

HP 1000 E-Series and F-Series Computer Microprogramming Reference Manual



HEWLETT
PACKARD

PUBLICATION NOTICE

Text changes are supplied in the form of manual change notices or complete revisions to the manual. The publication history of any changes to this edition of the manual is given under "Publication History" below. The last change listed reflects the current software.

All changed pages are identified by a change number at the bottom of the page. Changed information is specifically identified by a vertical line (change bar) on the outer margin of the page.

PUBLICATION HISTORY

Second Edition November 1979 (Software Revision Code 1940)
Change 1 April 1980 (Software Revision Code 2013)

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

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.

PART I – Why Microprogramming?

Section 1 Microprogramming Concept

Section 2 Controllable Functions

PART II – Microprogramming Methods

Section 3 Microprogramming Preparation Steps

Section 4 Microinstruction Formats

Section 5 Timing Considerations

Section 6 Mapping to the User's Microprogramming Area

Section 7 Microprogramming Considerations

Section 8 Preparation with the Microassembler

PART III – Microprogramming Support Software and Hardware

Section 9 Using the RTE Microassembler

Section 10 Using the RTE Microdebug Editor

Section 11 Writable Control Store (WCS) Support Software

Section 12 Using pROM Generation Support Software and Hardware

Section 13 Using Special Facilities of the Computer

PART IV – Microprogramming Examples

Section 14 Microprograms

APPENDIXES

Appendix A Abbreviations and Definitions

Appendix B Microinstruction Formats

Appendix C Micro-Order Summary and Specialized Microprogramming

Appendix D Microprogramming Form

Appendix E Object Tape Formats

Appendix F HP M-Series-to-E-/F-Series Micro-Order Comparison Summary

**Appendix G HP 1000 E-Series Computer Base Set Microprogram Listing
and F-Series Jump Tables**

Appendix H Functional Block Diagram

Index

CONTENTS

Section	Page	pROM Tape Generator	3-5
Preface	iii	Preparatory Steps	3-5
PART I — WHY MICROPROGRAMMING?			
Section 1	Page	Section 4	Page
MICROPROGRAMMING CONCEPT		MICROINSTRUCTION FORMATS	
Microprogramming Overview	1-2	Microinstruction Binary Structures	4-1
Selecting an Analysis Method	1-3	Microassembler Formats	4-5
The Microprogramming Process	1-3	Word Type I	4-6
Executing Your Microprogram	1-6	Word Type II	4-6
Some Microprogramming Related Products	1-7	Word Type III	4-6
Summary	1-8	Word Type IV	4-9
		Micro-Order Definitions	4-10
		Summary	4-10
Section 2	Page	Section 5	Page
CONTROLLABLE FUNCTIONS		TIMING CONSIDERATIONS	
Computer Functions that Can Be Controlled	2-1	Computer Sections Involved in Timing	5-1
Control Processor	2-2	Review and Expansion of Timing Definitions	
Arithmetic/Logic Section	2-2	and Terms	5-2
Main Memory Section	2-2	Timing Variables	5-3
Input/Output Section	2-2	Short/Long Microcycles	5-3
Operator Panel	2-2	Pause	5-4
Memory Protect	2-5	Freeze	5-5
Dynamic Mapping System	2-5	Overall Timing	5-7
Dual Channel Port Controller	2-5	Timing Calculations	5-8
A Closer Look at the Functions	2-5	Arithmetic/Logic Section Operations	5-10
Some Definitions and Timing Points	2-10	Control Memory Branches	5-13
How Do All These Functions Interrelate?	2-10	I/O Operations	5-13
Control Memory	2-11	Main Memory Operations	5-14
Let's Talk About The Base Set	2-14	Reading from Memory	5-14
An Operational Overview	2-15	Writing to Memory	5-15
Fetching	2-16	Summary	5-17
Execution	2-17		
Microprogrammed Accessories	2-18	Section 6	Page
Summary	2-18	MAPPING TO THE USER'S	
		MICROPROGRAMMING AREA	
		Control Memory Mapping Method	6-2
		Software Entry Points	6-2
		The User Instruction Group	6-2
		HP Reserved UIG Codes	6-3
		User Area UIG Codes	6-4
		User's Area Mapping Example	6-6
		Main Memory/Control Memory Linkage	6-6
		Assembler Procedure	6-8
		Parameter Passing	6-9
		Control Memory/Main Memory Linkage	6-12
		Some Main Memory Program Procedures	6-12
		The MIC Pseudo-Instruction	6-12
		Parameter Assignment Example	6-13
		Example MIC Pseudo-Instruction Use	6-13
		Calling Microprograms from FORTRAN	6-14
		Summary	6-15
PART II — MICROPROGRAMMING METHODS			
Section 3	Page		
MICROPROGRAMMING PREPARATION			
STEPS			
Environment	3-1		
Microprogramming Hardware	3-1		
Microprogramming Support Software	3-3		
The RTE Microassembler	3-3		
Microassembler Cross-Reference Generator	3-3		
RTE Microdebug Editor	3-3		
Driver DVR36	3-4		
WLOAD	3-4		
Loading the Microprogramming Support			
Software	3-4		

CONTENTS (continued)

<p>Section 11 Page WRITABLE CONTROL STORE (WCS) SUPPORT SOFTWARE WCS Hardware 11-1 WCS Software 11-2</p> <p>Section 12 Page USING pROM GENERATION SUPPORT SOFTWARE AND HARDWARE Using the pROM Tape Generator 12-1 Initialize Phase 12-2 Punch Phase 12-5 Verify Phase 12-6 pROM Tape Generator Error Messages 12-7 pROM Hardware 12-9</p> <p>Section 13 Page USING SPECIAL FACILITIES OF THE COMPUTER Block I/O Data Transfers 13-2 Block I/O Byte Packing Burst Input Microprogram 13-2 Block I/O Address/Data Burst Input Microprogram 13-5 Block I/O Word Burst Output Microprogram 13-6 Microprogrammable Processor Port 13-6 Hardware Interface 13-7 MPP & MB10 Considerations 13-8 MPP Microprogram (E-Series Only) 13-9 Summary of MPP Transfer Rates 13-10 Hardware Floating Point Processor (F-Series Only) 13-11 Controllable Functions 13-11 Data Formats 13-12 FPP Instruction Word Format 13-12 Exponent Format 13-12 FPP Operation 13-14 Operand Source 13-14 Operand Length 13-14 Data Operations 13-14 Fix and Float Operations 13-15 Accumulator Operations 13-15 MPP Micro-Orders 13-15 FPP Instruction Store 13-15 FPP Addressing 13-17 Instruction Execution 13-17 Operand to FPP 13-17 Result to CPU 13-18 MPP1 Micro-Order Considerations 13-18 FPP Complete Test 13-18 Overflow Detection 13-19</p>	<p>MPP Micro-Order Summary 13-19 FPP Microprogramming Considerations 13-19 FPP Operation Execution Times 13-19 Execution in Progress 13-20 Interrupt Considerations 13-21 Microprogrammed FPP Operation Example 13-21 Microprogramming the Floating Point Processor 13-21</p> <p>PART IV — MICROPROGRAMMING EXAMPLES</p> <p>Section 14 Page MICROPROGRAMS WCS Initialization 14-2 Microprogramming with MDE 14-3 Shell Sort Example 14-5 Microprogrammed I/O Operation Example 14-18</p> <p>APPENDIX A Page ABBREVIATIONS AND DEFINITIONS A-1</p> <p>APPENDIX B Page MICROINSTRUCTION FORMATS B-1</p> <p>APPENDIX C Page MICRO-ORDER SUMMARY AND SPECIALIZED MICROPROGRAMMING C-1</p> <p>APPENDIX D Page MICROPROGRAMMING FORM D-1</p> <p>APPENDIX E Page OBJECT TAPE FORMATS E-1</p> <p>APPENDIX F Page HP 1000 M-SERIES-TO-HP 1000 E/F-SERIES MICRO-ORDER COMPARISON SUMMARY F-1</p> <p>APPENDIX G Page E-SERIES COMPUTER BASE SET MICROPROGRAM LISTING AND F-SERIES JUMP TABLES G-1</p> <p>APPENDIX H Page FUNCTIONAL BLOCK DIAGRAM H-1</p> <p>Index I-1</p>
--	---

ILLUSTRATIONS

Title	Page	Title	Page
Microprogramming Implementation Process	1-5	Detailed Pause Time Calculation Flowchart	
Some Microprogramming Products	1-7	(Using an HP 2102B Memory as an Example) ..	5-16
HP 21MX E-Series Computer Overall Block		Overflow Register Control	7-11
Diagram	2-3	Scheduling MDE (MDEP)	10-16
Simplified Control Processor Block Diagram	2-9	Calling MDE (MDES)	10-16
E-Series Control Memory Map	2-12	Interactive Debugging Operations	10-17
F-Series Control Memory Map	2-13	General Tape Format	12-2
Word Type/Binary Format Summary	4-1	FPP Overall Functional Block Diagram	13-11
Micro-Order Binary Formats	4-3	Floating Point Data Format	13-12
RTE Microassembler Word Format Summary	4-5	FPP Instruction Word Format	13-13
Microassembler Format Micro-Orders	4-7	Typical FPP Microprogramming	
Jump Address Decoding	4-9	Sequence Flowchart	13-16
Basic Timing Definitions	5-2	FPP Microprogramming Example Flowchart	13-23
Variable Microcycles with Pause Conditions	5-6	Example 3, Microprogrammed Shell Sort	
Overall Microcycle Timing Flowchart	5-7	Flowchart	14-8
Consolidated Microcycle Estimating Flowchart	5-9	Example 4, Microprogrammed Privileged Section	
Detailed Microcycle Time Determination		Flowchart	14-24
Flowchart	5-11		

TABLES

Title	Page	Title	Page
Computer Functions	2-6	MEM Signals Invoked by Micro-Orders	7-28
Program Partitioning Capabilities	3-5	DMS Micro-Order Control Signals	7-29
Preparatory Steps	3-6	Microassembler and Cross-Reference Generator	
Manual/Software References	3-8	Error Messages	9-8
Micro-Order Definitions	4-11	MDE Operator Command Syntax	10-1
Summary of Timing Factors	5-18	Summary of Microdebug Editor Commands	10-3
Control Memory User Instruction Group		Microdebug Editor Error Messages	10-13
Software Entry Point Assignments (E-Series)	6-4	Default Formats by Vendor	12-4
Software Entry Point Assignments		MPP Signal Summary	13-7
(F-Series)	6-5	Special Facilities Transfer Rate Summary	13-10
Backplane I/O Signal Generation Determined by		Overflow and Underflow Ranges	13-13
IR Bits 11 through 6	7-19	Summary of FPP Control Micro-orders	13-19
I/O Micro-Order Summary	7-26	FPP Operation Internal Execution Times	13-20

PART I
Why Microprogramming?

Section 1

MICROPROGRAMMING CONCEPT

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).

Concept

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 unwieldy 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 grammatical 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.

Concept

- 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 be 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

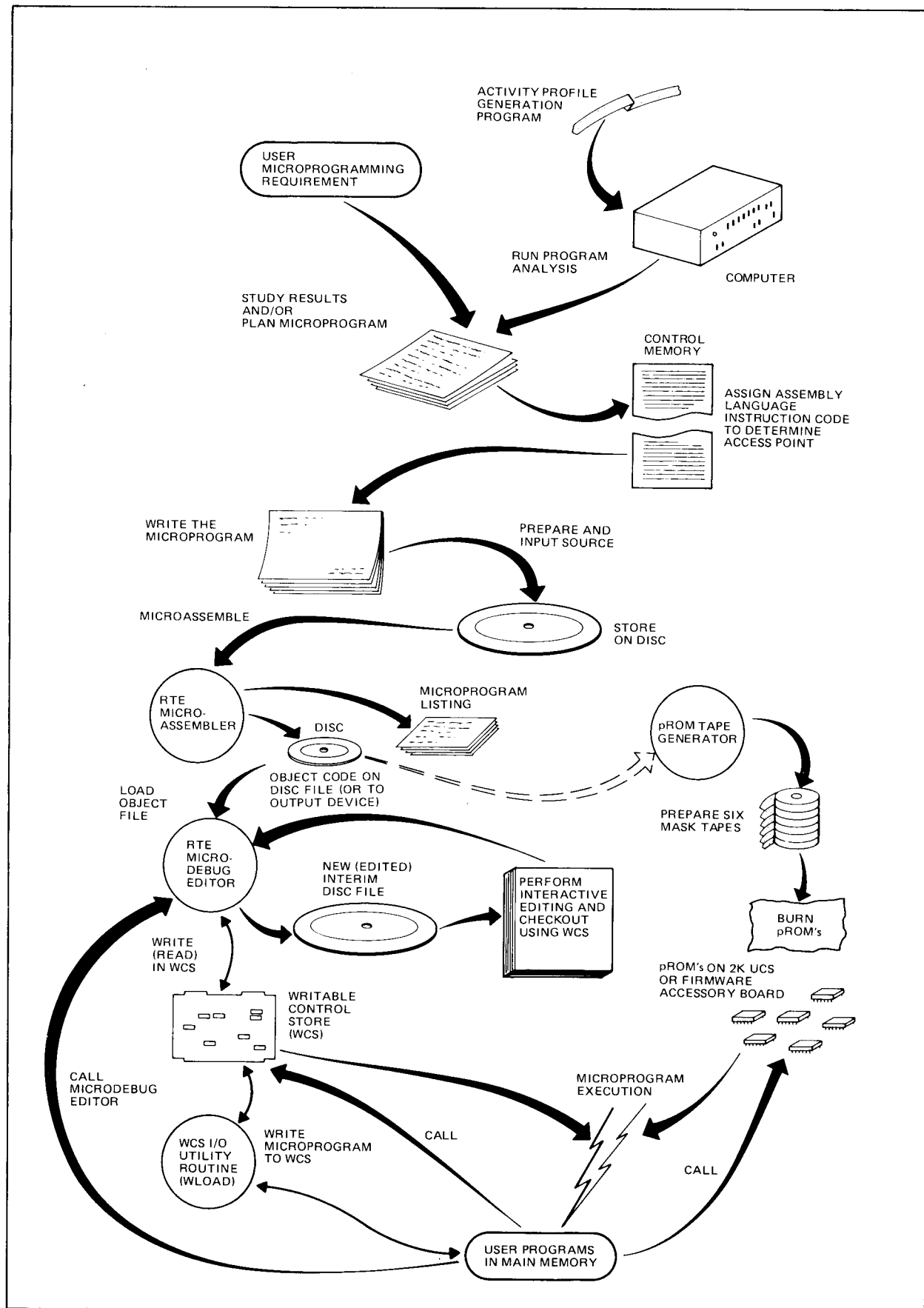
The HP 13197A Writable Control Store Kit is an integral part of microprogramming. Information on writing microprograms to be stored in WCS is the primary purpose of this manual; however, installation and additional reference information on WCS will be found in the *HP 13197A Writable Control Store Reference 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), the HP 13047A User Control Store (UCS) Kit (in the I/O card cage), or on the HP 12791A Firmware Expansion Module (FEM) (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 12791A Firmware Expansion Module Installation and Reference Manual*, part no. 12791-90001.



7115-1

Figure 1-1. Microprogramming Implementation Process

Concept

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.)

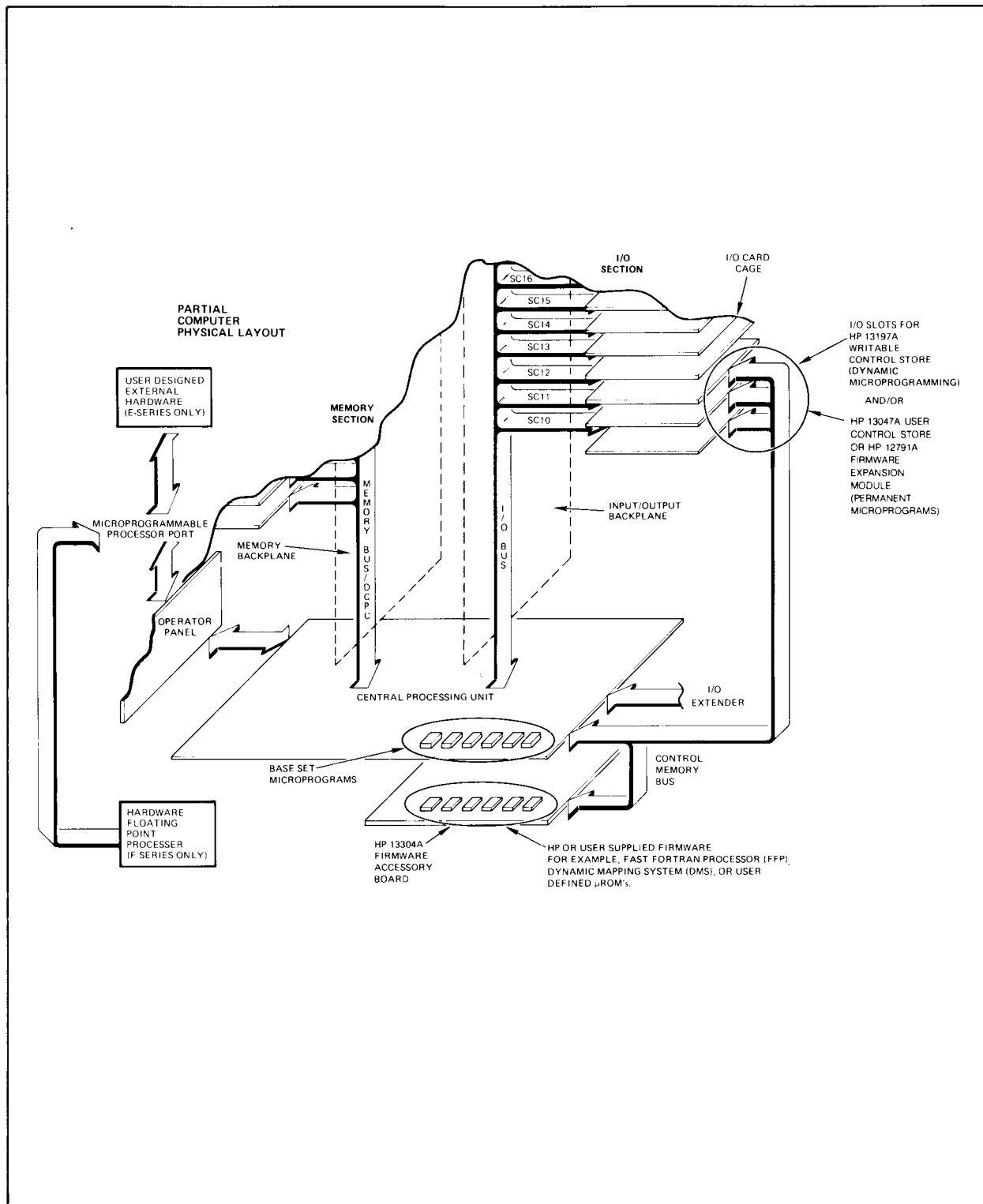
Either the WCS I/O Utility routine WLOAD can be used to load WCS (through a call from FORTRAN, ALGOL, or Assembly language) or the MDE can be used to load WCS. The microprogram can then be called for execution from the main program in the same manner as described for a pROM stored microprogram. To summarize,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



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).
- HP 12791A Firmware Expansion Module (FEM).

