 057347                    AND  S8       S8
0928 25250  305 005047          JSB                     FDIV7     Z = C2 / Z
```

```
PAGE 0023 RTE MICRO-ASSEMBLER REV.A 760818
0930 25251  345 051047            IMM       HIGH S2   124B      SET (S2 S3) = C1 =
0931 25252  342 054507            IMM       LOW  L    226B
0932 25253  012 043047                      AND  S2   S2
0933 25254  355 035107            IMM       CMHI S3   116B
0934 25255  353 172507            IMM       CMLO L    375B
0935 25256  017 045107                      NOR  S3   S3
0936 25257  305 004607            JSB                 FADD      Z = C1 + Z
0937 25260  010 007047                      PASS S2   A         SET (S2 S3) = X
0938 25261  010 011107                      PASS S3   B
0939 25262  305 003007            JSB                 FMPY      Z = Z * X
0940 25263  325 034747            JMP                 EXIT2     EXIT ROUTINE
0941                      *
0942                      *
0943 25264  000 075707  TANH2               DEC  P    P         SET RETURN ADDRESS
0944 25265  010 006747                      PASS      A         TEST X
0945 25266  327 113542            JMP  CNDX AL15      NEG       JUMP IF X < 0
0946 25267  356 176147            IMM       CMHI A    277B      SET (A B) = 1.0
0947 25270  353 172207            IMM       CMLO B    375B
0948 25271  227 174707  EXIT3     READ      INC  PNM  P         START READ
0949 25272  370 036747            RTN                           RETURN
0950                      *
0951                      *
0952 25273  355 176147  NEG       IMM       CMHI A    177B      SET (A B) = -1.0
0953 25274  006 036207                      ZERO B
0954 25275  227 174707            READ      INC  PNM  P         START READ
0955 25276  370 036747            RTN                           RETURN