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, FEM, 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, 8K microword capacity FEM, 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, the WCS, FEM, 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.

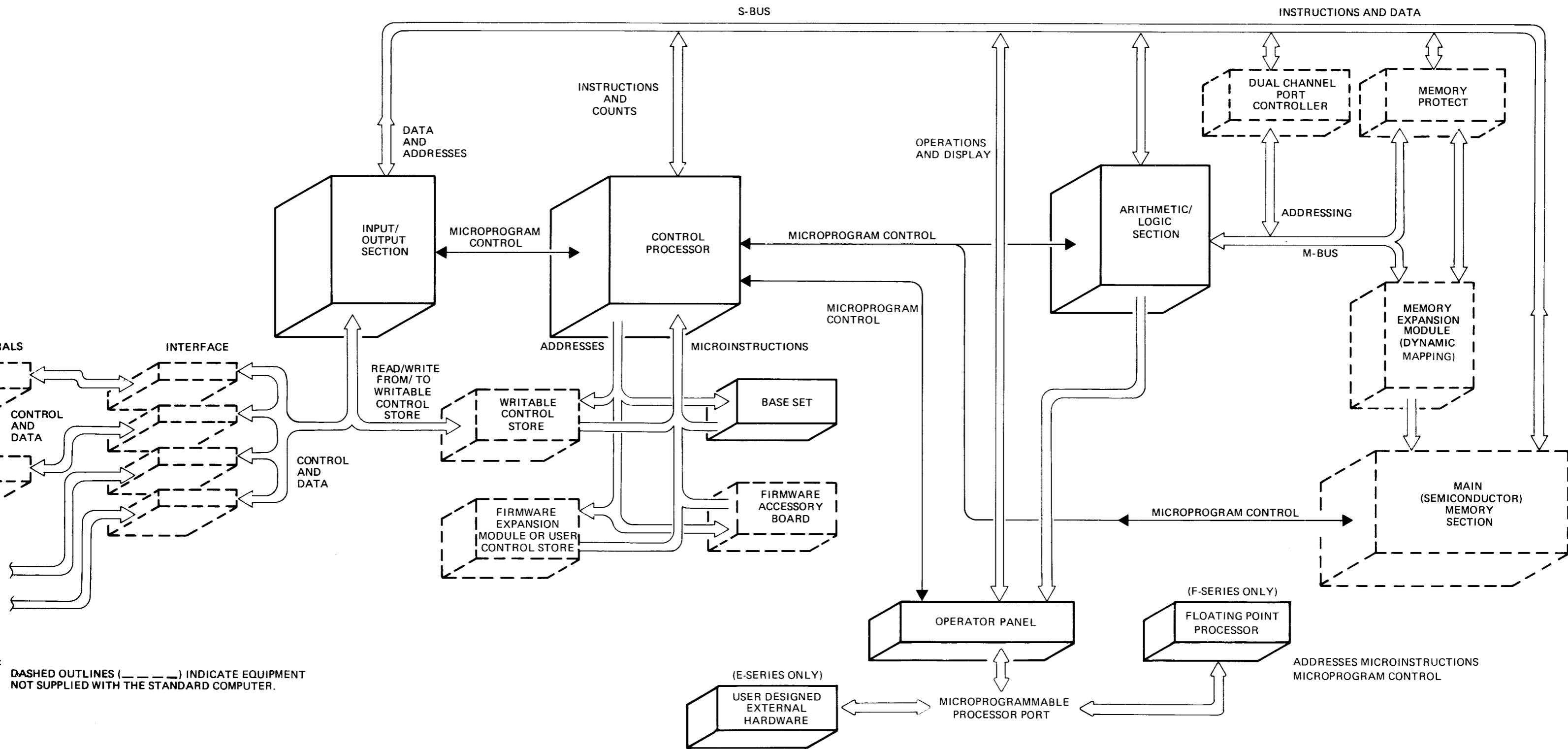


Figure 2-1. HP 1000 E - and F - Series Computer Overall Block Diagram

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	<p>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.</p> <p style="text-align: center;">NOTE</p> <p>"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.</p>
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	<p>Testable conditional flags associated with the ALU and R/S functions include:</p> <ul style="list-style-type: none"> ALU Bit 0 Set ALU Bit 15 Set ALU Carry Out ALU Ones ALU Zero CPU Flag

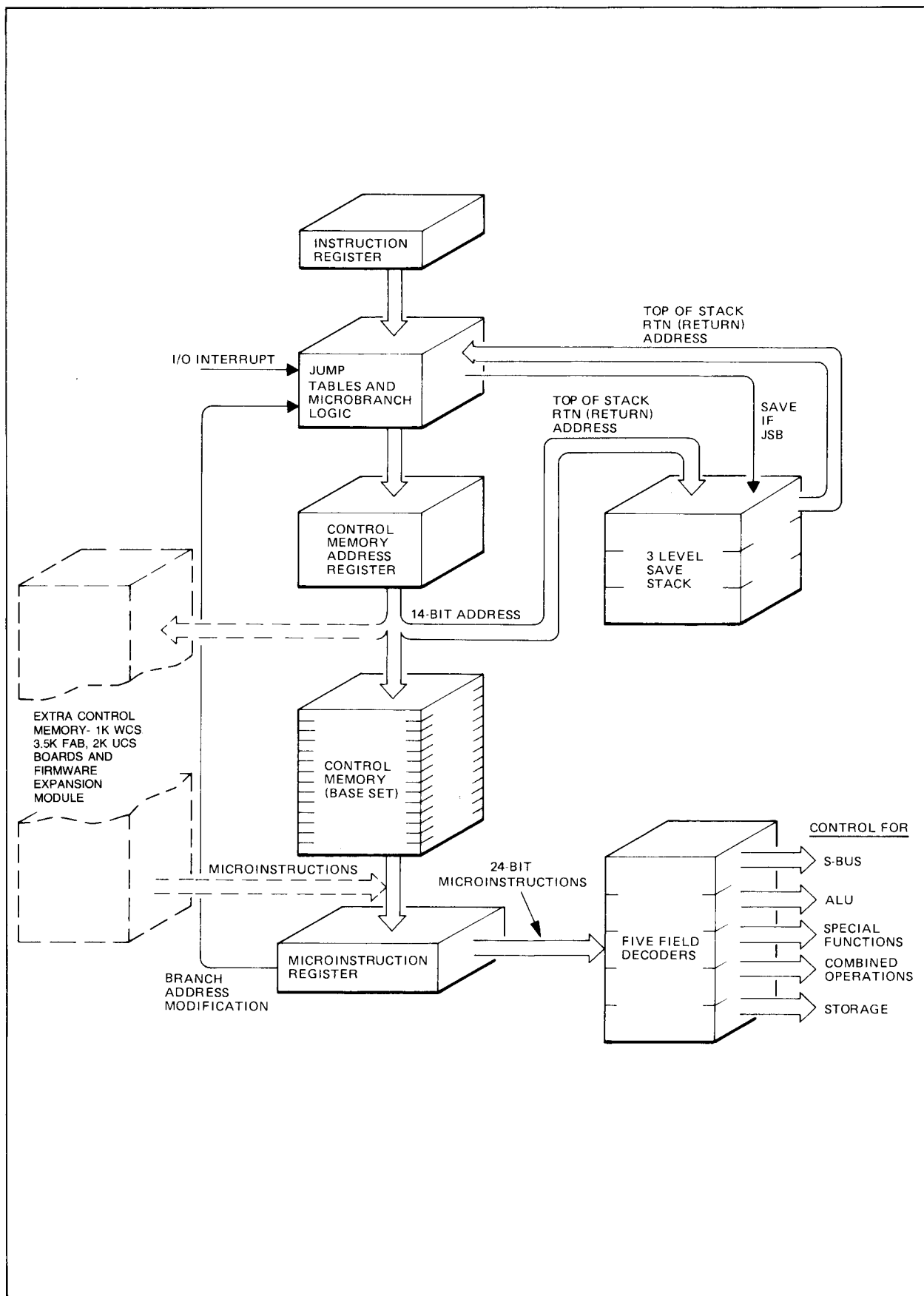
Table 2-1. Computer Functions (Continued)

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		SOFTWARE ENTRY POINT		
		DECIMAL	OCTAL			
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		
AVAILABLE FOR USER MICROPROGRAMMING	11	02816-03071	05400-05777	NO	4K	
	12	03072-03327	06000-06377	NO		
	13	03328-03583	06400-06777	NO		
	14	03584-03849	07000-07377	NO		
AVAILABLE FOR USER MICROPROGRAMMING	15	03850-04095	07400-07777	NO	5K	
	16	04096-04351	10000-10377	NO		
	17	04352-04607	10400-10777	NO		
	18	04608-04863	11000-11377	NO		
AVAILABLE FOR USER MICROPROGRAMMING	19	04864-05119	11400-11777	NO	6K	
	20	05120-05375	12000-12377	NO		
	21	05376-05631	12400-12777	NO		
	22	05632-05887	13000-13377	NO		
AVAILABLE FOR USER MICROPROGRAMMING	23	05888-06143	13400-13777	NO	7K	
	24	06144-06399	14000-14377	NO		
	25	06400-06655	14400-14777	NO		
	26	06656-06911	15000-15377	NO		
AVAILABLE FOR USER MICROPROGRAMMING	27	06912-07167	15400-15777	NO	8K	
	28	07168-07423	16000-16377	NO		
	29	07424-07679	16400-16777	NO		
	30	07680-07935	17000-17377	NO		
HP DYNAMIC MAPPING SYSTEM	31	07936-08191	17400-17777	NO		
HP FAST FORTRAN PROCESSOR	32	08192-08447	20000-20377	YES	9K	
	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		
HP RESERVED	44	11264-11519	26000-26377	YES	12K	
	45	11520-11775	26400-26777	YES		
	46	11776-12031	27000-27377	YES		
	47	12032-12287	27400-27777	YES		
HP RESERVED	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		
RECOMMENDED FOR USER MICROPROGRAMMING	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		
RECOMMENDED FOR USER MICROPROGRAMMING	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		
RECOMMENDED FOR USER MICROPROGRAMMING	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		SOFTWARE ENTRY POINT	
		DECIMAL	OCTAL		
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	NO	
	7	01762-02047	03400-03777	NO	
VECTOR INSTRUCTION SET	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	
VECTOR INSTRUCTION SET	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	
HP RESERVED	16	04096-04351	10000-10377	YES	5K
	17	04352-04607	10400-10777	NO	
	18	04608-04863	11000-11377	YES	
	19	04864-05119	11400-11777	NO	
HP RESERVED	20	05120-05375	12000-12377	YES	6K
	21	05376-05631	12400-12777	NO	
	22	05632-05887	13000-13377	NO	
	23	05888-06143	13400-13777	NO	
HP RESERVED	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 HP FAST FORTRAN PROCESSOR	32	08192-08447	20000-20377	YES	9K
	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	NO	
	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	NO	
	43	10918-11263	25400-25777	NO	
HP RESERVED	44	11264-11519	26000-26377	NO	12K
	45	11520-11775	26400-26777	NO	
	46	11776-12031	27000-27377	YES	
	47	12032-12287	27400-27777	YES	
HP RESERVED	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	
RECOMMENDED FOR USER MICROPROGRAMMING	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	
RECOMMENDED FOR USER MICROPROGRAMMING	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	
RECOMMENDED FOR USER MICROPROGRAMMING	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	

Figure 2-4. F-Series Control Memory Map

Functions

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

Functions

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.

Functions

- 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., FEM). 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



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.

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

Steps

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, FEM, 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. The 12791A Firmware Expansion Module can be used for user microprogramming and for the Vector Instruction Set option.

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

- *Your Computer Series Installation and Service Manual.*
- *HP 12791A Firmware Installation and Reference Manual*, part no. 12791-90001.

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 a disc file. 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 a minimum of 12K 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 12K 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 program 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 desirable 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. <ul style="list-style-type: none"> — WLOAD (if not already loaded) — Microassembler — Cross-Reference Generator — Microdebug Editor (MDEP) — Microdebug Editor (MDES) 	3,C F G H I 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.	9, 10, 11
	<div style="border: 1px solid black; display: inline-block; padding: 2px 10px;">CAUTION</div> 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.	
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).	C, 10, 11
	<div style="border: 1px solid black; padding: 2px; display: inline-block;">CAUTION</div>	
	Before executing development microprograms, ensure that your RTE system is not involved in running production programs.	
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	<i>HP 13197A Writable Control Store Reference Manual</i> , part no. 13197-90005.
B	<i>Your Computer Series Installation and Service Manual</i> .
C	<i>Real-Time Executive IV Software System Programming and Operating Manual</i> , part no. 92067-90001, or <i>Real-Time Executive II Software System Programming and Operating Manual</i> , part no. 92001-93001.
D	<i>RTE Driver DVR36 for HP 12978A/13197A Writable Control Store Board Programming and Reference Manual</i> , 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	<i>Your Computer Series Operating and Reference Manual</i> .
M	<i>HP 1000 M/EIF-Series Firmware Installation and Reference Manual</i> , part no. 12791-90001.
N	<i>HP 13047A User Control Store Kit Installation and Service Manual</i> , part no. 13047-90001.
P	<i>HP 21MX/21MX E-Series Computer I/O Interfacing Guide</i> , part no. 02109-90006.
Q	<i>Your Computer Series Engineering and Reference Documentation</i> .
R	<i>HP 12897B Dual-Channel Port Controller Installation Manual</i> , part no. 12897-90005.
S	<i>HP 12892B Memory Protect Installation Manual</i> , part no. 12892-90007.
T	<i>Your RTE Guide for New Users</i> .
U	<i>HP RTE Assembler Reference Manual</i> , part no. 92067-90003.

Section 4

MICROINSTRUCTION FORMATS



MICROINSTRUCTION FORMATS

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.

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 I	OPERATION				ALU				S-BUS				STORE				SPECIAL							
WORD TYPE II					MOD.				OPERAND															
WORD TYPE III	BRANCH				CONDITION				SENSE	ADDRESS (512 WORDS)														
WORD TYPE IV					ZERO				ADDRESS (16K WORDS)															

MOD. MEANS MODIFIER

Figure 4-1. Word Type/Binary Format Summary

Formats

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																																																																																																																																																																																																																																																																																																																																																						
		ARS	0001	ADD	00110	A	00011	A	00011	ASG	11000	CRS	0010	AND	10100	B	00100	B	00100	CLFL	01110	DIV	0101	CMPL	11010	CAB	00001	CAB	00001	COV	01011	ENV	1010	CMPS	11111	CIR	01010	CNTR	01011	DCNT	10101	ENVE	1011	DBLS	00011	CNTR	01011	CNTR	01011	FTCH	11011	LGS	0011	DEC	00000	DES	01110	DSPI	00111	IAK	11001	LWF	0110	INC	01111	DSPI	00111	DSPL	00110	ICNT	10110	MPY	0111	IOR	10001	DSPL	00110	DSPL	00110	INCI	11100	NOP	0000	NAND	11011	IOI	00101	IOO	00101	IOFF	11111	NRM	0100	NOR	11110	LDR	01100	IRCM	01100	IOG	00110	READ	1001	NSAL	11101	M	01101	L	01010	ION	00011	RTN	1111	NSOL	10111	MEU	01001	MPPA	00010	JTAB	00001	WRTE	1000	ONE	10011	MPPA	00010	MPPB	01000	L1	10010			OP1	01110	MPPB	01000	M	01101	L4	10011			OP2	01101	NOP	01111	MEU	01001	MESP	01010			OP3	01011	P	11110	MPPA	00010	MPCK	11110			OP4	01010	S	11111	MPPB	01000	MPP1	11010			OP5	01000	SP	11011	NOP	01111	MPP2	01001			OP6	00111	S1	10000	P	11110	NOP	00111			OP7	00101	S2	10001	PNM	01110	PRST	01101			OP8	00100	S3	10010	S	11111	RJ30	00100			OP10	00010	S4	10011	SP	11011	RPT	10111			OP11	00001	S5	10100	S1	10000	RTN	00000			OP13	11100	S6	10101	S2	10001	R1	10100			PASL	10101	S7	10110	S3	10010	SHLT	11101			PASS	10000	S8	10111	S4	10011	SOV	01100			SANL	11000	S9	11000	S5	10100	SRG1	10001			SONL	10010	S10	11001	S6	10101	SRG2	10000			SUB	01001	S11	11010	S7	10110	SRUN	01000			XNOR	10110	TAB	00000	S8	10111	STFL	01111			XOR	11001	X	11100	S9	11000					ZERO	01100	Y	11101	S10	11001									S11	11010									TAB	00000									X	11100									Y	11101	
WORD TYPE I																																																																																																																																																																																																																																																																																																																																																																								
FIELDS		OPERATION (OP)				MODIFIER	OPERAND (ANY 8-BIT CONSTANT TO THE S-BUS MODIFIED BY BITS 18 AND 19)										STORE (SAME AS ABOVE)				SPECIAL (SAME AS ABOVE)																																																																																																																																																																																																																																																																																																																																																			
		IMM	1110	CMHI	11	CMLO	10	HIGH	01	LOW	00																																																																																																																																																																																																																																																																																																																																																													
WORD TYPE II																																																																																																																																																																																																																																																																																																																																																																								

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	
FIELDS	BRANCH	CONDITION										B R A N C H S E N S E R J S 1	ADDRESS										SPECIAL			
	JMP 1101 JSB 1100 RTN 1111	ALZ 00000 ALO 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	(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 III																										
FIELDS	BRANCH	ADDRESS										MODIFIER/SPECIAL														
WORD TYPE IV	JMP 1101 JSB 1100	Z E R O 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													

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.

MICROASSEMBLER							
FIELD NUMBER	1	2	3	4	5	6	7
BEGINNING COLUMN NUMBER	1	10	15	20	25	30	40
WORD TYPE I				ALU		S-BUS	
WORD TYPE II	L A B E L	O P E R A T I O N	S P E C I A L	MOD.	S T O R E	OPERAND	C O M M E N T S
WORD TYPE III				COND.	BRANCH SENSE	A D D R E S S	
WORD TYPE IV				X	X		

MOD. MEANS MODIFIER
 COND. MEANS CONDITION
 X MEANS MAKE NO ENTRY

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 512₁₀ microword addressing space (except for conditional branches executed in the last location of the current 512₁₀ microword block, which will cause a branch into the next higher 512₁₀ block (target address + 512).) The return

FIELD NUMBER	1	2	3	4	5	6	7	
BEGINNING COLUMN NUMBER	1	10	15	20	25	30	40 72	
FIELDS		<u>OPERATION</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>		
WORD TYPE I		ARS NOP CRS NRM DIV READ ENV RTN ENVE WRTE LGS LWF MPY	ASG JTAB RTN CLFL L1 R1 COV L4 SHLT DCNT MESP SOV FTCH MPCK SRG1 IAK MPP1 SRG2 ICNT MPP2 SRUN INCI NOP STFL IOFF PRST IOG RJ30 ION RPT	ADD NSOL OP11 AND ONE OP13 CMPL OP1 PASL CMPS OP2 PASS DBLS OP3 SANL DEC OP4 SONL INC OP5 SUB IOR OP6 XNOR NAND OP7 XOR NOR OP8 ZERO NSAL OP10	A MPPA S5 B MPPB S6 CAB NOP S7 CNTR P S8 DSPI PNM S9 DSPL S S10 IOO SP S11 IRCM S1 TAB L S2 X M S3 Y MEU S4	A MEU S5 B MPPA S6 CAB MPPB S7 CIR NOP S8 CNTR P S9 DES S S10 DSPI SP S11 DSPL S1 TAB IOI S2 X LDR S3 Y M S4		
FIELDS		<u>OPERATION</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>		
WORD TYPE II		IMM	(SAME AS ABOVE)	CMHI HIGH CMLO LOW	(SAME AS ABOVE)	(DECIMAL OR OCTAL CONSTANT)	COMMENTS	
FIELDS	LABELS	<u>BRANCH</u>	<u>SPECIAL</u>	<u>CONDITION</u>	<u>BRANCH SENSE</u>	<u>ADDRESS</u>		
WORD TYPE III		JMP JSB RTN	CNDX (MUST BE ENTERED)	ALZ L0 NRT AL0 L15 NSFP AL15 MPP NSNG CNT4 MRG NSTB CNT8 NDEC NSTR COUT NINC ONES E NINT OVFL FLAG NLDR RUN HOI NLT RUNE IR8 NMDE SKPF IR11 NMLS	RJS (OR NO ENTRY)	(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).		
FIELDS		<u>BRANCH</u>	<u>MODIFIER/SPECIAL</u>	X		<u>ADDRESS</u>		
WORD TYPE IV		JMP JSB	IOFF NOP IOG RJ30 ION RPT J74 STFL	X		(ANY ADDRESS IN CONTROL MEMORY) *		

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

Figure 4-4. Microassembler Format Micro-Orders

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.

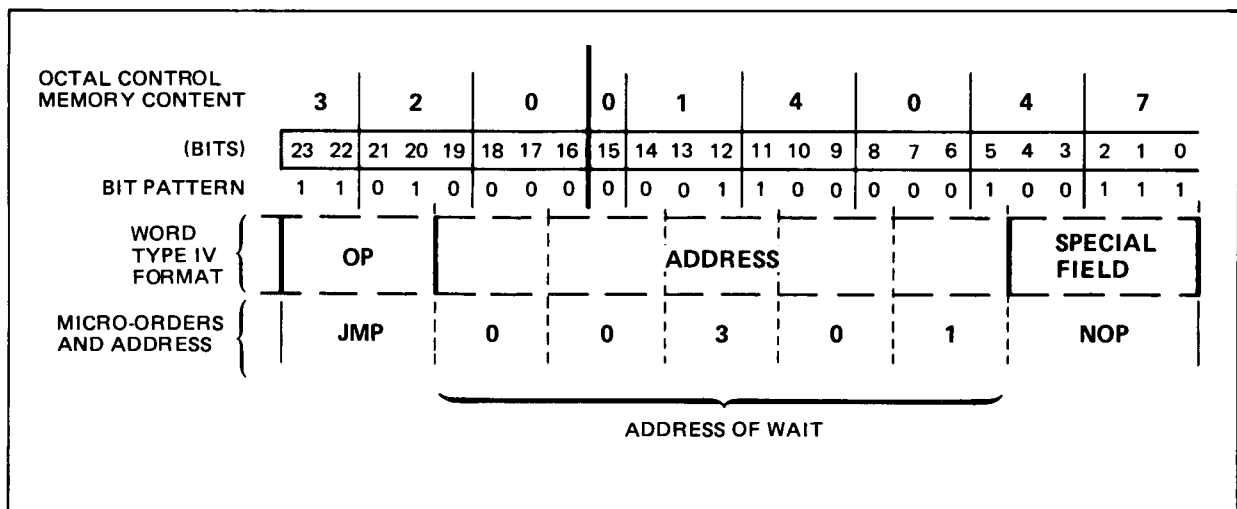


Figure 4-5. Jump Address Decoding

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

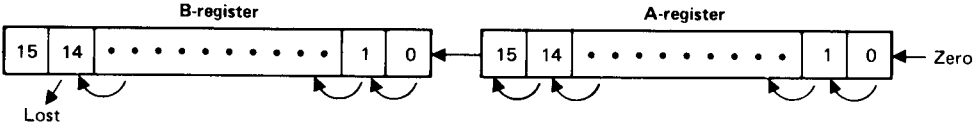
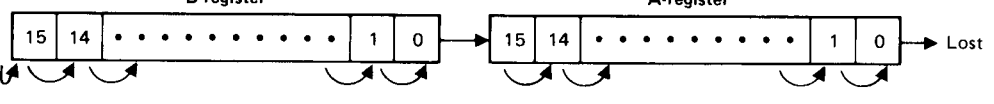
MICRO-ORDER	DEFINITION										
WORD TYPE I OP FIELD											
ARS	<p>Meaning: Perform a single bit arithmetic shift of the A- and B-registers combined, with the A-register forming the low-order 16 bits. The direction of the shift is specified in the Special field: L1 for left, R1 for right.</p> <p>Required micro-order (field) entries:</p> <table border="0" style="width: 100%; text-align: center;"> <tr> <td><u>OP</u></td> <td><u>SPECIAL</u></td> <td><u>ALU</u></td> <td><u>STORE</u></td> <td><u>S-BUS</u></td> </tr> <tr> <td>ARS</td> <td>L1 or R1</td> <td>PASS</td> <td>B</td> <td>B</td> </tr> </table> <p>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.</p> <p style="text-align: center;">ARITHMETIC LEFT SHIFT: SPECIAL = L1</p>  <p>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.</p> <p style="text-align: center;">ARITHMETIC RIGHT SHIFT: SPECIAL = R1</p> 	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	ARS	L1 or R1	PASS	B	B
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>							
ARS	L1 or R1	PASS	B	B							

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

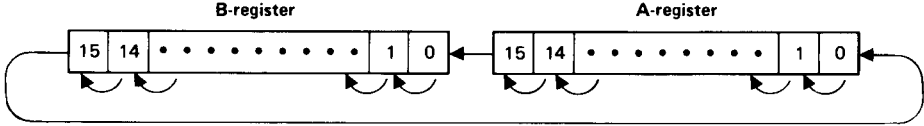
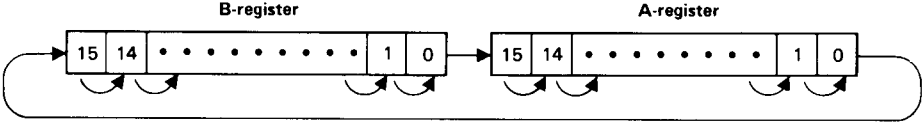
MICRO-ORDER	DEFINITION										
WORD TYPE I - OP FIELD (CONT.)											
CRS	<p>Meaning: Perform a single bit circular rotate/shift on the combined A- and B-registers with the A-register forming the low order 16 bits. The direction of the rotate is specified in the Special field: L1 for left, and R1 for right.</p> <p>Required micro-order (field) entries:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;"><u>OP</u></th> <th style="text-align: center;"><u>SPECIAL</u></th> <th style="text-align: center;"><u>ALU</u></th> <th style="text-align: center;"><u>STORE</u></th> <th style="text-align: center;"><u>S-BUS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">CRS</td> <td style="text-align: center;">L1 or R1</td> <td style="text-align: center;">PASS</td> <td style="text-align: center;">B</td> <td style="text-align: center;">B</td> </tr> </tbody> </table> <p>If the Special field contains L1, bit 15 of the B-register is transferred to bit 0 of the A-register.</p> <p style="text-align: center;">CIRCULAR LEFT SHIFT: SPECIAL = L1</p>  <p>If the Special field contains R1, bit 0 of the A-register is transferred to bit 15 of the B-register.</p> <p style="text-align: center;">CIRCULAR RIGHT SHIFT: SPECIAL = R1</p> 	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	CRS	L1 or R1	PASS	B	B
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>							
CRS	L1 or R1	PASS	B	B							

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

MICRO-ORDER	DEFINITION																												
WORD TYPE I - OP FIELD (CONT.)																													
DIV	<p>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.</p> <p>Required micro-order (field) entries:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;"><u>OP</u></td> <td style="text-align: center;"><u>SPECIAL</u></td> <td style="text-align: center;"><u>ALU</u></td> <td style="text-align: center;"><u>STORE</u></td> <td style="text-align: center;"><u>S-BUS</u></td> </tr> <tr> <td style="text-align: center;">DIV</td> <td style="text-align: center;">L1</td> <td style="text-align: center;">SUB</td> <td style="text-align: center;">B</td> <td style="text-align: center;">B</td> </tr> </table> <p>The divide step is executed as follows:</p> <ol style="list-style-type: none"> 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. <p>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:</p> <p>INITIAL CONTENTS:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center; width: 30%;">B-register</td> <td style="text-align: center; width: 30%;">A-register</td> <td style="text-align: center; width: 30%;">L-register</td> </tr> <tr> <td style="text-align: center; border: 1px solid black; padding: 10px;">Dividend 16 Most Significant bits</td> <td style="text-align: center; border: 1px solid black; padding: 10px;">Dividend 16 Least Significant bits</td> <td style="text-align: center; border: 1px solid black; padding: 10px;">Divisor (Absolute Value)</td> </tr> <tr> <td colspan="3" style="text-align: center;">(Left Shifted)</td> </tr> <tr> <td colspan="3" style="text-align: center;">AFTER REPEAT 16 TIMES OF DIVIDE STEP:</td> </tr> <tr> <td style="text-align: center; width: 30%;">B-register</td> <td style="text-align: center; width: 30%;">A-register</td> <td style="text-align: center; width: 30%;">L-register</td> </tr> <tr> <td style="text-align: center; border: 1px solid black; padding: 10px;">Remainder Doubled</td> <td style="text-align: center; border: 1px solid black; padding: 10px;">16-Bit Quotient of (B, A) / L</td> <td style="text-align: center; border: 1px solid black; padding: 10px;">Divisor (Unchanged)</td> </tr> </table>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	DIV	L1	SUB	B	B	B-register	A-register	L-register	Dividend 16 Most Significant bits	Dividend 16 Least Significant bits	Divisor (Absolute Value)	(Left Shifted)			AFTER REPEAT 16 TIMES OF DIVIDE STEP:			B-register	A-register	L-register	Remainder Doubled	16-Bit Quotient of (B, A) / L	Divisor (Unchanged)
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>																									
DIV	L1	SUB	B	B																									
B-register	A-register	L-register																											
Dividend 16 Most Significant bits	Dividend 16 Least Significant bits	Divisor (Absolute Value)																											
(Left Shifted)																													
AFTER REPEAT 16 TIMES OF DIVIDE STEP:																													
B-register	A-register	L-register																											
Remainder Doubled	16-Bit Quotient of (B, A) / L	Divisor (Unchanged)																											

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

MICRO-ORDER	DEFINITION															
WORD TYPE I - OP FIELD (CONT.)																
ENV	<p>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.</p> <p>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.</p>															
ENVE	<p>Meaning: Enable the overflow and extend logic for the current ALU operation.</p> <p>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).</p> <p>Example:</p> <table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 5px;"><u>OP</u></th> <th style="text-align: center; padding: 5px;"><u>SPECIAL</u></th> <th style="text-align: center; padding: 5px;"><u>ALU</u></th> <th style="text-align: center; padding: 5px;"><u>STORE</u></th> <th style="text-align: center; padding: 5px;"><u>S-BUS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center; padding: 5px;">[ENV]</td> <td></td> <td style="text-align: center; padding: 5px;">ADD</td> <td style="text-align: center; padding: 5px;">S3</td> <td style="text-align: center; padding: 5px;">S3</td> </tr> <tr> <td style="text-align: center; padding: 5px;">[ENVE]</td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> <p>See section 7 information on programmatically setting and clearing the Overflow register.</p>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	[ENV]		ADD	S3	S3	[ENVE]				
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>												
[ENV]		ADD	S3	S3												
[ENVE]																

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

MICRO-ORDER	DEFINITION										
WORD TYPE I - OP FIELD (CONT.)											
LGS	<p>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.</p> <p>Required micro-order (field) entries:</p> <table border="1" data-bbox="487 534 1313 623" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th><u>OP</u></th> <th><u>SPECIAL</u></th> <th><u>ALU</u></th> <th><u>STORE</u></th> <th><u>S-BUS</u></th> </tr> </thead> <tbody> <tr> <td>LGS</td> <td>L1 or R1</td> <td>PASS</td> <td>B</td> <td>B</td> </tr> </tbody> </table> <p>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.</p> <div data-bbox="404 851 1350 975" style="text-align: center;"> <p>LOGICAL LEFT SHIFT: SPECIAL = L1</p> </div> <p>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.</p> <div data-bbox="404 1390 1350 1514" style="text-align: center;"> <p>LOGICAL RIGHT SHIFT: SPECIAL = R1</p> </div>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	LGS	L1 or R1	PASS	B	B
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>							
LGS	L1 or R1	PASS	B	B							