0956                      *
0957                      *
0958                      ************************************************************
0959                              END
END OF PASS 2: NO ERRORS

PAGE 0024 RTE MICRO CROSS-REFERENCE REV.1813 771212
SYMBOLS=0062  REFERENCES=0168  SOURCE LINES=0959
ALOG       0484  0018  0561
ALOGT      0561  0023
ATAN       0580  0019
ATAN1      0607  0602
ATAN3      0616  0593  0606
BAB23      0207  0504  0753  0766  0903  0914
BOTH       0774  0758
CKSGN      0464  0427
COS        0385  0020
COSAG      0400  **NOT REFERENCED**
DISPLAY    0702  0693  0696  0699
ERRET      0372  0475  0553  0800  0877
EVEN       0747  **NOT REFERENCED**
EXIT1      0361  0356  0465  0467  0545  0570
EXIT2      0664  0656  0661  0909  0940
EXIT3      0948  0740
EXP        0809  0022  0898
EXPERR     0872  0835  0838  0864
EXPNXT     0840  0833
FADA2      0205  0508  0614  0907
FADD       0163  0328  0338  0352  0416  0422  0426  0445  0453  0459
           0535  0539  0630  0640  0654  0774  0785  0790  0855  0860
           0921  0936
FDIV7      0175  0509  0615  0655  0780  0789  0857  0908  0928
FDVAC2     0219  0335  0360  0527  0605  0637
FETCH      0013  0128
FLUN       0040  0488  0742
FMPY       0097  0345  0355  0409  0438  0462  0544  0569  0647  0845
```

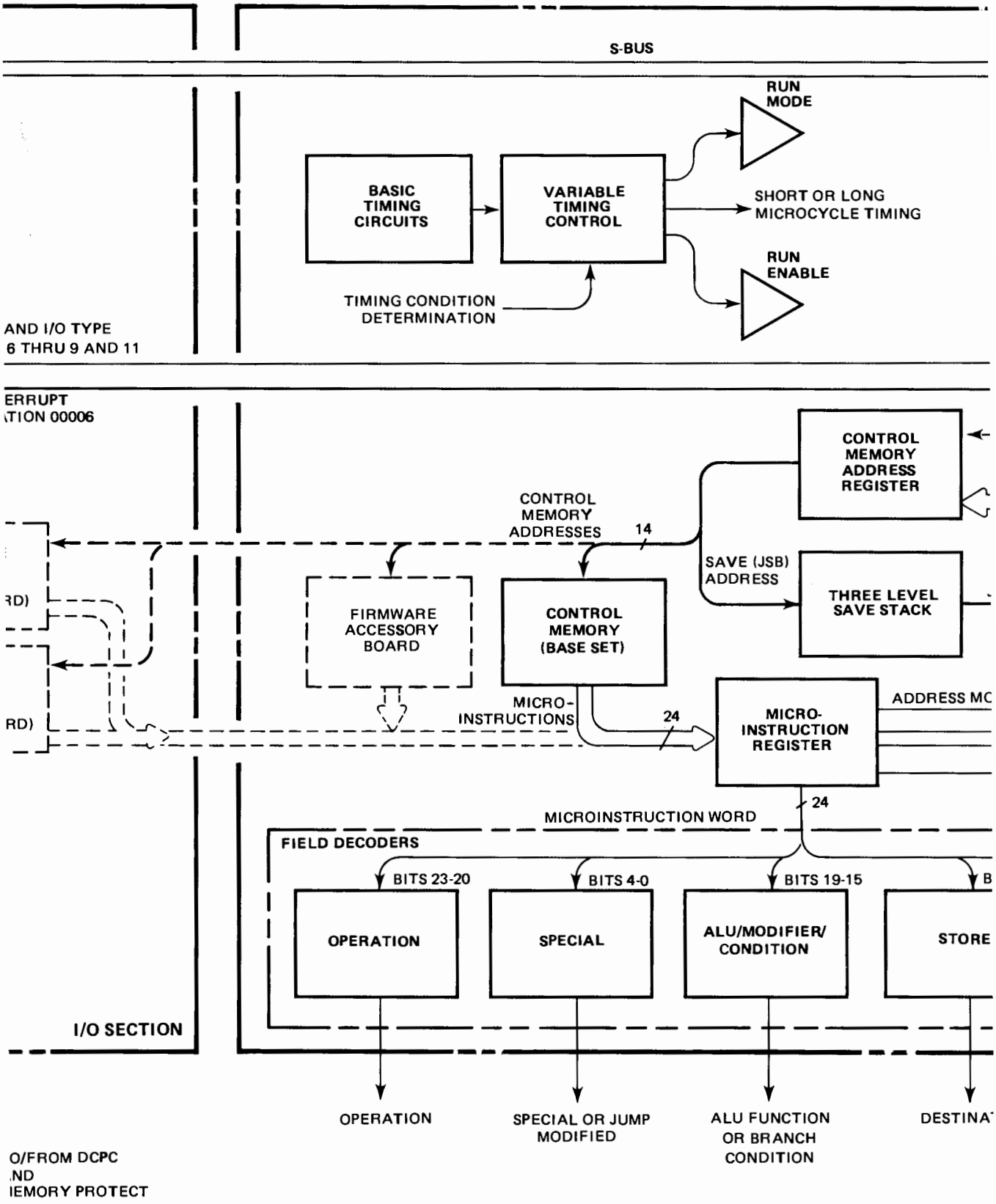
```
PAGE 0025 RTE MICRO CROSS-REFERENCE REV.1813 771212
          0939
FOPI      0259  0314  0392
FSUB      0247  0610
FSUB2     0151  0520  0659  0848
HORI      0014  0141
INTRT1    0137  0117
LGNXT     0495  0491
LOGERR    0548  0486  0487
LOOP1     0118  0121  0137
NEG       0952  0945
NEGATE    0243  0466  0662
NEGCHK    0830  **NOT REFERENCED**
ODD       0760  0745
PDONE     0083  0057
PNEXT1    0064  0061
PNEXT2    0077  0072  0074