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

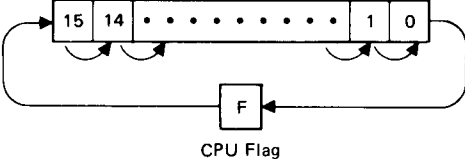
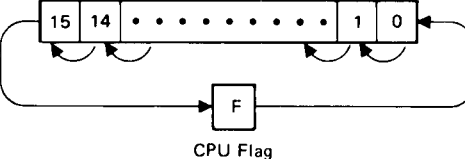
MICRO-ORDER	DEFINITION										
WORD TYPE I - OP FIELD (CONT.)											
LWF	<p>Meaning: Perform a one bit rotational shift of a 17-bit operand in the Rotate/Shifter where bit 17 is formed by the CPU flag (link with flag). The data rotates left one bit if L1 is in the Special field, or right one bit if R1 is in the Special field. If neither L1 or R1 are specified, LWF clears the CPU flag and no rotate takes place.</p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>ROTATIONAL RIGHT SHIFT: SPECIAL = R1</p>  <p>ALU Contents</p> <p>CPU Flag</p> </div> <div style="text-align: center;"> <p>ROTATIONAL LEFT SHIFT: SPECIAL = L1</p>  <p>ALU Contents</p> <p>CPU Flag</p> </div> </div>										
MPY	<p>Meaning: Perform a multiply step where the multiplier is in the L-register and the multiplicand is in the A-register.</p> <p>Required micro-order (field) entries:</p> <table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 5px;"><u>OP</u></th> <th style="text-align: center; padding: 5px;"><u>SPECIAL</u></th> <th style="text-align: center; padding: 5px;"><u>ALU</u></th> <th style="text-align: center; padding: 5px;"><u>STORE</u></th> <th style="text-align: center; padding: 5px;"><u>S-BUS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center; padding: 5px;">MPY</td> <td style="text-align: center; padding: 5px;">R1</td> <td style="text-align: center; padding: 5px;">ADD</td> <td style="text-align: center; padding: 5px;">B</td> <td style="text-align: center; padding: 5px;">B</td> </tr> </tbody> </table> <p>The multiply step is executed as follows:</p> <ol style="list-style-type: none"> 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. <p>Usage: This microinstruction is usually repeated 16 times by specifying the Special field micro-order RPT in the preceding microinstruction.</p> <p>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.</p> <p>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).</p>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	MPY	R1	ADD	B	B
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>							
MPY	R1	ADD	B	B							

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

MICRO-ORDER	DEFINITION						
WORD TYPE I - OP FIELD (Cont.)							
<p>MPY (Continued)</p>	<p>$(B,A) = [(AxL) + B]$</p> <p>This may be useful in some computational procedures. For example: $X(2) = X(1) + (YxZ)$.</p> <p>Initial Contents:</p> <p>INITIAL CONTENTS:</p> <table style="width: 100%; text-align: center;"> <tr> <td data-bbox="404 696 671 874"> <p>B-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Value to be added to the final result</p> </div> </td> <td data-bbox="735 731 1003 874"> <p>A-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplicand</p> </div> </td> <td data-bbox="1070 731 1339 874"> <p>L-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplier</p> </div> </td> </tr> </table> <p>AFTER REPEATING THE MULTIPLY STEP 16 TIMES:</p> <table style="width: 100%; text-align: center;"> <tr> <td data-bbox="398 1116 666 1259"> <p>B-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>$(AxL) + B$ 16 Most Significant bits</p> </div> </td> <td data-bbox="730 1116 998 1259"> <p>A-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>$(AxL) + B$ 16 Least Significant bits</p> </div> </td> <td data-bbox="1066 1123 1334 1265"> <p>L-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplier (Unchanged)</p> </div> </td> </tr> </table>	<p>B-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Value to be added to the final result</p> </div>	<p>A-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplicand</p> </div>	<p>L-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplier</p> </div>	<p>B-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>$(AxL) + B$ 16 Most Significant bits</p> </div>	<p>A-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>$(AxL) + B$ 16 Least Significant bits</p> </div>	<p>L-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplier (Unchanged)</p> </div>
<p>B-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Value to be added to the final result</p> </div>	<p>A-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplicand</p> </div>	<p>L-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplier</p> </div>					
<p>B-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>$(AxL) + B$ 16 Most Significant bits</p> </div>	<p>A-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>$(AxL) + B$ 16 Least Significant bits</p> </div>	<p>L-register</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Multiplier (Unchanged)</p> </div>					

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

MICRO-ORDER	DEFINITION																				
WORD TYPE I - OP FIELD (CONT.)																					
NOP	<p>Meaning: No operation is specified for the OP field.</p> <p>Usage: This is the default micro-order when the OP field is left blank.</p>																				
NRM	<p>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.</p> <p>Left shift: The left normalizing shift requires that the following micro-orders be used:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;"><u>OP</u></th> <th style="text-align: center;"><u>SPECIAL</u></th> <th style="text-align: center;"><u>ALU</u></th> <th style="text-align: center;"><u>STORE</u></th> <th style="text-align: center;"><u>S-BUS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">NRM</td> <td style="text-align: center;">L1</td> <td style="text-align: center;">PASS</td> <td style="text-align: center;">*</td> <td style="text-align: center;">*</td> </tr> </tbody> </table> <p>*Desired Register</p> <p>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.)</p> <div style="text-align: center; margin: 10px 0;"> </div> <p>Right shift: The right normalizing shift requires that the following micro-orders be used:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;"><u>OP</u></th> <th style="text-align: center;"><u>SPECIAL</u></th> <th style="text-align: center;"><u>ALU</u></th> <th style="text-align: center;"><u>STORE</u></th> <th style="text-align: center;"><u>S-BUS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">NRM</td> <td style="text-align: center;">R1</td> <td style="text-align: center;">PASS</td> <td style="text-align: center;">*</td> <td style="text-align: center;">*</td> </tr> </tbody> </table> <p>*Desired Register</p> <p>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).</p> <div style="text-align: center; margin: 10px 0;"> </div>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	NRM	L1	PASS	*	*	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	NRM	R1	PASS	*	*
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>																	
NRM	L1	PASS	*	*																	
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>																	
NRM	R1	PASS	*	*																	

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

MICRO-ORDER	DEFINITION																																		
WORD TYPE I - OP FIELD (Cont.)																																			
<p>NRM (Continued)</p>	<p>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.</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;"><u>OP</u></th> <th style="text-align: center;"><u>SPECIAL</u></th> <th style="text-align: center;"><u>ALU</u></th> <th style="text-align: center;"><u>STORE</u></th> <th style="text-align: center;"><u>S-BUS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">NRM</td> <td style="text-align: center;">R1</td> <td style="text-align: center;">PASS</td> <td style="text-align: center;">S1</td> <td style="text-align: center;">S1</td> </tr> </tbody> </table> <p>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:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;"><u>LABEL</u></th> <th style="text-align: center;"><u>OP</u></th> <th style="text-align: center;"><u>SPECIAL</u></th> <th style="text-align: center;"><u>ALU/ COND.</u></th> <th style="text-align: center;"><u>STORE</u></th> <th style="text-align: center;"><u>S-BUS- ADDRESS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">NRM48</td> <td style="text-align: center;">IMM</td> <td></td> <td style="text-align: center;">LOW DBLS XOR</td> <td style="text-align: center;">CNTR L</td> <td style="text-align: center;">0 B B</td> </tr> <tr> <td></td> <td style="text-align: center;">JMP</td> <td style="text-align: center;">CNDX RPT</td> <td style="text-align: center;">AL15</td> <td></td> <td style="text-align: center;">*+ 4</td> </tr> <tr> <td></td> <td style="text-align: center;">NRM JMP</td> <td style="text-align: center;">L1</td> <td style="text-align: center;">PASS</td> <td style="text-align: center;">S1</td> <td style="text-align: center;">S1 NRM48+ 1</td> </tr> </tbody> </table> <p>Upon exit, the number is normalized and the counter contains the two's complement of the number of shifts performed.</p> <p style="text-align: center;">NOTE</p> <p>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.</p>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	NRM	R1	PASS	S1	S1	<u>LABEL</u>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU/ COND.</u>	<u>STORE</u>	<u>S-BUS- ADDRESS</u>	NRM48	IMM		LOW DBLS XOR	CNTR L	0 B B		JMP	CNDX RPT	AL15		*+ 4		NRM JMP	L1	PASS	S1	S1 NRM48+ 1
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>																															
NRM	R1	PASS	S1	S1																															
<u>LABEL</u>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU/ COND.</u>	<u>STORE</u>	<u>S-BUS- ADDRESS</u>																														
NRM48	IMM		LOW DBLS XOR	CNTR L	0 B B																														
	JMP	CNDX RPT	AL15		*+ 4																														
	NRM JMP	L1	PASS	S1	S1 NRM48+ 1																														

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

MICRO-ORDER	DEFINITION
WORD TYPE I OP FIELD (CONT.)	
READ	<p>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.</p> <p>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.</p>
RTN	<p>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).</p> <p>Usage: For word type I, CNDX is <i>not allowed</i> in the Special field so the "pop" operation and branch are unconditionally made.</p>
WRTE	<p>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.</p> <p>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</p>

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

MICRO-ORDER	DEFINITION															
WORD TYPE I AND II - SPECIAL FIELD																
ASG	<p>Meaning: Bits 6 and 7 of the Instruction Register (IR) determine which of the following functions are to be performed:</p> <table border="1" data-bbox="545 418 1146 741"> <thead> <tr> <th>IR</th> <th>bit</th> <th>Alter/Skip Group</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>6</td> <td>Instruction</td> </tr> <tr> <td>0</td> <td>1</td> <td>(CLE) Clear Extend register</td> </tr> <tr> <td>1</td> <td>0</td> <td>(CME) Complement Extend register</td> </tr> <tr> <td>1</td> <td>1</td> <td>(CCE) Set Extend register</td> </tr> </tbody> </table> <p>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 <i>not</i> "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.</p> <p>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.</p>	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
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														
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	<p>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.</p> <p>Usage: Not normally used by the user microprogrammer. Refer to section 7 for interrupt handling techniques.</p>															
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	<p>Meaning: Increment the Indirect Counter in Memory Protect (if installed) by one.</p> <p>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.</p>															

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

MICRO-ORDER	DEFINITION
WORD TYPE I AND II - SPECIAL FIELD (CONT.)	
IOFF	<p>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).</p> <p>Usage: After the occurrence of a JTAB or three occurrences of INCI (if Memory Protect is installed) interrupts are again recognized.</p> <p>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.</p>
IOG	<p>Meaning: Freeze the CPU until time period T2. Then enable the generation of I/O timing signals dependent upon the instruction in the IR.</p> <p>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.</p>
ION	<p>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.</p> <p>Usage: An interrupt from any I/O device can be detected in two ways:</p> <ol style="list-style-type: none"> 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 <i>Computer Series Operating and Reference Manual</i>. Refer to section 7 of this manual for interrupt handling considerations.
JTAB	<p>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.</p>
L1	<p>Meaning: Left shift one bit command to the Rotate/Shifter.</p> <div style="text-align: center;"> </div> <p>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.</p>

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

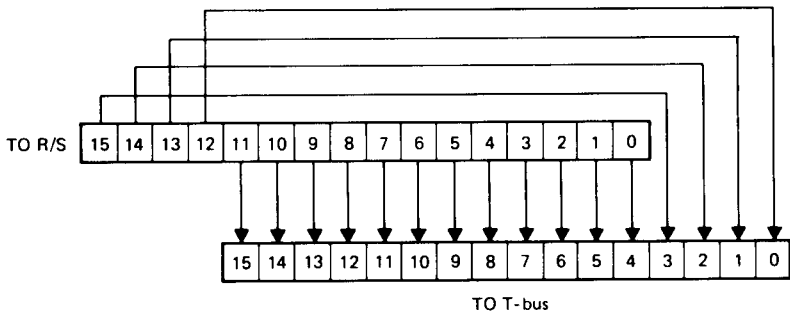
MICRO-ORDER	DEFINITION										
WORD TYPE I AND II - SPECIAL FIELD (CONT.)											
L4	<p>Meaning: Four bit left rotate command to the Rotate/Shifter.</p> 										
MESP	<p>Meaning: Dynamic Mapping System (DMS) signal generation micro-order used in conjunction with the MEU micro-order in the Store 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.</p> <p>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.</p>										
MPCK	<p>Meaning: Request a Memory Protect check of the address in the M-register for a Memory Protect fence or DMS violation.</p> <p>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:</p> <ol style="list-style-type: none"> 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 <i>must</i> follow the microinstruction containing a READ or WRTE by one or two microinstructions. The MPCK <i>must never</i> 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. <table border="0" style="width: 100%; text-align: center;"> <tr> <td><u>OP</u></td> <td><u>SPECIAL</u></td> <td><u>ALU</u></td> <td><u>STORE</u></td> <td><u>S-BUS</u></td> </tr> <tr> <td>WRTE</td> <td>MPCK</td> <td>PASS</td> <td>TAB</td> <td>S1</td> </tr> </table>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	WRTE	MPCK	PASS	TAB	S1
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>							
WRTE	MPCK	PASS	TAB	S1							

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

MICRO-ORDER	DEFINITION
WORD TYPE I AND II - SPECIAL FIELD (CONT.)	
MPP1	<p>Meaning: Generate a signal $\overline{PP1SP}$ to the Microprogrammable Processor Port (MPP).</p> <p>Usage: Refer to the <i>HP 21MX/21MX E-Series Computer I/O Interfacing Guide</i> for further information. Example microprogrammed use can be found in section 13 of this manual.</p>
MPP2	<p>Meaning: Generate a signal PP2SP to the MPP.</p> <p>Usage: Refer to the <i>HP 21MX/21MX E-Series Computer I/O Interfacing Guide</i> for further information. Example microprogrammed use can be found in section 13 of this manual.</p>
NOP	<p>Meaning: No operation in the Special field.</p> <p>Usage: This is the default operation if no other micro-order is specified in the Special field.</p>
PRST	<p>Meaning: This micro-order will clear the A- and B-Addressable flip-flops (AAF and BAF).</p> <p>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.</p>
RJ30	<p>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).</p>
RPT	<p>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.</p> <p>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.</p>
RTN	<p>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).</p> <p>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.</p>
R1	<p>Meaning: Right shift one bit command to the Rotate/Shifter.</p> <div style="text-align: center;"> </div> <p>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.</p>

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

MICRO-ORDER	DEFINITION																						
WORD TYPE I AND II SPECIAL FIELD (CONT.)																							
SHLT	<p>Meaning: Clear the Run flip-flop.</p> <p>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.</p>																						
SOV	<p>Meaning: Set the Overflow register. Refer to section 7 for information on programmatically clearing and setting the Overflow register.</p>																						
SRG1	<p>Meaning: Execute the shift/rotate function specified by bits 6 through 9 of the IR. (Refer to your <i>Computer Series Operating and Reference Manual</i>.) 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:</p> <table border="0" data-bbox="478 745 1081 1419"> <thead> <tr> <th data-bbox="478 745 556 777" style="text-align: center;">BITS</th> <th data-bbox="588 777 1081 808" style="text-align: center;"><u>FUNCTION PERFORMED IN ROTATE/SHIFTER</u></th> </tr> <tr> <th data-bbox="478 777 556 808" style="text-align: center;"><u>9 8 7 6</u></th> <th></th> </tr> </thead> <tbody> <tr> <td data-bbox="486 828 548 859">1000</td> <td data-bbox="588 828 1081 859">Arithmetic left shift one bit.</td> </tr> <tr> <td data-bbox="486 880 548 911">1001</td> <td data-bbox="588 880 1081 911">Arithmetic right shift one bit.</td> </tr> <tr> <td data-bbox="486 932 548 963">1010</td> <td data-bbox="588 932 1081 963">Rotational left shift one bit.</td> </tr> <tr> <td data-bbox="486 984 548 1015">1011</td> <td data-bbox="588 984 1081 1015">Rotational right shift one bit.</td> </tr> <tr> <td data-bbox="486 1036 548 1067">1100</td> <td data-bbox="588 1036 1081 1067">Arithmetic left shift one bit, clear sign (bit 15).</td> </tr> <tr> <td data-bbox="486 1087 548 1118">1101</td> <td data-bbox="588 1087 1081 1149">Rotational right shift one bit with E-register forming bit 16 (17th bit).</td> </tr> <tr> <td data-bbox="486 1170 548 1201">1110</td> <td data-bbox="588 1170 1081 1232">Rotational left shift one bit with E-register forming bit 16 (the 17th bit).</td> </tr> <tr> <td data-bbox="486 1253 548 1284">1111</td> <td data-bbox="588 1253 1081 1284">Rotational left shift four bits.</td> </tr> <tr> <td data-bbox="486 1305 548 1336">0xxx</td> <td data-bbox="588 1305 1081 1419">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 <i>Computer Series Operating and Reference Manual</i> Shift/Rotate Group information for instructions on how to avoid this situation.)</td> </tr> </tbody> </table> <p data-bbox="392 1491 1097 1522">Usage: Refer to section 7 for considerations when using SRG1.</p>	BITS	<u>FUNCTION PERFORMED IN ROTATE/SHIFTER</u>	<u>9 8 7 6</u>		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 <i>Computer Series Operating and Reference Manual</i> Shift/Rotate Group information for instructions on how to avoid this situation.)
BITS	<u>FUNCTION PERFORMED IN ROTATE/SHIFTER</u>																						
<u>9 8 7 6</u>																							
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 <i>Computer Series Operating and Reference Manual</i> Shift/Rotate Group information for instructions on how to avoid this situation.)																						