POS       0588  0585  0586
POSCHK    0835  0828
PWR2      0055  0792  0861
REDUCE    0270  0820
RNXT      0294  0290  0292
SELFTEST  0684  0031
SIN       0389  0021  0386
SINAG     0429  0398
SINERR    0470  0393
SQRERR    0795  0741
SQRT      0738  0017  0691
TAN       0312  0016
```

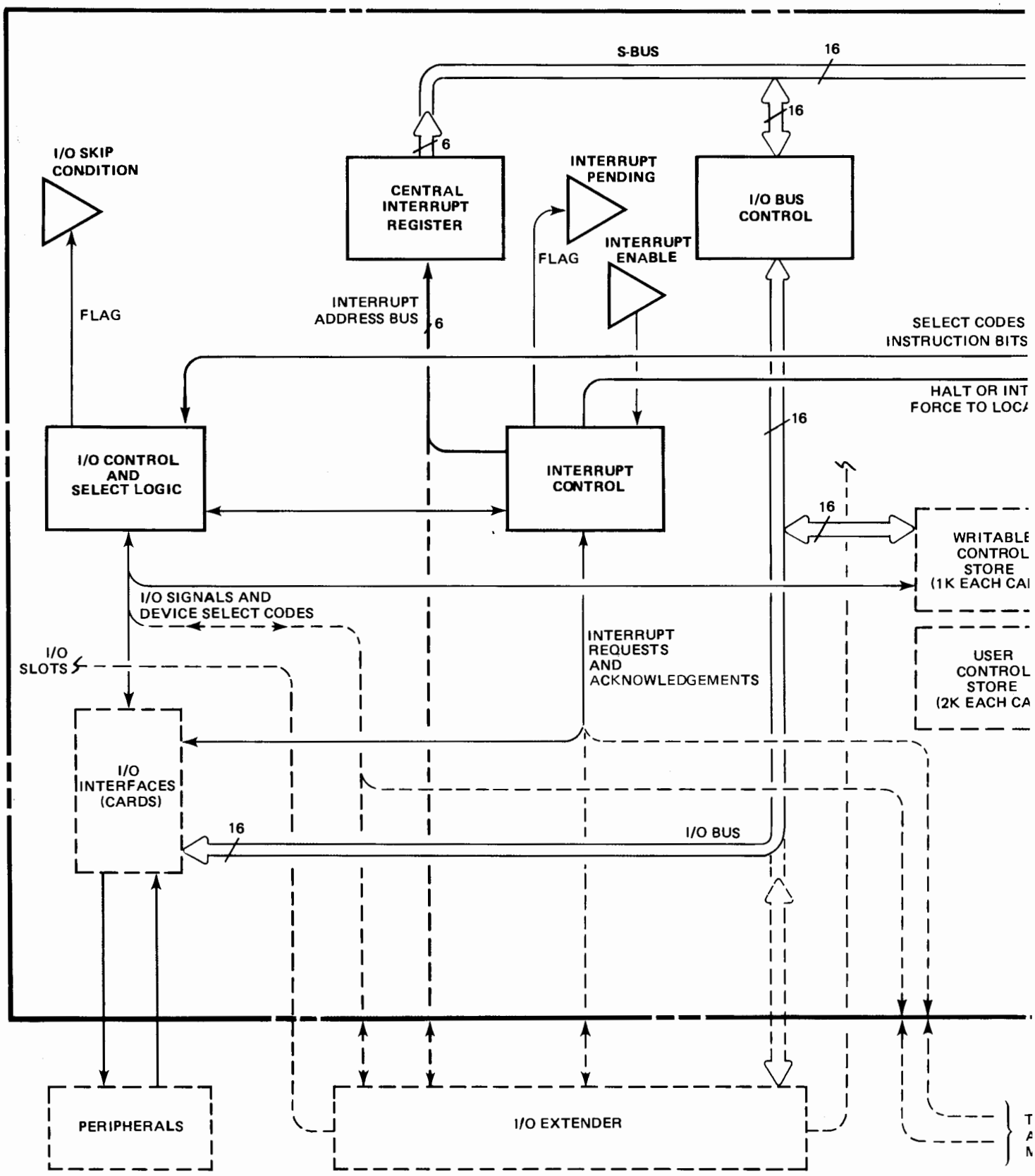
```
PAGE 0026 RTE MICRO CROSS-REFERENCE REV.1813 771212
TANERR    0367  0315
TANH      0886  0024
TANH1     0911  0891
TANH2     0943  0894
VMPY      0231  0417  0423  0446  0454  0536
WAIT1     0116  0055  0097  0151  0163  0175  0187  0205  0219  0231
          0247  0280  0286  0297  0325  0361  0406  0435  0500  0505
          0510  0607  0611  0616  0627  0664  0777  0786  0904
XSQ       0187  0318  0397  0840
ZERO1     0671  0584
ZERO2     0867  0832
```

Appendix H
FUNCTIONAL BLOCK DIAGRAM

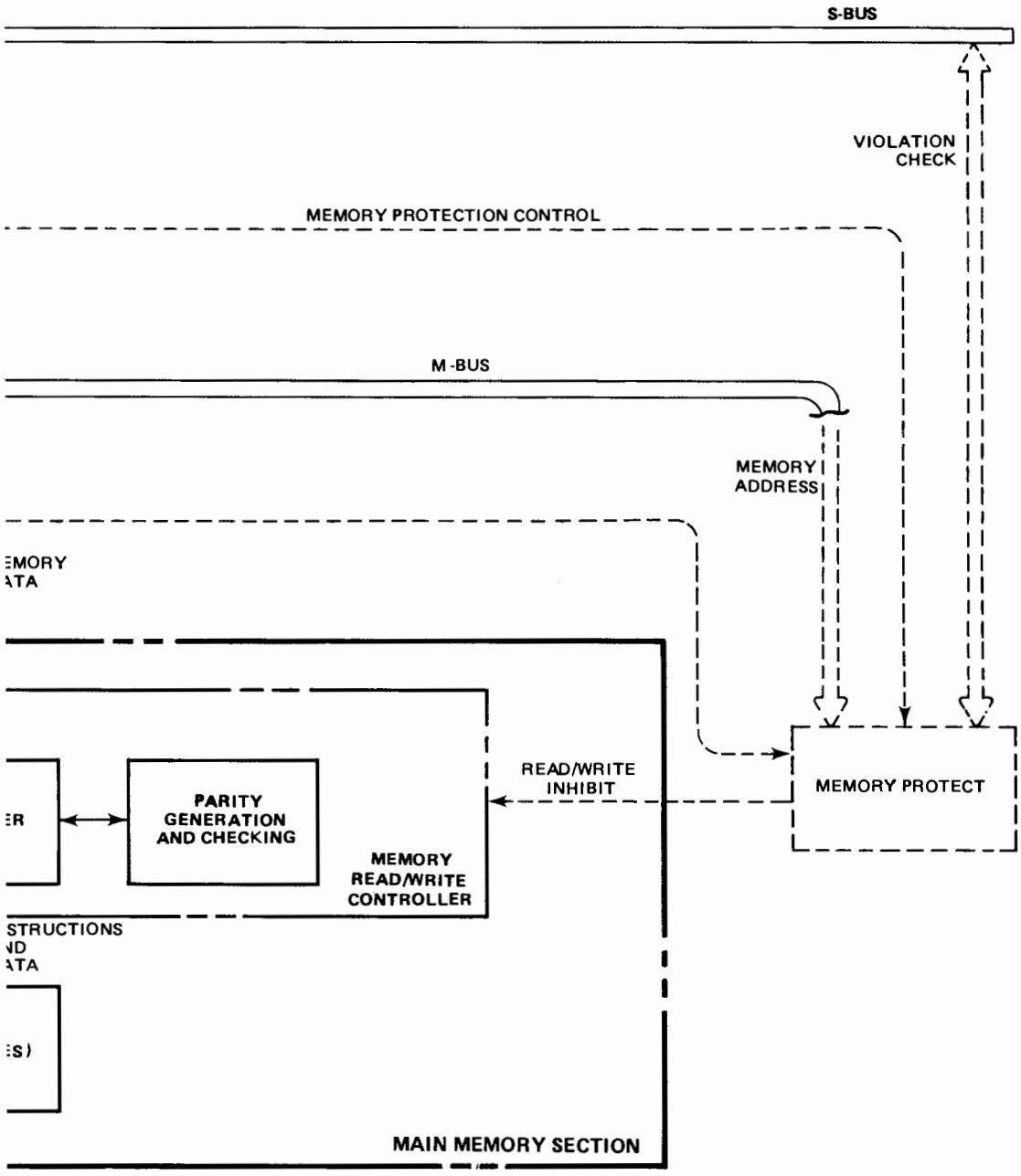


E-Series and F-Series Computer