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

MICRO-ORDER	DEFINITION																						
WORD TYPE I AND IISPECIAL FIELD (CONT.)																							
SRG2	<p>Meaning: Execute the shift/rotate function specified by bits 0, 1, 2, and 4 of the IR. (Refer to your <i>Computer Series Operating and Reference Manual</i>.) 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:</p> <table border="0" data-bbox="454 528 1324 1191"> <thead> <tr> <th style="text-align: center;">BITS</th> <th style="text-align: center;">FUNCTION PERFORMED IN ROTATE/SHIFTER</th> </tr> <tr> <th style="text-align: center;"><u>4 2 1 0</u></th> <th></th> </tr> </thead> <tbody> <tr> <td>1 0 0 0</td> <td>Arithmetic left shift one bit.</td> </tr> <tr> <td>1 0 0 1</td> <td>Arithmetic right shift one bit.</td> </tr> <tr> <td>1 0 1 0</td> <td>Rotational left shift one bit.</td> </tr> <tr> <td>1 0 1 1</td> <td>Rotational right shift one bit.</td> </tr> <tr> <td>1 1 0 0</td> <td>Arithmetic left shift one bit, clear sign (bit 15).</td> </tr> <tr> <td>1 1 0 1</td> <td>Rotational right shift one bit with E-register forming bit 16 (the 17th bit).</td> </tr> <tr> <td>1 1 1 0</td> <td>Rotational left shift one bit with E-register forming bit 16 (the 17th bit).</td> </tr> <tr> <td>1 1 1 1</td> <td>Rotational left shift four bits.</td> </tr> <tr> <td>0 x x x</td> <td>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 <i>Computer Series Operating and Reference Manual</i> Shift/Rotate Group information for instructions on how to avoid this situation.)</td> </tr> </tbody> </table> <p>Usage: Refer to section 7 for considerations when using SRG2.</p>	BITS	FUNCTION PERFORMED IN ROTATE/SHIFTER	<u>4 2 1 0</u>		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 <i>Computer Series Operating and Reference Manual</i> Shift/Rotate Group information for instructions on how to avoid this situation.)
BITS	FUNCTION PERFORMED IN ROTATE/SHIFTER																						
<u>4 2 1 0</u>																							
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 <i>Computer Series Operating and Reference Manual</i> Shift/Rotate Group information for instructions on how to avoid this situation.)																						
SRUN	<p>Meaning: Set the Run flip-flop.</p> <p>Usage: The RUN condition is not actually set until the next word type I or II is executed.</p>																						
STFL	<p>Meaning: Set the CPU flag.</p>																						

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

MICRO-ORDER	DEFINITION
WORD TYPE I ALU FIELD	
	<p style="text-align: center;">NOTE</p> <p>Symbols used in the following ALU field equations are defined here for reference.</p> <p>+ means arithmetic function +</p> <p>– means arithmetic function –</p> <p>• means logical function “and”.</p> <p>+ means logical function “or”.</p> <p>⊕ means logical function “exclusive or”.</p> <p>\bar{S} or \bar{L} means the one’s complement of the S-bus or the one’s complement of the L-register.</p>
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: $(L \cdot S)$.
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: $(S \cdot L)$. 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 $(S \cdot L)$.
OP7	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: $(S + \bar{L})$ plus $(S \cdot L)$.
OP8	Meaning: Perform the following logical function in the ALU with the L-register and S-bus: $(S \cdot L)$ 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: $(S + \bar{L})$ 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 $(S \cdot \bar{L})$; 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 $(S + \bar{L})$; 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 $(\bar{L} \oplus S)$; 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	<p>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 <i>Computer Series Operating and Reference Manual</i> 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:</p> <table border="1" data-bbox="482 996 1224 1210" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Display Indicator (S- bus) bit</th> <th>7</th> <th>6</th> <th>5</th> <th>4</th> <th>3</th> <th>2</th> <th>1</th> <th>0</th> </tr> </thead> <tbody> <tr> <td>Register Displayed in Normal Mode</td> <td>-</td> <td>-</td> <td>S</td> <td>P</td> <td>T</td> <td>M</td> <td>B</td> <td>A</td> </tr> <tr> <td>Function Displayed in Special Mode</td> <td>-</td> <td>-</td> <td>s</td> <td>f</td> <td>t</td> <td>m</td> <td>y</td> <td>x</td> </tr> </tbody> </table> <p>NOTE: Bits 7 and 6 not used.</p> <p>Usage: The Operator Panel Display Indicator or Indicators can be lit by bits 5 through 0 from the S-bus as follows:</p> <table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th><u>OP</u></th> <th><u>SPECIAL</u></th> <th><u>MOD.</u></th> <th><u>STORE</u></th> <th><u>OPERAND</u></th> </tr> </thead> <tbody> <tr> <td>IMM</td> <td></td> <td>LOW</td> <td>DSPI</td> <td>373B</td> </tr> </tbody> </table> <p style="text-align: center;">Lights indicator pointing to M-register.</p> <p>whereas:</p> <table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th><u>OP</u></th> <th><u>SPECIAL</u></th> <th><u>MOD.</u></th> <th><u>STORE</u></th> <th><u>OPERAND</u></th> </tr> </thead> <tbody> <tr> <td>IMM</td> <td></td> <td>LOW</td> <td>DSPI</td> <td>010B</td> </tr> </tbody> </table> <p>Lights all indicators (Special mode) except the function "t" mode (i.e., indicates that DMS map content is displayed in the Display Register).</p>	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	<u>OP</u>	<u>SPECIAL</u>	<u>MOD.</u>	<u>STORE</u>	<u>OPERAND</u>	IMM		LOW	DSPI	373B	<u>OP</u>	<u>SPECIAL</u>	<u>MOD.</u>	<u>STORE</u>	<u>OPERAND</u>	IMM		LOW	DSPI	010B
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																																								
<u>OP</u>	<u>SPECIAL</u>	<u>MOD.</u>	<u>STORE</u>	<u>OPERAND</u>																																												
IMM		LOW	DSPI	373B																																												
<u>OP</u>	<u>SPECIAL</u>	<u>MOD.</u>	<u>STORE</u>	<u>OPERAND</u>																																												
IMM		LOW	DSPI	010B																																												

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

MICRO-ORDER	DEFINITION
WORD TYPE I AND II - STORE FIELD (CONT.)	
DSPL	<p>Meaning: Store the data on the S-bus in the Operator Panel Display Register. This information should be coordinated with the Display Indicator.</p>
IOO	<p>Meaning: Enable the S-bus onto the I/O bus.</p> <p>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.</p>
IRCM	<p>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.</p> <p>Usage: Refer to section 7 for information on interfacing with Memory Protect.</p>
L	<p>Meaning: Store the data at the output of the ALU into the L-register.</p> <p>Usage: The L-register is used as the second operand in arithmetic functions.</p>
M	<p>Meaning: Store the data on the S-bus in the M-register.</p> <p>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.</p>
MEU	<p>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.</p> <p>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.</p>
MPPA and MPPB	<p>Meaning: Generate the signals <u>MPPAST</u> and MPBST to the MPP.</p> <p>Usage: Refer to the <i>HP 21MX/21MX E-Series Computer I/O Interfacing Guide</i> for further information. Example microprogram use can be found in section 13 of this manual.</p>
NOP	<p>Meaning: No store operation is performed; this is the default micro-order when the Store field is left blank.</p>
P	<p>Meaning: Store the data on the T-bus in the P-register (Program Counter).</p>

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

MICRO-ORDER	DEFINITION																		
WORD TYPE I AND II - STORE FIELD (CONT.)																			
PNM	<p>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.</p> <p>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:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;"><u>OP</u></td> <td style="text-align: center;"><u>SPECIAL</u></td> <td style="text-align: center;"><u>ALU</u></td> <td style="text-align: center;"><u>STORE</u></td> <td style="text-align: center;"><u>S-BUS</u></td> </tr> <tr> <td style="text-align: center;">READ</td> <td></td> <td style="text-align: center;">INC</td> <td style="text-align: center;">PNM</td> <td style="text-align: center;">P</td> </tr> </table> <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.</p>	<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>	READ		INC	PNM	P								
<u>OP</u>	<u>SPECIAL</u>	<u>ALU</u>	<u>STORE</u>	<u>S-BUS</u>															
READ		INC	PNM	P															
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	<p>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:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th rowspan="2">M-bus address when M-register store is specified</th> <th colspan="2">FF States</th> <th rowspan="2">Register referenced by TAB in store (or S-bus) field.</th> </tr> <tr> <th>AAF</th> <th>BAF</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">A</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">B</td> </tr> <tr> <td style="text-align: center;">Any other value</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">T</td> </tr> </tbody> </table> <p>Note that the PRST micro-order clears the AAF and BAF flip-flops.</p> <p>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.</p>	M-bus address when M-register store is specified	FF States		Register referenced by TAB in store (or S-bus) field.	AAF	BAF	0	1	0	A	1	0	1	B	Any other value	0	0	T
M-bus address when M-register store is specified	FF States		Register referenced by TAB in store (or S-bus) field.																
	AAF	BAF																	
0	1	0	A																
1	0	1	B																
Any other value	0	0	T																
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. <table border="1" data-bbox="592 1031 1193 1120" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 2px;">Switch No.</td> <td style="padding: 2px;">8</td> <td style="padding: 2px;">7</td> <td style="padding: 2px;">6</td> <td style="padding: 2px;">5</td> <td style="padding: 2px;">4</td> <td style="padding: 2px;">3</td> <td style="padding: 2px;">2</td> <td style="padding: 2px;">1</td> </tr> <tr> <td style="padding: 2px;">S-Bus bit</td> <td style="padding: 2px;">15</td> <td style="padding: 2px;">14</td> <td style="padding: 2px;">10</td> <td style="padding: 2px;">9</td> <td style="padding: 2px;">8</td> <td style="padding: 2px;">7</td> <td style="padding: 2px;">6</td> <td style="padding: 2px;">0</td> </tr> </table> Undriven S-bus bits are logical ones. Usage: Used in the base set microprogrammed bootstrap routine. Refer to your <i>Computer Series Operating and Reference Manual</i> 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.	Switch No.	8	7	6	5	4	3	2	1	S-Bus bit	15	14	10	9	8	7	6	0
Switch No.	8	7	6	5	4	3	2	1											
S-Bus bit	15	14	10	9	8	7	6	0											
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	<p>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:</p> <p>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.</p>
M	<p>Meaning: Place the 15-bit contents of the M-register on the S-bus. Bit 15 of the S-bus is zero.</p>
MEU	<p>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 Q₀ through Q₇ for reference) by combinations of MESP and MEU. The combinations of these signals and their functions are described in section 7.</p> <p>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.</p>

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

MICRO-ORDER	DEFINITION
WORD TYPE I - S-BUS FIELD (CONT.)	
MPPA and MPPB	<p>Meaning: Generate signals <u>MPAEN</u> and MPBEN. (MPAEN is not used.)</p> <p>Usage: Refer to the <i>HP 21MX M-Series and E-Series Computers I/O Interfacing Guide</i> for further information. Example microprogram use can be found in section 13 of this manual.</p>
NOP	<p>Meaning: All ones are on the S-bus.</p> <p>Usage: This is the default micro-order when the S-bus field is not specified in a microinstruction.</p>
P	<p>Meaning: Place the content of the P-register on the S-bus.</p>
S	<p>Meaning: Place the content of the S-register on the S-bus.</p>
SP	<p>Meaning: Place the contents of the SP-register on the S-bus.</p>
S1 thru S11	<p>Meaning: Place the contents of the indicated Scratch Register (S1 through S11) on the S-bus.</p>
TAB	<p>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.</p> <p>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.</p>
X	<p>Meaning: Place the contents of the X-register on the S-bus.</p>
Y	<p>Meaning: Place the contents of the Y-register on the S-bus.</p>
WORD TYPE II - OP FIELD	
IMM	<p>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.</p> <p>Usage: Refer to the word type II Modifier field micro-orders for Operand examples.</p>
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																																																																															
<p>CMHI</p>	<p>Meaning: The 16 bits received by the S-bus consist of the following:</p> <p>Bits 15 through 8 = Operand. (Refer to the information on word type II Operand.)</p> <p>Bits 7 through 0 = all ones.</p> <p>The S-bus data is then complemented as it passes through the ALU.</p> <p>Usage: See below.</p> <p>MICROINSTRUCTION:</p> <table border="0" style="margin-left: 40px;"> <tr> <td style="text-align: center;"><u>OP</u></td> <td style="text-align: center;"><u>SPECIAL</u></td> <td style="text-align: center;"><u>MODIFIER</u></td> <td style="text-align: center;"><u>STORE</u></td> <td style="text-align: center;"><u>OPERAND</u></td> </tr> <tr> <td style="text-align: center;">IMM</td> <td></td> <td style="text-align: center;">CMHI</td> <td style="text-align: center;">L</td> <td style="text-align: center;">367B</td> </tr> </table> <p>S-bus</p> <table border="1" style="margin-left: 40px;"> <tr> <td style="text-align: center;">BIT NO.</td> <td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">CONTENT</td> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td> </tr> </table> <p style="text-align: center; margin-left: 40px;">OPERAND (367B)</p> <p>Result Out of ALU</p> <table border="1" style="margin-left: 40px;"> <tr> <td style="text-align: center;">BIT NO.</td> <td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">CONTENT</td> <td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> </table> <p style="text-align: center; margin-left: 40px;">OPERAND Complemented</p>	<u>OP</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>	IMM		CMHI	L	367B	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	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
<u>OP</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>																																																																											
IMM		CMHI	L	367B																																																																											
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																																																															
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																																																															
<p>CMLO</p>	<p>Meaning: The 16 bits received by the S-bus consist of the following:</p> <p>Bits 15 through 8 = all ones.</p> <p>Bits 7 through 0 = Operand. (Refer to the information on word type II Operand.)</p> <p>The S-bus data is then complemented as it passes through the ALU.</p> <p>Usage: See below.</p> <p>MICROINSTRUCTION:</p> <table border="0" style="margin-left: 40px;"> <tr> <td style="text-align: center;"><u>OP</u></td> <td style="text-align: center;"><u>SPECIAL</u></td> <td style="text-align: center;"><u>MODIFIER</u></td> <td style="text-align: center;"><u>STORE</u></td> <td style="text-align: center;"><u>OPERAND</u></td> </tr> <tr> <td style="text-align: center;">IMM</td> <td></td> <td style="text-align: center;">CMLO</td> <td style="text-align: center;">S2</td> <td style="text-align: center;">020B</td> </tr> </table> <p>S-bus</p> <table border="1" style="margin-left: 40px;"> <tr> <td style="text-align: center;">BIT NO.</td> <td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">CONTENT</td> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> </table> <p style="text-align: center; margin-left: 40px;">OPERAND</p> <p>Result Out of ALU</p> <table border="1" style="margin-left: 40px;"> <tr> <td style="text-align: center;">BIT NO.</td> <td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">CONTENT</td> <td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td> </tr> </table> <p style="text-align: center; margin-left: 40px;">OPERAND Complemented</p>	<u>OP</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>	IMM		CMLO	S2	020B	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	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
<u>OP</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>																																																																											
IMM		CMLO	S2	020B																																																																											
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																																																															
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																																																															

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

MICRO-ORDER	DEFINITION																																												
WORD TYPE II - MODIFIER FIELD (CONT.)																																													
HIGH	<p>Meaning: The 16 bits received by the S-bus consist of the following:</p> <p style="padding-left: 40px;">Bits 15 through 8 = Operand. (Refer to the information on word type II Operand.)</p> <p style="padding-left: 40px;">Bits 7 through 0 = all ones.</p> <p>The S-bus data is then passed through the ALU without modification.</p> <p>Usage: See below.</p> <p>MICROINSTRUCTION:</p> <table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 2px;"><u>OP</u></td> <td style="text-align: center; padding: 2px;"><u>SPECIAL</u></td> <td style="text-align: center; padding: 2px;"><u>MODIFIER</u></td> <td style="text-align: center; padding: 2px;"><u>STORE</u></td> <td style="text-align: center; padding: 2px;"><u>OPERAND</u></td> </tr> <tr> <td style="text-align: center; padding: 2px;">IMM</td> <td></td> <td style="text-align: center; padding: 2px;">HIGH</td> <td style="text-align: center; padding: 2px;">S5</td> <td style="text-align: center; padding: 2px;">232B</td> </tr> </table> <p>S-bus and Result Out of ALU {</p> <table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 2px;">BIT NO.</td> <td style="text-align: center; padding: 2px;">15</td> <td style="text-align: center; padding: 2px;">14</td> <td style="text-align: center; padding: 2px;">13</td> <td style="text-align: center; padding: 2px;">12</td> <td style="text-align: center; padding: 2px;">11</td> <td style="text-align: center; padding: 2px;">10</td> <td style="text-align: center; padding: 2px;">9</td> <td style="text-align: center; padding: 2px;">8</td> <td style="text-align: center; padding: 2px;">7</td> <td style="text-align: center; padding: 2px;">6</td> <td style="text-align: center; padding: 2px;">5</td> <td style="text-align: center; padding: 2px;">4</td> <td style="text-align: center; padding: 2px;">3</td> <td style="text-align: center; padding: 2px;">2</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">0</td> </tr> <tr> <td style="text-align: center; padding: 2px;">CONTENT</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> </tr> </table> <p style="text-align: center; margin-left: 40px;">} <u>OPERAND</u></p>	<u>OP</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>	IMM		HIGH	S5	232B	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
<u>OP</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>																																									
IMM		HIGH	S5	232B																																									
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																													
LOW	<p>Meaning: The 16 bits received by the S-bus consist of the following:</p> <p style="padding-left: 40px;">Bits 15 through 8 = all ones.</p> <p style="padding-left: 40px;">Bits 7 through 0 = Operand. (Refer to the information on the word type II Operand.)</p> <p>The S-bus data is then passed through the ALU without modification.</p> <p>Usage: See below.</p> <p>MICROINSTRUCTION:</p> <table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 2px;"><u>OP</u></td> <td style="text-align: center; padding: 2px;"><u>SPECIAL</u></td> <td style="text-align: center; padding: 2px;"><u>MODIFIER</u></td> <td style="text-align: center; padding: 2px;"><u>STORE</u></td> <td style="text-align: center; padding: 2px;"><u>OPERAND</u></td> </tr> <tr> <td style="text-align: center; padding: 2px;">IMM</td> <td></td> <td style="text-align: center; padding: 2px;">LOW</td> <td style="text-align: center; padding: 2px;">S11</td> <td style="text-align: center; padding: 2px;">111B</td> </tr> </table> <p>S-bus and Result Out of ALU {</p> <table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 2px;">BIT NO.</td> <td style="text-align: center; padding: 2px;">15</td> <td style="text-align: center; padding: 2px;">14</td> <td style="text-align: center; padding: 2px;">13</td> <td style="text-align: center; padding: 2px;">12</td> <td style="text-align: center; padding: 2px;">11</td> <td style="text-align: center; padding: 2px;">10</td> <td style="text-align: center; padding: 2px;">9</td> <td style="text-align: center; padding: 2px;">8</td> <td style="text-align: center; padding: 2px;">7</td> <td style="text-align: center; padding: 2px;">6</td> <td style="text-align: center; padding: 2px;">5</td> <td style="text-align: center; padding: 2px;">4</td> <td style="text-align: center; padding: 2px;">3</td> <td style="text-align: center; padding: 2px;">2</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">0</td> </tr> <tr> <td style="text-align: center; padding: 2px;">CONTENT</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">1</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">0</td> <td style="text-align: center; padding: 2px;">1</td> </tr> </table> <p style="text-align: center; margin-left: 40px;">} <u>OPERAND</u></p>	<u>OP</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>	IMM		LOW	S11	111B	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
<u>OP</u>	<u>SPECIAL</u>	<u>MODIFIER</u>	<u>STORE</u>	<u>OPERAND</u>																																									
IMM		LOW	S11	111B																																									
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																													

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																					
<p>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:</p> <p>a. The decimal number must be in the range 0 to 255.</p> <p>b. The octal number must be in the range 0 to 377, followed by "B".</p> <p>Examples:</p> <p style="text-align: center;">117B, 117, 198, 5, 10B</p>																					
WORD TYPE III - BRANCH FIELD																					
<p>JMP</p>	<p>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.</p> <p>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:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;"><u>BRANCH</u></th> <th style="text-align: center;"><u>SPECIAL</u></th> <th style="text-align: center;"><u>CONDITION</u></th> <th style="text-align: center;"><u>BRANCH SENSE</u></th> <th style="text-align: center;"><u>ADDRESS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">JMP</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">AL15</td> <td></td> <td style="text-align: center;">*+2</td> </tr> </tbody> </table> <p>A microbranch will occur if bit 15 of the ALU output was set during execution of the last word type I or II microinstruction.</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;"><u>BRANCH</u></th> <th style="text-align: center;"><u>SPECIAL</u></th> <th style="text-align: center;"><u>CONDITION</u></th> <th style="text-align: center;"><u>BRANCH SENSE</u></th> <th style="text-align: center;"><u>ADDRESS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">JMP</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">AL15</td> <td style="text-align: center;">RJS</td> <td style="text-align: center;">ADDRESS</td> </tr> </tbody> </table> <p>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).</p>	<u>BRANCH</u>	<u>SPECIAL</u>	<u>CONDITION</u>	<u>BRANCH SENSE</u>	<u>ADDRESS</u>	JMP	CNDX	AL15		*+2	<u>BRANCH</u>	<u>SPECIAL</u>	<u>CONDITION</u>	<u>BRANCH SENSE</u>	<u>ADDRESS</u>	JMP	CNDX	AL15	RJS	ADDRESS
<u>BRANCH</u>	<u>SPECIAL</u>	<u>CONDITION</u>	<u>BRANCH SENSE</u>	<u>ADDRESS</u>																	
JMP	CNDX	AL15		*+2																	
<u>BRANCH</u>	<u>SPECIAL</u>	<u>CONDITION</u>	<u>BRANCH SENSE</u>	<u>ADDRESS</u>																	
JMP	CNDX	AL15	RJS	ADDRESS																	
<p>JSB</p>	<p>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.</p> <p>Usage: Three levels of microsubroutine branches can be made.</p>																				

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

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

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

MICRO-ORDER	DEFINITION
WORD TYPE III - CONDITION FIELD (CONT.)	
MPP	<p>Meaning: Test for a signal $\overline{\text{MPP}}$ received at the MPP. The L-register must not be changed in the microinstruction immediately preceding the microinstruction containing MPP.</p> <p>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 <i>HP 21MX M-Series and E-Series Computers I/O Interfacing Guide</i> for further information. Example microprogram use will be found in section 13 of this manual.</p>
MRG	<p>Meaning: A Memory Reference Group instruction is in the IR; i.e., IR bits 14, 13, and 12 are not all zero.</p>
NDEC	<p>Meaning: The Operator Panel DEC M/m pushbutton is not actuated.</p>
NINC	<p>Meaning: The Operator Panel INC M/m pushbutton is not actuated.</p>
NINT	<p>Meaning: An interrupt is not pending.</p>
NLDR	<p>Meaning: The Operator Panel IBL/TEST pushbutton is not actuated.</p>
NLT	<p>Meaning: The Operator Panel Register Select (left) pushbutton is not actuated.</p>
NMDE	<p>Meaning: The Operator Panel MODE pushbutton is not actuated.</p>
NMLS	<p>Meaning: Memory was not lost as a result of the last power down or power failure.</p>
NRT	<p>Meaning: The Operator Panel Register Select (right) pushbutton is not actuated.</p>
NSFP	<p>Meaning: A standard Operator Panel is not installed on the computer.</p>
NSNG	<p>Meaning: The Operator Panel INSTR STEP pushbutton is not actuated.</p>
NSTB	<p>Meaning: None of the following Operator Panel pushbuttons are actuated:</p> <p style="padding-left: 40px;">INSTR STEP Register Select right (→) Register Select left (←) MODE IBL/TEST INC M/m DEC M/m STORE RUN PRESET</p>
NSTR	<p>Meaning: The Operator Panel STORE pushbutton is not actuated.</p>
ONES	<p>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.</p>
OVFL	<p>Meaning: The Overflow register bit is set.</p>
RUN	<p>Meaning: The computer's Run flip-flop is set.</p>

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

MICRO-ORDER	DEFINITION
WORD TYPE III - CONDITION FIELD (CONT.)	
RUNE	<p>Meaning: The LOCK/OPERATE switch is in the OPERATE position.</p> <p style="text-align: center;">NOTE</p> <p style="text-align: center;">In LOCK position, the RUN and HALT switches are disabled.</p>
SKPF	<p>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.</p> <p>Usage: Refer to section 7 for information on I/O microprogramming considerations for use of the SKPF micro-order.</p>
WORD TYPE III - BRANCH SENSE FIELD	
RJS	<p>Meaning: Perform the branch or return specified in the Branch field if the condition specified in the Condition field is <i>not</i> 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.</p> <p>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).</p>

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

MICRO-ORDER	DEFINITION																																																
WORD TYPE III - ADDRESS FIELD																																																	
<p>A branch may be made to any address in the current or next 512₁₀ word control memory block for word type III. The entry for the microassembler format can be an octal, decimal, or a computed address.</p> <p>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 512₁₀ word block (i.e., address is xx777₈), the range is defined as the next 512₁₀ word block. A computed address which is within the decimal or octal range must be in one of the following forms:</p> <p>*+ d *- d LABEL + d LABEL - d *+ kB *- kB LABEL + kB LABEL - kB LABEL</p> <p>where:</p> <p>* 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.</p> <p>Examples:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;"><u>BRANCH</u></th> <th style="text-align: center;"><u>SPECIAL</u></th> <th style="text-align: center;"><u>CONDITION</u></th> <th style="text-align: center;"><u>BRANCH SENSE</u></th> <th style="text-align: center;"><u>ADDRESS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">JMP</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">NSNG</td> <td></td> <td style="text-align: center;">*+ 2</td> </tr> <tr> <td style="text-align: center;">JMP</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">FLAG</td> <td></td> <td style="text-align: center;">*- 4</td> </tr> <tr> <td style="text-align: center;">JSB</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">CNT4</td> <td style="text-align: center;">RJS</td> <td style="text-align: center;">FETCH + 1</td> </tr> <tr> <td style="text-align: center;">JSB</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">IR8</td> <td></td> <td style="text-align: center;">TIME - 4</td> </tr> <tr> <td style="text-align: center;">JMP</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">IR11</td> <td style="text-align: center;">RJS</td> <td style="text-align: center;">*+ 7B</td> </tr> <tr> <td style="text-align: center;">JMP</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">L0</td> <td></td> <td style="text-align: center;">*- 2B</td> </tr> <tr> <td style="text-align: center;">JMP</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">ALZ</td> <td></td> <td style="text-align: center;">LOOP</td> </tr> <tr> <td style="text-align: center;">RTN</td> <td style="text-align: center;">CNDX</td> <td style="text-align: center;">ALZ</td> <td style="text-align: center;">RJS</td> <td></td> </tr> </tbody> </table> <p style="text-align: center;">NOTE</p> <p style="text-align: center;">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.</p> <p>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.</p> <p>Refer to section 6 for additional information on CM addressing. Refer to section 8 for information on using the RTE Microassembly language.</p>					<u>BRANCH</u>	<u>SPECIAL</u>	<u>CONDITION</u>	<u>BRANCH SENSE</u>	<u>ADDRESS</u>	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	
<u>BRANCH</u>	<u>SPECIAL</u>	<u>CONDITION</u>	<u>BRANCH SENSE</u>	<u>ADDRESS</u>																																													
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																																														

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

MICRO-ORDER	DEFINITION																														
WORD TYPE IV - BRANCH FIELD																															
JMP	<p>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.</p> <p>Usage: Refer to the Modifier/Special field micro-orders and the Address field discussions.</p>																														
JSB	<p>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.</p> <p>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.</p>																														
WORD TYPE IV - MODIFIER/SPECIAL FIELD																															
IOFF	<p>Meaning: Turn off the Interrupt Enable flag to disable recognition of normal interrupts. (Does not disable power fail, Memory Protect, or parity interrupts.)</p> <p>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.</p>																														
IOG	<p>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:</p> <table border="1" data-bbox="486 1212 1324 1719" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;"><u>ASSEMBLY LANGUAGE INSTRUCTION IN IR</u></th> <th style="text-align: center;"><u>IR BITS 8, 7, 6</u></th> <th style="text-align: center;"><u>ADDRESS FIELD BITS 8 AND 7 REPLACED BY:</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MIA or MIB</td> <td style="text-align: center;">1 0 0</td> <td style="text-align: center;">0 0</td> </tr> <tr> <td style="text-align: center;">LIA or LIB</td> <td style="text-align: center;">1 0 1</td> <td style="text-align: center;">0 1</td> </tr> <tr> <td style="text-align: center;">OTA or OTB</td> <td style="text-align: center;">1 1 0</td> <td style="text-align: center;">1 0</td> </tr> <tr> <td style="text-align: center;">HLT</td> <td style="text-align: center;">0 0 0</td> <td style="text-align: center;">1 1</td> </tr> <tr> <td style="text-align: center;">CLO or CLF</td> <td style="text-align: center;">0 0 1</td> <td style="text-align: center;">1 1</td> </tr> <tr> <td style="text-align: center;">STO or STF</td> <td style="text-align: center;">0 0 1</td> <td style="text-align: center;">1 1</td> </tr> <tr> <td style="text-align: center;">SFC or SOC</td> <td style="text-align: center;">0 1 0</td> <td style="text-align: center;">1 1</td> </tr> <tr> <td style="text-align: center;">SFS or SOS</td> <td style="text-align: center;">0 1 1</td> <td style="text-align: center;">1 1</td> </tr> <tr> <td style="text-align: center;">STC or CLC</td> <td style="text-align: center;">1 1 1</td> <td style="text-align: center;">1 1</td> </tr> </tbody> </table> <p>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.</p>	<u>ASSEMBLY LANGUAGE INSTRUCTION IN IR</u>	<u>IR BITS 8, 7, 6</u>	<u>ADDRESS FIELD BITS 8 AND 7 REPLACED BY:</u>	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
<u>ASSEMBLY LANGUAGE INSTRUCTION IN IR</u>	<u>IR BITS 8, 7, 6</u>	<u>ADDRESS FIELD BITS 8 AND 7 REPLACED BY:</u>																													
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																													

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

MICRO-ORDER	DEFINITION
WORD TYPE IV - MODIFIER/SPECIAL FIELD (CONT.)	
ION	<p>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).</p> <p>Usage: An interrupt from any I/O device can be detected in two ways:</p> <ol style="list-style-type: none"> 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 enclosed 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 <i>Computer Series Operating and Reference Manual</i>. Refer to section 7 for considerations for interrupt handling.
J74	<p>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.</p>
NOP	<p>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.</p>
RJ30	<p>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.</p> <p>Usage: Refer to the word type I OP field READ micro-order definition for M-register considerations.</p>
RPT	<p>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.</p> <p>Usage: Same as for the word type I and II Special field RPT micro-order.</p>
STFL	<p>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.</p>

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

MICRO-ORDER	DEFINITION																		
WORD TYPE IV - ADDRESS FIELD																			
<p>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.</p> <p>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:</p> <ul style="list-style-type: none"> * + d * - d LABEL + d LABEL - d *+ kB *- kB LABEL + kB LABEL - kB LABEL <p>where:</p> <ul style="list-style-type: none"> * 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. <p>Examples:</p> <table border="1" data-bbox="247 1239 1350 1404" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;"><u>BRANCH</u></th> <th style="text-align: center;"><u>MODIFIER/ SPECIAL</u></th> <th style="text-align: center;"><u>(NO ENTRY)</u></th> <th style="text-align: center;"><u>(NO ENTRY)</u></th> <th style="text-align: center;"><u>ADDRESS</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">JSB</td> <td style="text-align: center;">IOFF</td> <td></td> <td></td> <td style="text-align: center;">* + 11</td> </tr> <tr> <td style="text-align: center;">JMP</td> <td></td> <td></td> <td></td> <td style="text-align: center;">FETCH</td> </tr> </tbody> </table> <p>(Refer to the word type III Address field examples.)</p> <p>Refer to section 6 for additional information on CM addressing. Refer to section 8 for information on using the RTE Microassembly language.</p>					<u>BRANCH</u>	<u>MODIFIER/ SPECIAL</u>	<u>(NO ENTRY)</u>	<u>(NO ENTRY)</u>	<u>ADDRESS</u>	JSB	IOFF			* + 11	JMP				FETCH
<u>BRANCH</u>	<u>MODIFIER/ SPECIAL</u>	<u>(NO ENTRY)</u>	<u>(NO ENTRY)</u>	<u>ADDRESS</u>															
JSB	IOFF			* + 11															
JMP				FETCH															

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.