Functional Block Diagram
(Sheet 1 of 2)


S-BUS
RUN MODE
BASIC TIMING CIRCUITS
VARIABLE TIMING CONTROL
SHORT OR LONG MICROCYCLE TIMING
RUN ENABLE
TIMING CONDITION DETERMINATION
AND I/O TYPE
6 THRU 9 AND 11
ERRUPT
TION 00006
CONTROL MEMORY ADDRESS REGISTER
CONTROL MEMORY ADDRESSES
14
SAVE (JSB) ADDRESS
THREE LEVEL SAVE STACK
FIRMWARE ACCESSORY BOARD
CONTROL MEMORY (BASE SET)
MICRO-INSTRUCTIONS
24
MICRO-INSTRUCTION REGISTER
ADDRESS MC
24
MICROINSTRUCTION WORD
FIELD DECODERS
BITS 23-20
BITS 4-0
BITS 19-15
OPERATION
SPECIAL
ALU/MODIFIER/ CONDITION
STORE
I/O SECTION
OPERATION
SPECIAL OR JUMP MODIFIED
ALU FUNCTION OR BRANCH CONDITION
DESTINA
O/FROM DCPC
ND
IEMORY PROTECT


S-BUS
16
16
6
I/O SKIP CONDITION
CENTRAL INTERRUPT REGISTER
INTERRUPT PENDING
I/O BUS CONTROL
FLAG
INTERRUPT ENABLE
FLAG
INTERRUPT ADDRESS BUS
6
SELECT CODES INSTRUCTION BITS
HALT OR INT
FORCE TO LOC
16
I/O CONTROL AND SELECT LOGIC