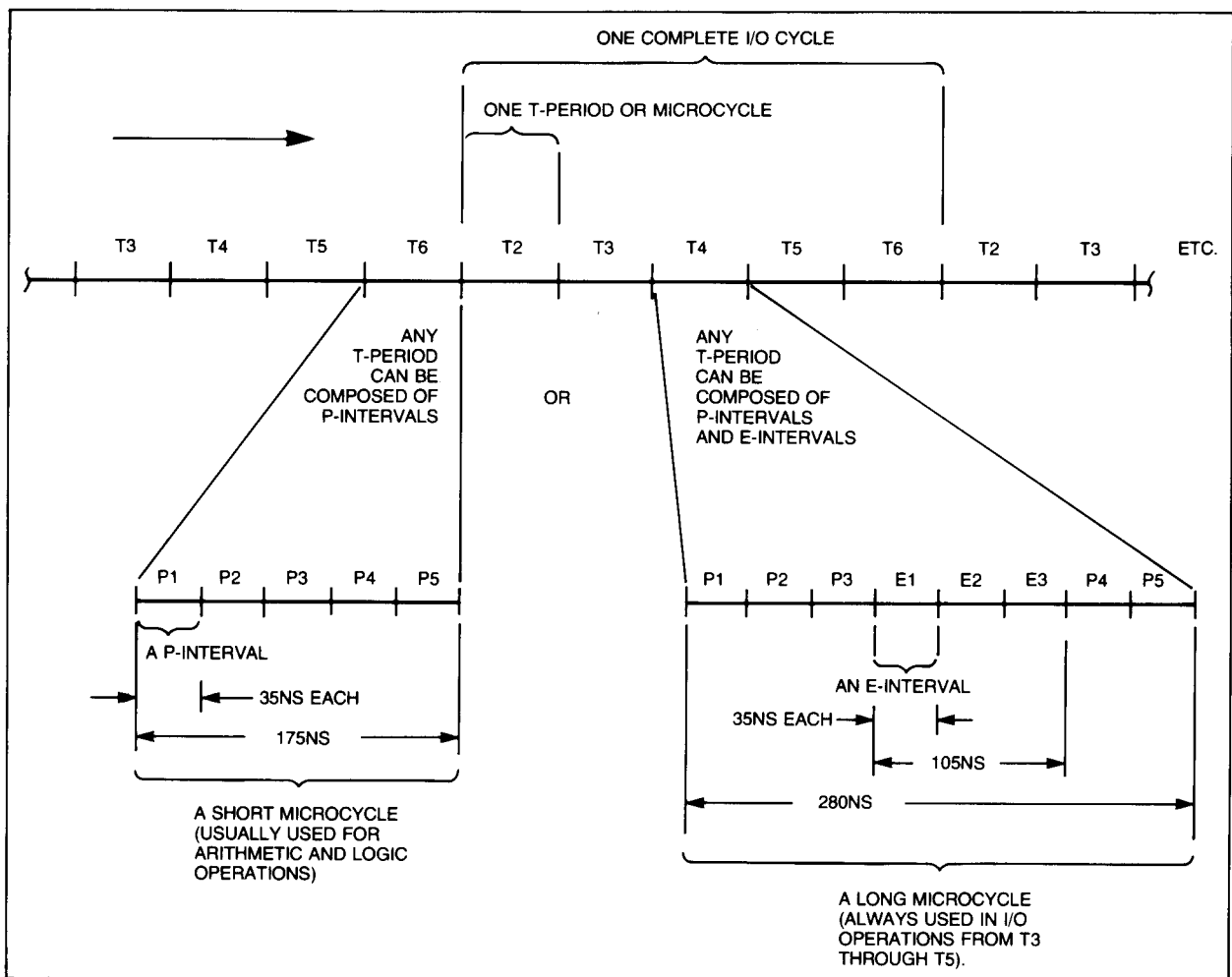
Each of these "functions" essentially operates asynchronously until they are required to communicate in order to perform a "unit" task such as a main memory read or write operation, or some I/O operation.

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 M-Series and 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.

Timing

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

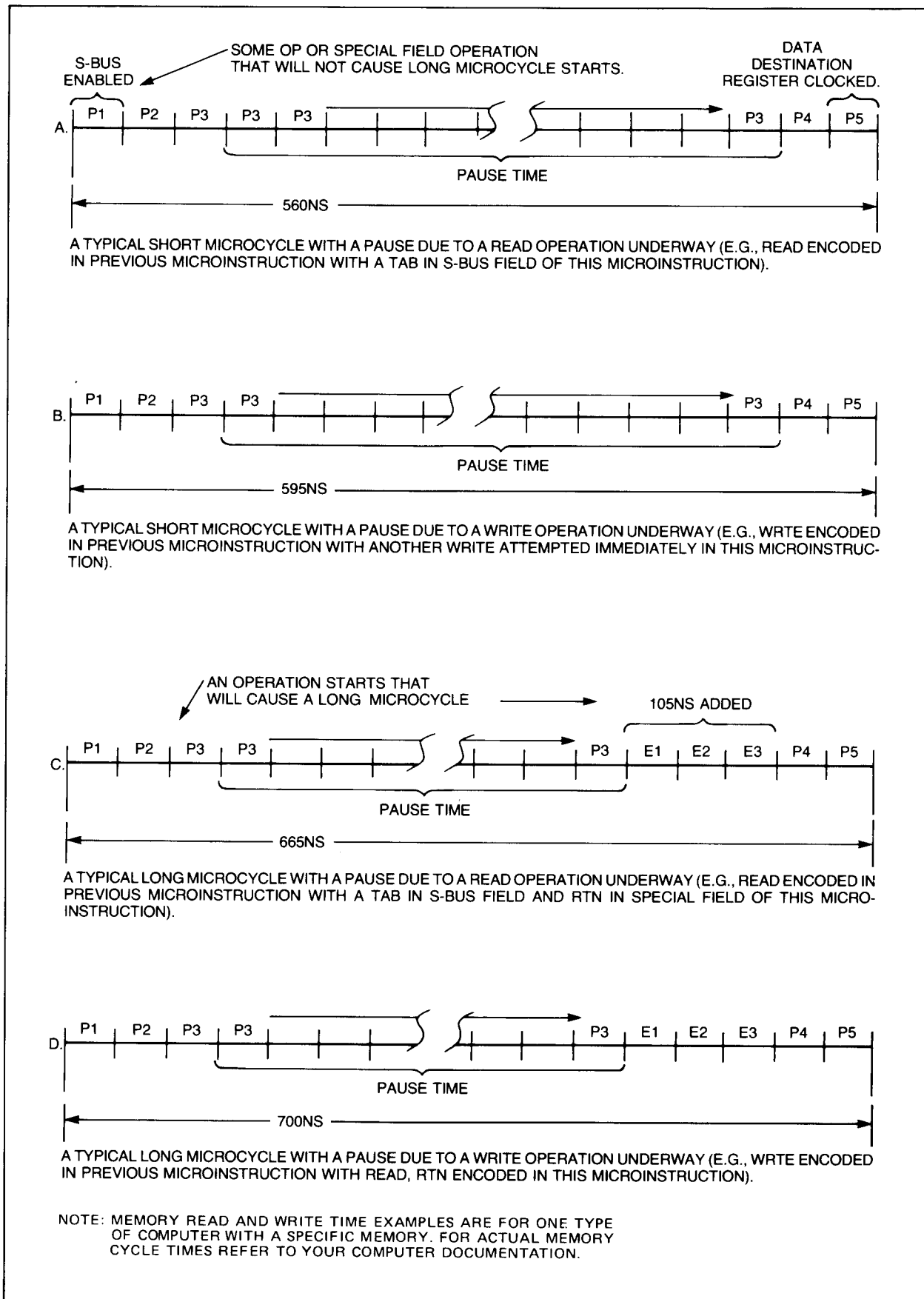
The Control Processor and I/O section operate asynchronously until an I/O instruction begins execution and communication is needed. That is, although T-periods run sequentially from T2 through T6, and each T-period is initiated by P1 of any microcycle, I/O microinstructions must begin at the appropriate 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.

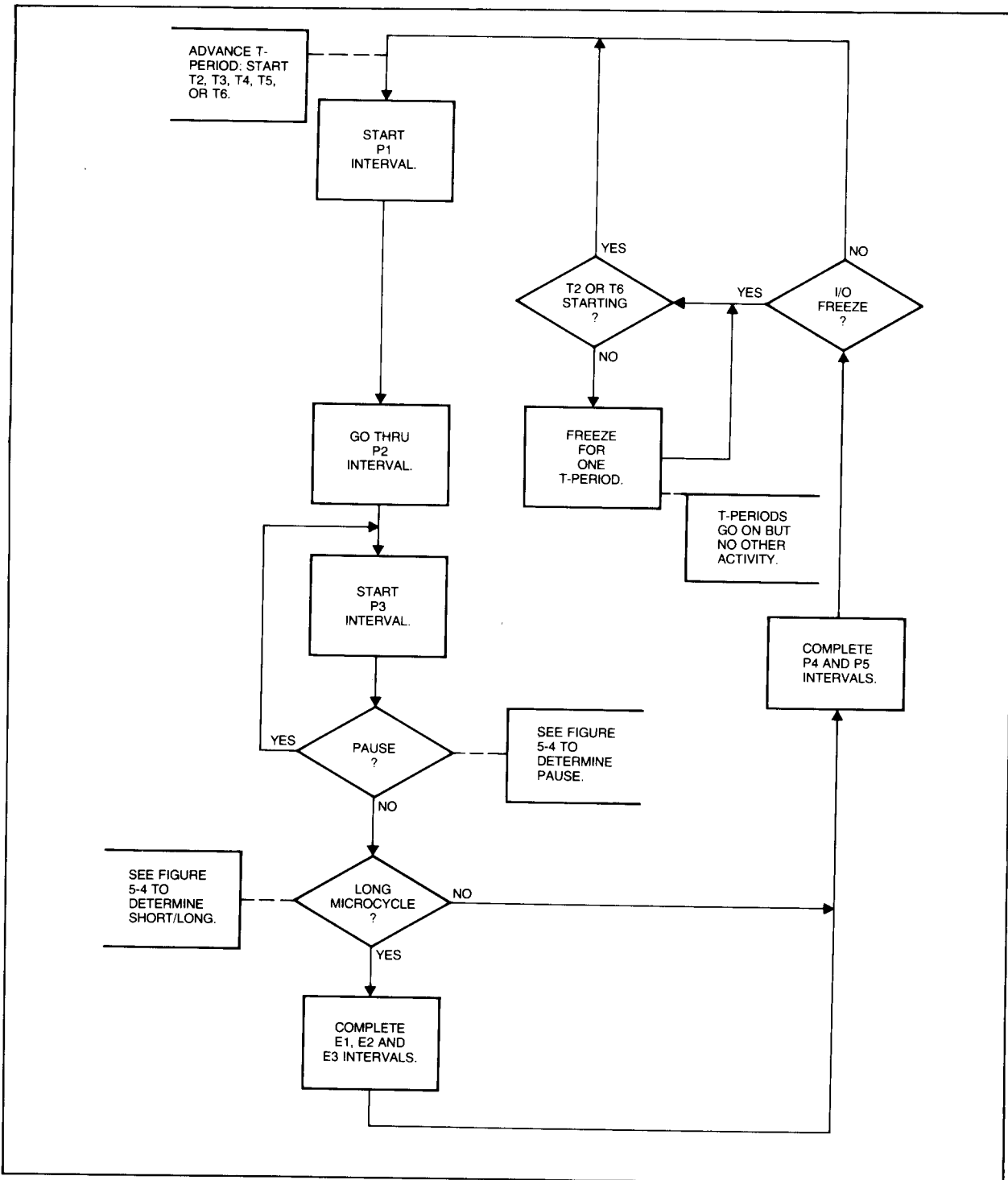


7115-14

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 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

Timing

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.

5-9. ARITHMETIC/LOGIC SECTION OPERATIONS

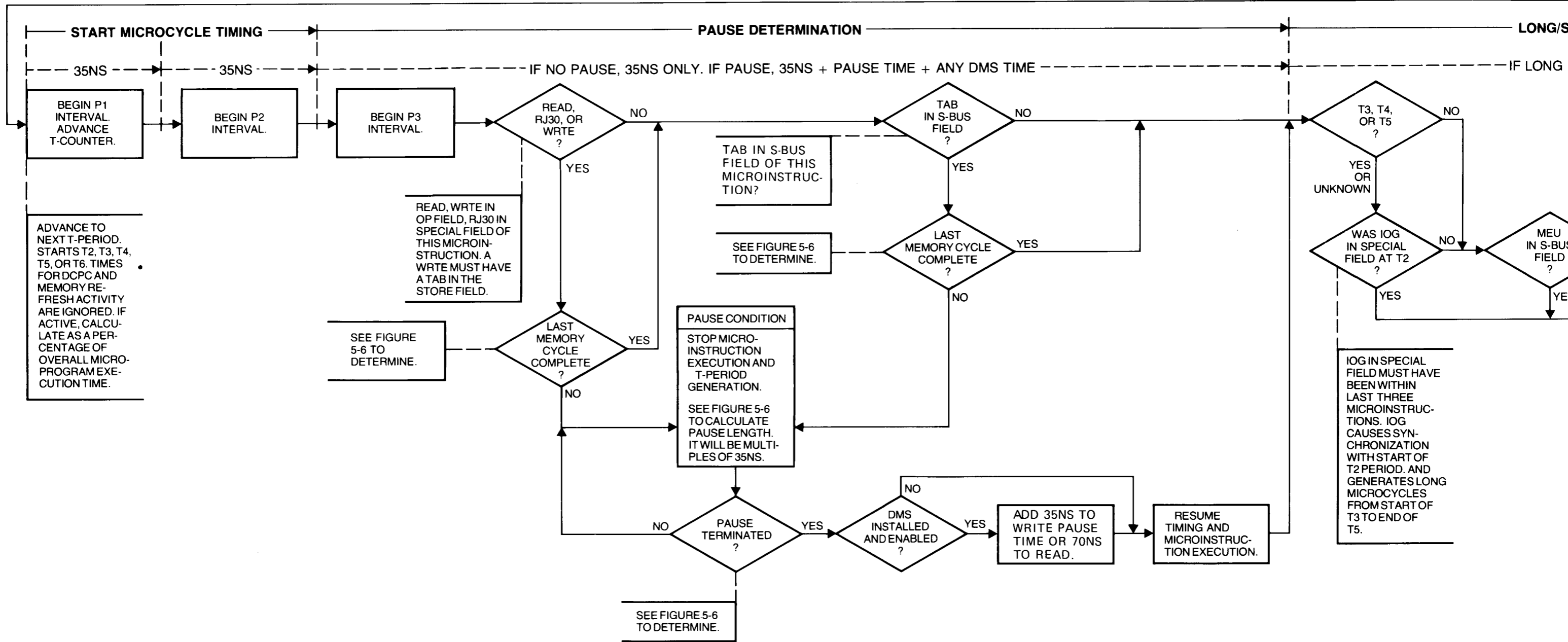
The fastest microcycle timing is found when microprogrammed operations deal with the Arithmetic/Logic section registers. For example, suppose the timing for the following portion of a microroutine is to be estimated:

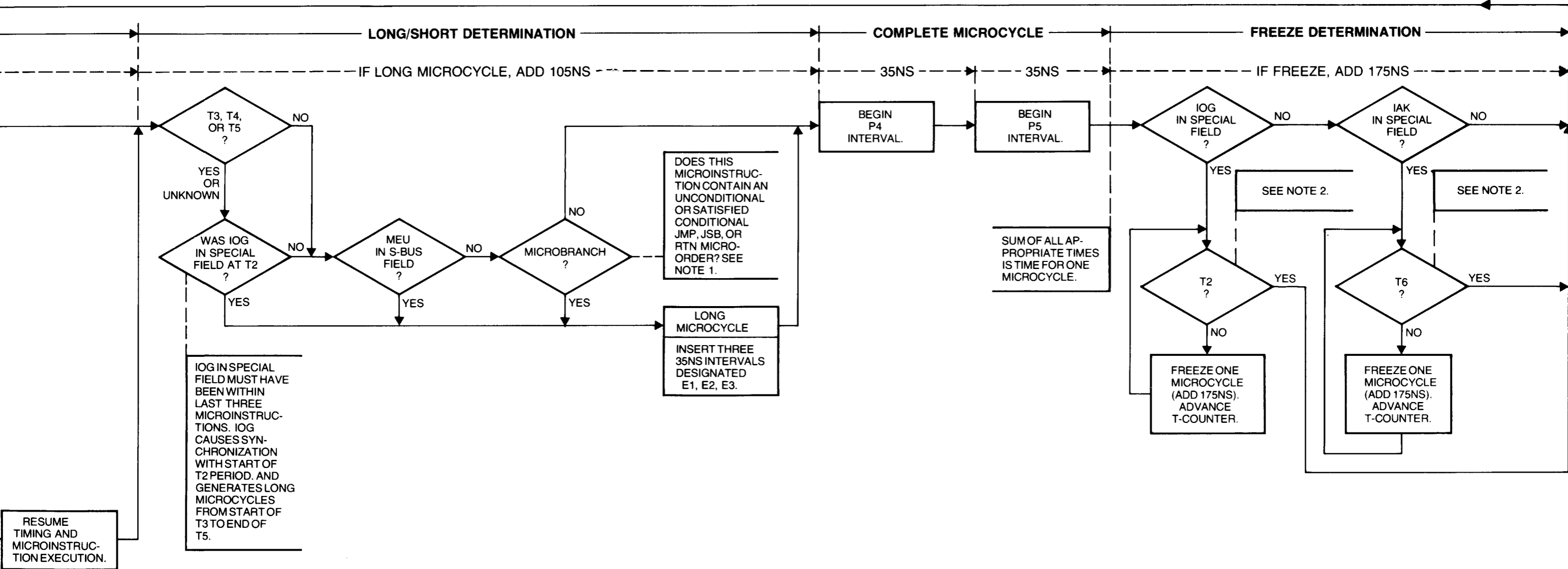
LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
FIRST		STFL	CMPS	B	B	
SECOND			CMPS	A	A	
THIRD			INC	A	A	
			.			
			.			
			(ETC.)			

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.





NOTES:
 1. CONDITIONAL MICROBRANCHES NOT MET MAY BE DIFFICULT TO DETERMINE. ASSUME A PERCENTAGE OF BRANCHES MET BASED ON YOUR APPLICATION.
 2. TO DETERMINE WHICH T-PERIOD IS PRESENT WHEN BEGINNING AN I/C CYCLE TREAT THE ESTIMATE AS RANDOM.

Figure 5-5. Detailed Microcycle Time Determination Flowchart

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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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	
						<hr/>	<hr/>
						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):

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS	
*						TIME (NS)	
			.				
			.				
XXX		I0G		IRCM	S4	T4	175
*		↓		↓	↓	T5	175
*		(SUSPENDED EXECUTION UNTIL T2)				T6	175
*		(NOW EXECUTION CONTINUES)				T2	175
		NOP				T3	280
		NOP				T4	280
				S5	I0I	T5	280
			INC	S8	S3	T6	175
						T2	
			.				
			.				
			.				
			(ETC.)				