INTERRUPT CONTROL
16
WRITABLE CONTROL STORE (1K EACH CA
I/O SIGNALS AND DEVICE SELECT CODES
I/O SLOTS
INTERRUPT REQUESTS AND ACKNOWLEDGEMENTS
USER CONTROL STORE (2K EACH CA
I/O INTERFACES (CARDS)
16
I/O BUS
PERIPHERALS
I/O EXTENDER

7115-21

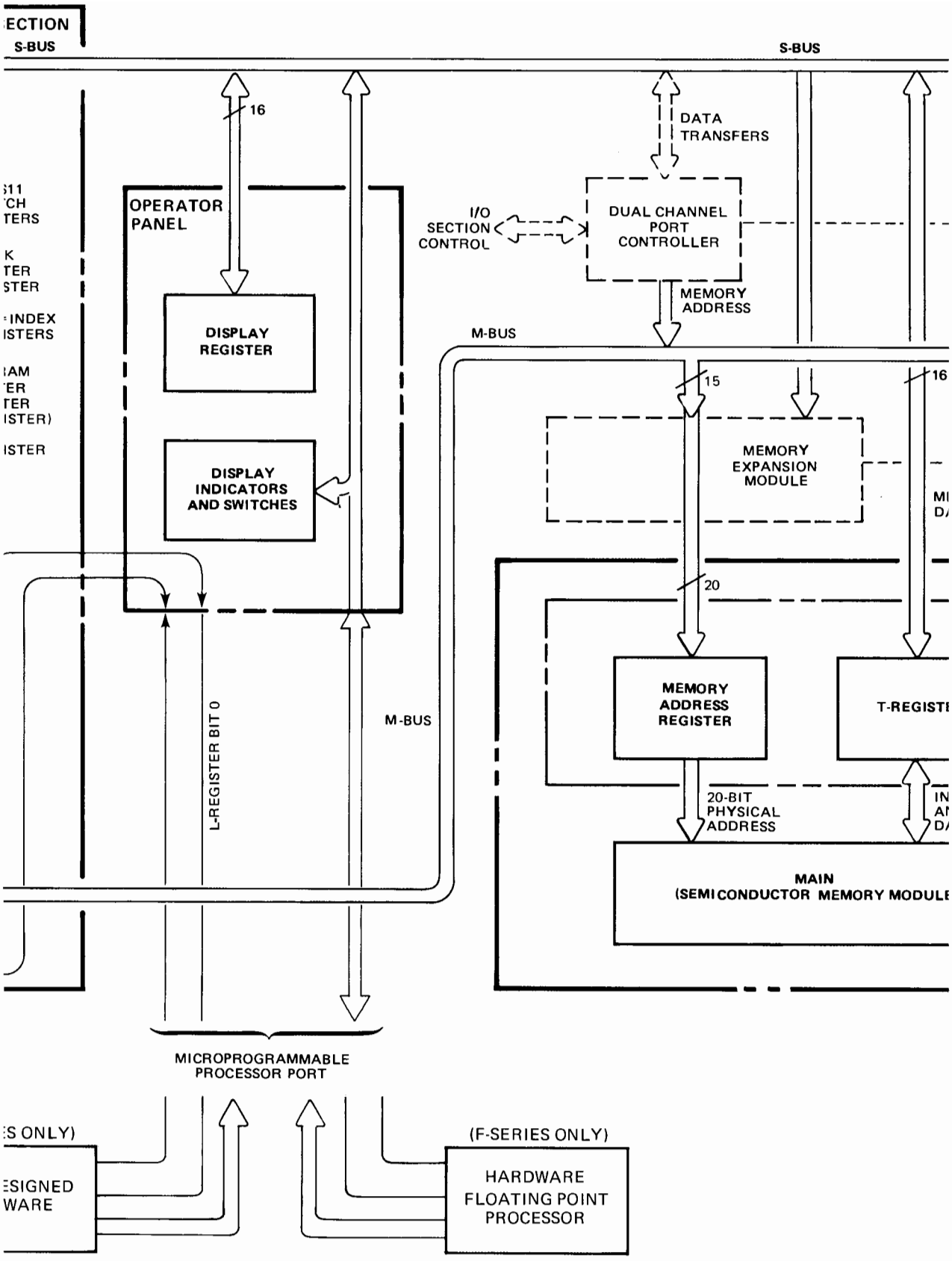


NOTES:

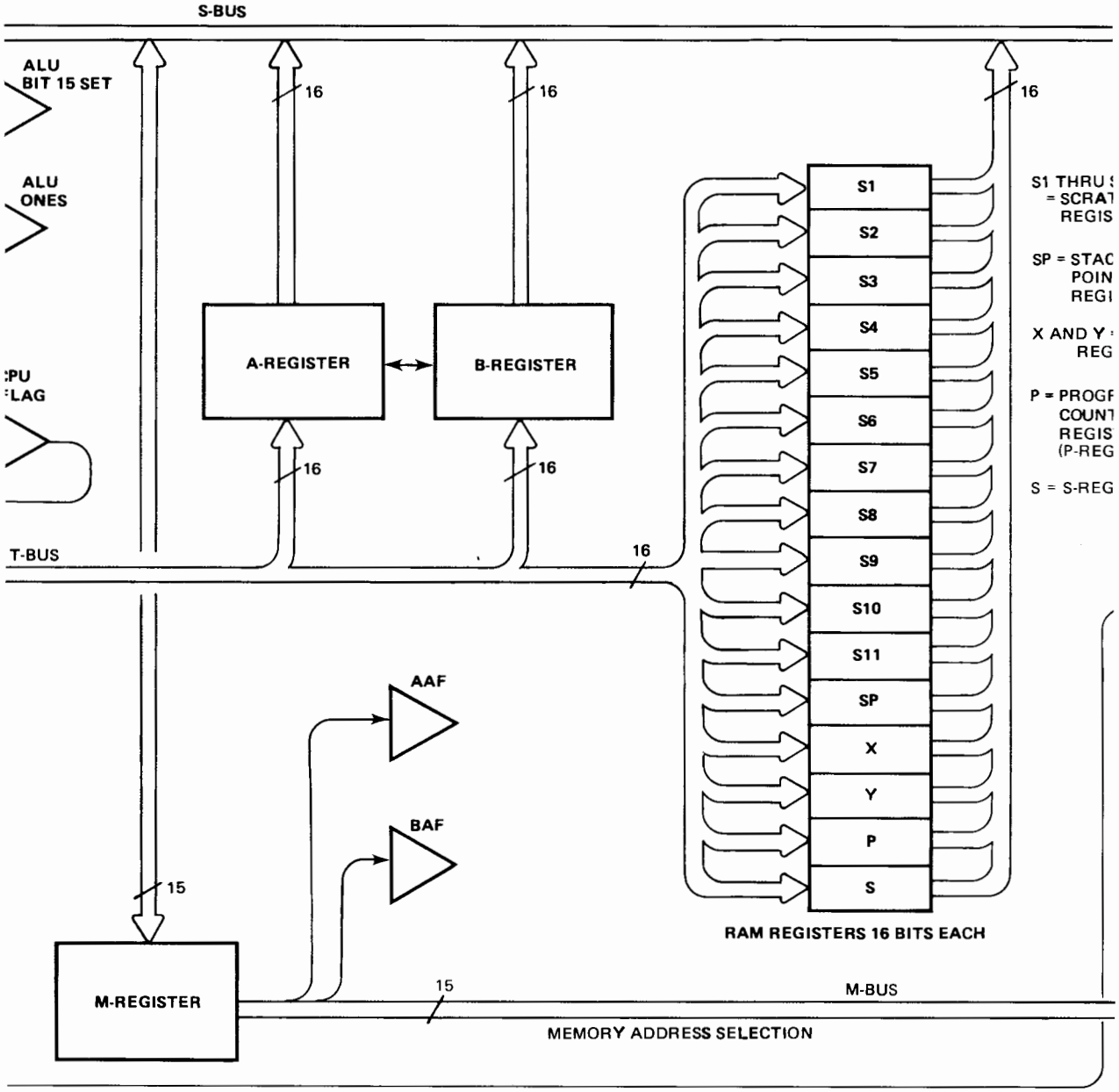
1. ▷ DENOTES PROGRAM FLAGS OR FLIP-FLOPS

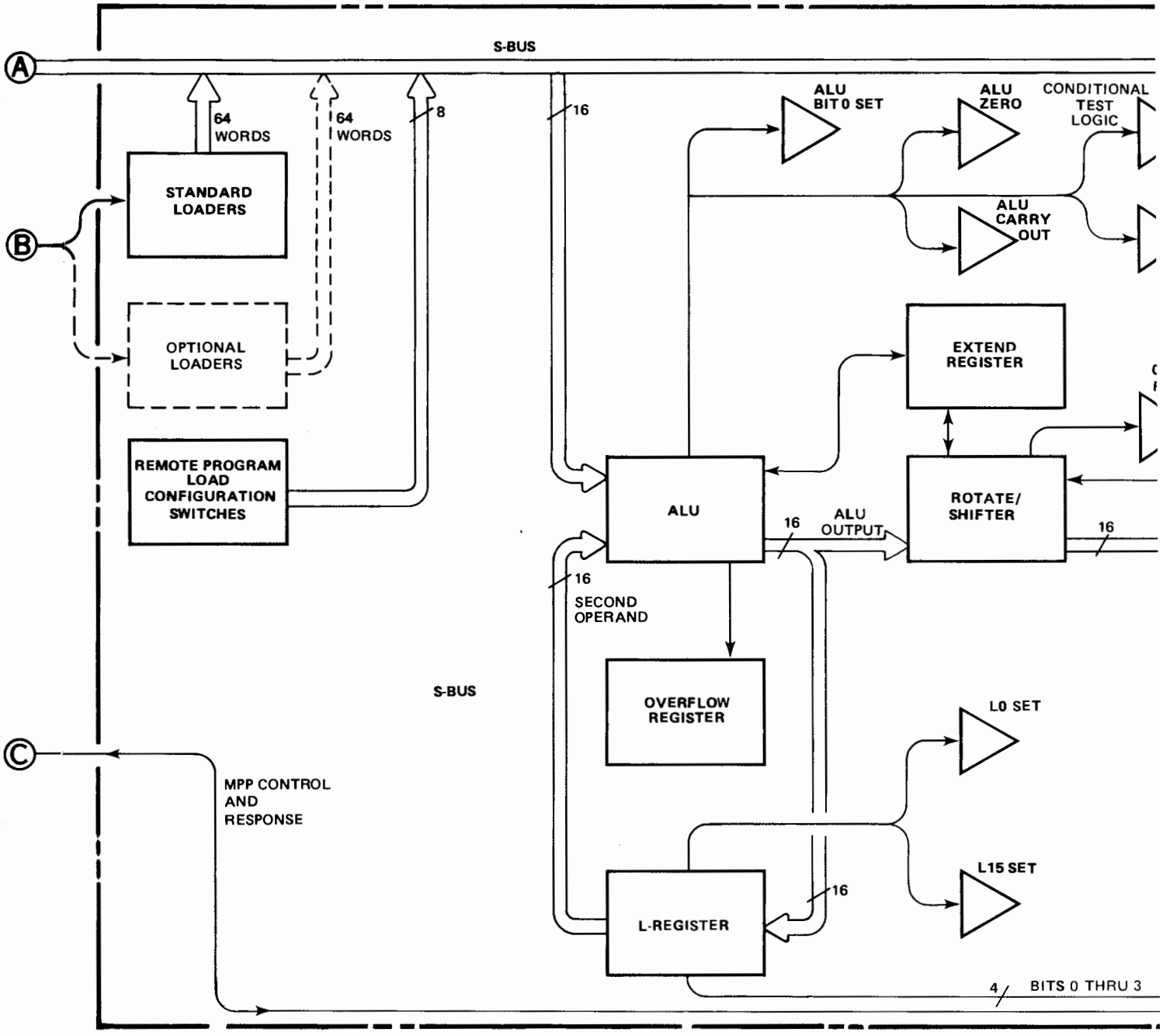
2. ACCESSORIES SHOWN IN LIGHT DASHED LINES

E-Series and F-Series Computer
Functional Block Diagram
(Sheet 2 of 2)


ECTION
S-BUS
S-BUS
16
DATA
TRANSFERS
OPERATOR
PANEL
I/O
SECTION
CONTROL
DUAL CHANNEL
PORT
CONTROLLER
MEMORY
ADDRESS
DISPLAY
REGISTER
M-BUS
15
16
MEMORY
EXPANSION
MODULE
DISPLAY
INDICATORS
AND SWITCHES
20
MEMORY
ADDRESS
REGISTER
T-REGISTE
L-REGISTER BIT 0
M-BUS
20-BIT
PHYSICAL
ADDRESS
MAIN
(SEMICONDUCTOR MEMORY MODULE
MICROPROGRAMMABLE
PROCESSOR PORT
(F-SERIES ONLY)
HARDWARE
FLOATING POINT
PROCESSOR

7115-22


S-BUS
ALU
BIT 15 SET
ALU
ONES
:PU
:LAG
16
16
16
A-REGISTER
B-REGISTER
16
16
T-BUS
16
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
SP
X
Y
P
S
RAM REGISTERS 16 BITS EACH
S1 THRU S
= SCRAT
REGIS
SP = STAC
POIN
REGI
X AND Y
REG
P = PROGF
COUNT
REGIS
(P-REG
S = S-REG
AAF
BAF
15
M-REGISTER
15
M-BUS
MEMORY ADDRESS SELECTION


S-BUS
A
B
C
64 WORDS
64 WORDS
8
16
STANDARD LOADERS
OPTIONAL LOADERS
REMOTE PROGRAM LOAD CONFIGURATION SWITCHES
ALU BIT 0 SET
ALU ZERO
CONDITIONAL TEST LOGIC
ALU CARRY OUT
EXTEND REGISTER
ALU
16
ALU OUTPUT
ROTATE/ SHIFTER
16
16
SECOND OPERAND
S-BUS
OVERFLOW REGISTER
L0 SET
MPP CONTROL AND RESPONSE
L15 SET
16
L-REGISTER
4
BITS 0 THRU 3

7115-22A

Index

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X Y Z

MANUAL PART NO. 02109-90004
MICROFICHE PART NO. 02109-90010
Printed in U.S.A. 7/78

HEWLETT hp PACKARD