Timing

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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
*						TIME (NS)
			.			
			.			
START			PASS	S1	P	175
FIRST	READ		PASS	DSPL	S11	175
SECOND			INC	PNM	P	175
THIRD			DEC	X	X	175
DATA			PASS	S2	TAB	210
END		RTN		IRCM	S2	280
			.			
			.			
			.			
			(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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
*						TIME (NS)
			.			
			.			
ENTER			INC	X	X	175 ← A
WRITE	WRTE	MPCK	PASS	TAB	X	175 ← B
CHECK	JMP	CNDX	ALZ	RJS	**+2	175 ← C
GO	READ	RTN	INC	PNM	P	560 ← E
			.			
			.			
			(ETC.)			

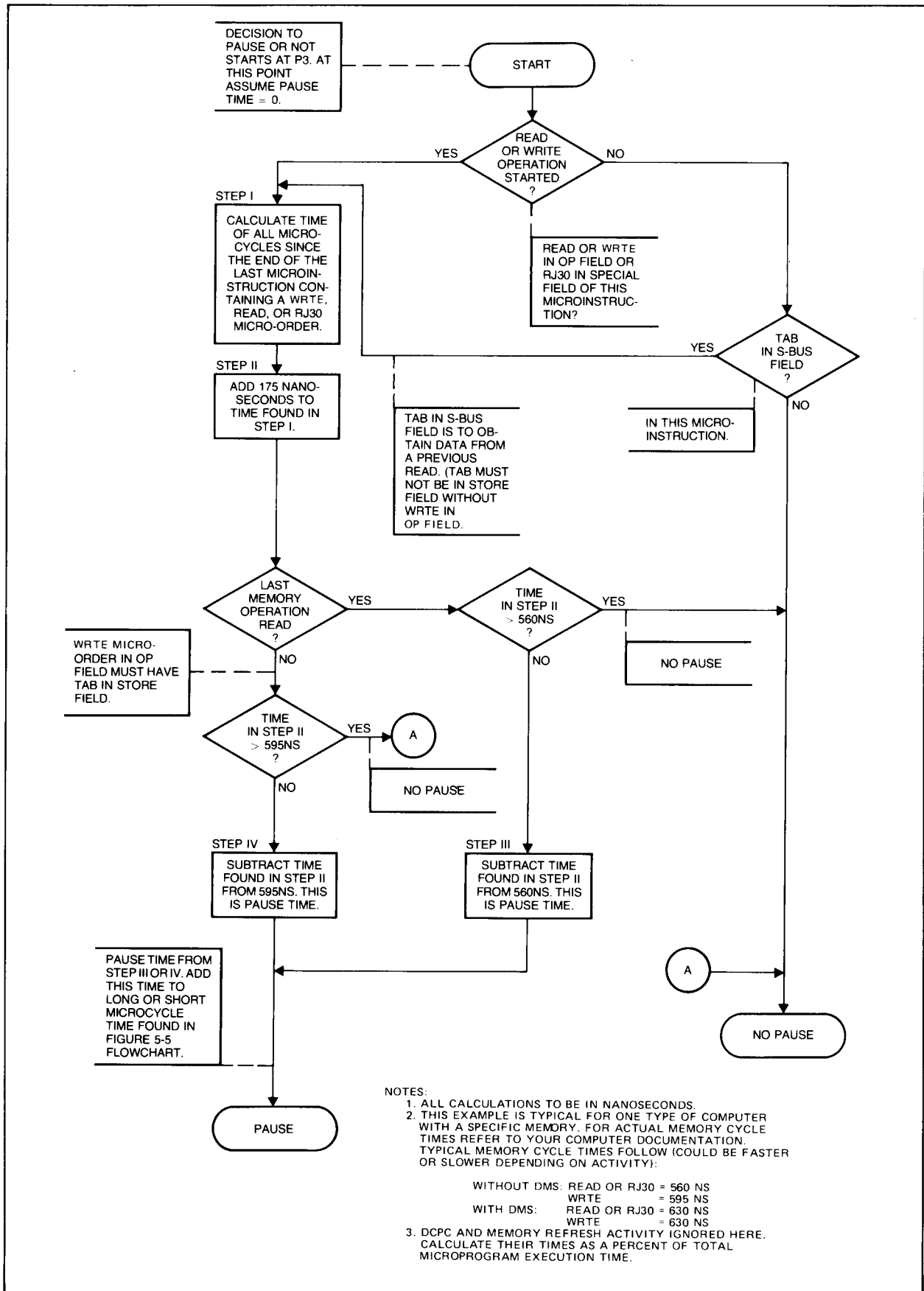


Figure 5-6. Detailed Pause Time Calculation Flowchart
(Using an HP 2102B Memory as an Example)

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
<hr/>	
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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
*						TIME (NS)
			.			
			.			
START	READ	CLFL	PASS	M	S1	175
			PASS	L	S2	175
	ENVE		ADD	S3	TAB	385
	READ		PASS	M	S3	175
	IMM		CMLD	L	374B	175
			ADD	L	S3	175
	ENVE		ADD	S3	TAB	210
	RTN	CNDX	OVFL			280/175
	RTN	SDV				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, FEM, 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 overlaid) 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 discretion 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 tables 6-1 and 6-2. These tables are 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 tables 6-1 and 6-2, 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 these tables, 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 105437

or

101 (or 105) $\left\{ \begin{array}{l} 460 \text{ through } 477 \\ 700 \text{ through } 777 \end{array} \right.$

Table 6-1. Control Memory User Instruction Group Software Entry Point Assignments
(E-Series)

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
105200-105217	34	21000-21017	FFP
105220-105237	35	21400-21417	FFP
105240-105257	36	22000-22017	EMA
105260-105277	37	22400-22417	HP Reserved
105300-105317	38	23000-23017	DS/1000
105320-105337	40	24000-24017	HP Reserved
105340-105357	16	10000-10017	HP Reserved
105360-105377	42	25000-25017	HP Reserved
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	39	23400-23417	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.

6-5. USER AREA UIG CODES. Modules 46 through 62 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 tables 6-1 and 6-2. As apparent from the tables, 11 of the 18 modules in the range 46 through 62 are directly accessible. Entry to other control memory modules will require an extra branch after reaching control memory.

Table 6-1A. Control Memory User Instruction Group Software Entry Point Assignments
(F-Series)

RANGE OF UIG INSTRUCTION (MAIN MEMORY) VALUES USED (OCTAL)	MODULE MAPPED TO	CONTROL MEMORY ENTRY POINTS (RANGE OF ADDRESSES) (OCTAL) (NOTE 3)	USE
105000-105137	3	01xxx (NOTE 1)	Floating Point
105140-105157	60	36000-36017	User area
105160-105177	62	37000-37017	User area
105200-105217	34	21000-21017	FFP
105220-105237	35	21400-21417	FFP
105240-105257	36	22000-22017	EMA
105260-105277	37	22400-22417	HP Reserved
105300-105317	38	23000-23017	DS/1000
105320-105337	40	24000-24017	SIS
105340-105357	16	10000-10017	HP Reserved
105360-105377	18	11000-11017	HP Reserved
101 (or 105) 400-417	4	02000-02017	HP Reserved
101 (or 105) 420-437	20	12000-12017	HP Reserved
101 (or 105) 440-457	46	27000-27017	User area
101 (or 105) 460-477	12	06000-06017	VIS
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:

- xxx signifies last three digits for the entry address. See appendix C for details.
- All modules except 2, 3, and 32 have 16 entry points. See appendix C.

As can also be seen in the tables, 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 JTAB	PASS INC	IRCM PNM	TAB P	IR = 105602, L = 0 M = I + 1, P = I + 2
			.			
			.			

Mapping

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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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):

LOCATION	LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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 micro-program 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.

Mapping

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 tables 6-1 and 6-2) 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

```

0001          MICMXE,L                21MX E-SERIES
0002          $CODE=MPORJ,REPLACE    OBJECT TO DISC
0003          INDIRECT EQU 34355R    USER WRITTEN
0004          *                      INDIRECT
0005          *                      MICROPROGRAM
0006          *                      (SEE EXAMPLE 2)
0007          ORG 34003R              105603 => 34003
0008 34003 327 001407              JMP          INST03MC SAVE ENTRY
0009          *                      POINTS
0010          * THIS MICROPROGRAM IS AN EXAMPLE OF HOW TO
0011          * RETRIEVE MAIN MEMORY PARAMETERS AND ADDRESSES
0012          *
0013          * A USER WRITTEN MICROSUBROUTINE (SEE EXAMPLE 2)
0014          * WILL BE USED TO RESOLVE INDIRECT ADDRESSES
0015          *
0016          * INITIALIZE THE CNTR
0017          * THE USER WRITTEN INDIRECT MICROPROGRAM (EXAMPLE 2),
0018          * IF INTERRUPTED, USES THE CNTR TO ADJUST P (I.E.
0019          * SET P TO MAIN MEMORY ADDRESS + 1 OF THE
0020          * MICROPROGRAM OP CODE)
0021          ORG 34030B
0022 34030 343 176547 INST03MC IMM      LOW CNTR 377R      CNTR = -1
0023          *
0024          * GET PARAMETERS:
0025          * SELECT CODE, DATA COUNT, BUFFER ADDRESS
0026 34031 227 174725          READ DCNT INC PNM P      GET SELECT CODE
0027 34032 307 016647          JSR                      INDIRECT RESOLVE ADDR
0028 34033 010 000507          L          TAB          L = SELECT CODE
0029          *
0030 34034 227 174725          READ DCNT INC PNM P      GET DATA COUNT
0031 34035 307 016647          JSR                      INDIRECT RESOLVE ADDR
0032 34036 353 007123          IMM L4 CML0 S3 303B      (SEE NOTE 1)
0033 34037 010 001147          S4          TAB          S4 = DATA COUNT
0034          *
0035 34040 227 174725          READ DCNT INC PNM P      GET BUFFER ADDR
0036 34041 010 145107          IOP S3 S3          (SEE NOTE 1)
0037 34042 307 016647          JSR                      INDIRECT RESOLVE ADDR
0038 34043 010 033207          S5          M          S5 = BUFFER ADDR
0039          *
0040          * NOTE 1. ONE NON-FREEZABLE MICROINSTRUCTION MAY
0041          * PRECEDE AND/OR FOLLOW THE JSB INDIRECT'S
0042          *
0043 34044 227 174700          READ RTN INC PNM P      START FETCH FOR
0044          *                      NEXT MAIN MEMORY
0045          *                      INSTRUCTION
0046          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=INDORJ,REPLACE         OBJECT TO DISC
0003          HORI      EQU  6              BASE SET HALT-
0004          *                               OR-INTERRUPT
0005          *                               MICROROUTINE
0006          *                               ORG  34355H
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   TAR   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 NSMG RJS  NEXT  INSTR STEP?
0042          *                               NO, NEXT ADDR
0043 34365 010 026507  *                               YES, ADJUST P
0044 34366 320 000307  *                               JMP          L   CNTR
0045          *                               JMP          HORI  EXIT TO HORI
          *                               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 tables 6-1 and 6-2

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 MID,105602B,4
0007*
0008 00000 000000  START NOP
0009*
0010*
0011 00001 105600  SDRT  SRT
0012*
0013*
0014 00002 105602  MC10  MID **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$

```

Mapping

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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

Considerations

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
	IMM		CMLD	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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			CMLD	S3	376B	S3 = 1
	IMM			M	S3	175 NS, PRST CLEARS BAF
	READ	PRST		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.

Considerations

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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
	IMM		CMLD	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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			.			
	WRTE	MPCK		M	S3	
				TAB	S4	175 NS
			.			
			.			
			.			

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

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			.			
			CMLD	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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			.			
			ZERO	S3		
				M	S3	
*	WRTE	PRST MPCK		TAB	S4	PRST CLEARS AAF 175 NS, MEMORY LOCATION 0 = S4, A-REGISTER UNALTERED
			.			
			.			
			.			

Considerations

Writing into main memory location 01 is accomplished as follows:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
	IMM		CMLD	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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			.			
*	WRTE	MPCK		M TAB	S3 S4	M = ADDRESS TO BE WRITTEN INTO. MPCK AS USED HERE WILL CORRECTLY CHECK FOR A MEMORY PROTECT VIOLATION.
*	READ	MPCK		M	S5	MPCK AS USED HERE WILL CORRECTLY 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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			.			
	READ			M	S3	175 NS
	WRTE			TAB	TAB	595 NS
			.			
			.			
			.			

The following READ is conditionally valid:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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
			.			
			.			
			.			

Considerations

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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

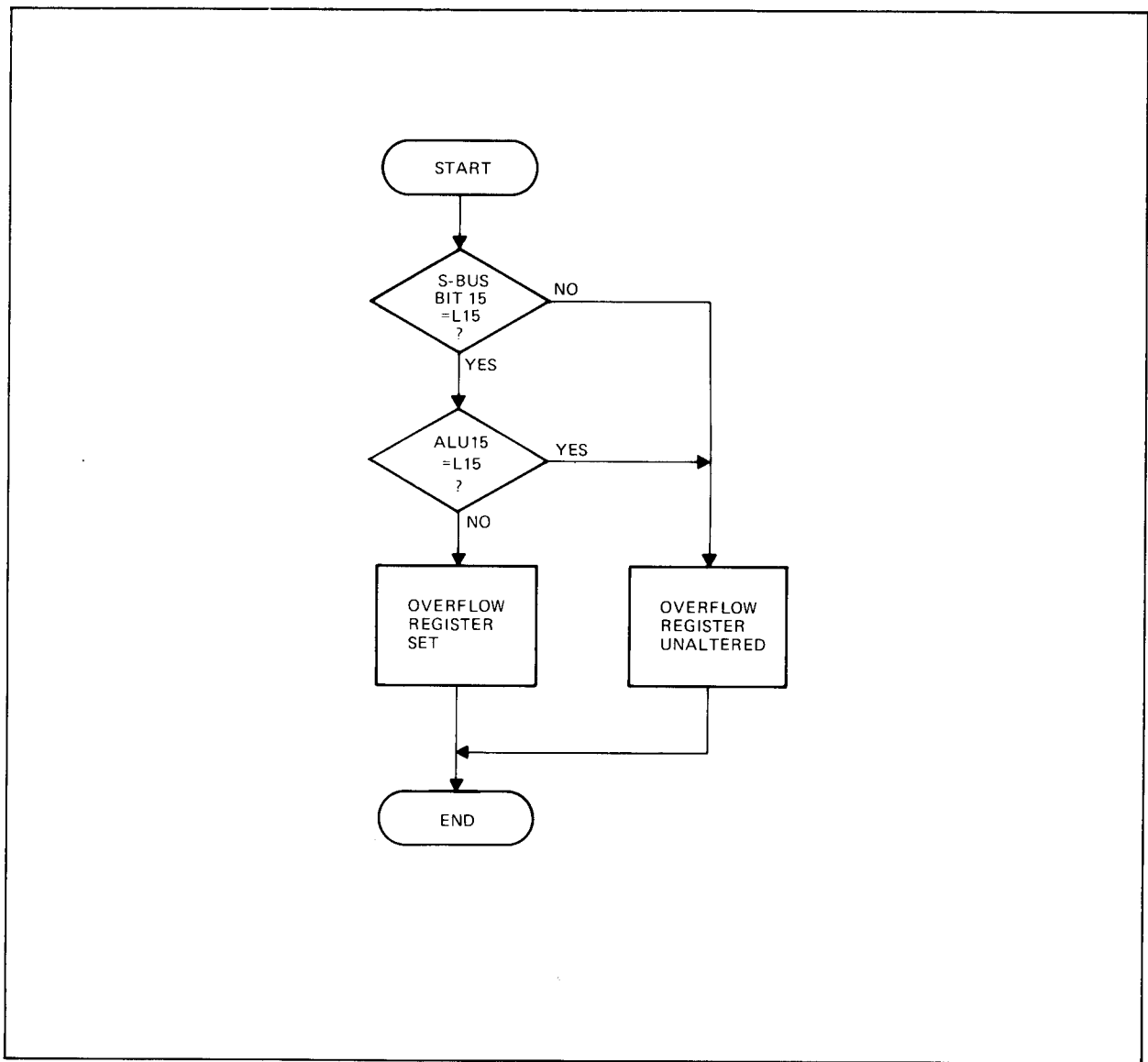
LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
	IMM		LOW	IRCM	305B	IR(7,6,2,0) = 1,1,1,1, = CCE,RSS.
		ASG	INC	S4	S4	CCE,S4 = S4+1
			.			
			.			

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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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
* SETTING OVERFLOW ARITHMETICALLY						
	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 microinstructions, 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			.	S3	P	SAVE P
				P	B	P = BUFFER ADDRESS
			.			
			.			
LOOP	READ		INC	PNM	P	READ BUFFER, UPDATE BUFFER
LOOPEND						ADDRESS.
			.			
			.			
MPEND	READ	RTN	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 microinstruction 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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
			.			
				CNTR	A	CNTR = LOOP COUNT.
			.			
			.			
LOOP	READ	DCNT	INC	PNM	P	READ BUFFER, UPDATE BUFFER
*						ADDRESS AND LOOP COUNT.
LOOPEND	JMP	CNDX	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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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, NO, C1 LESS THAN C2.
C1.LT.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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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						.
						.
						.

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 MPIO 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.

Considerations

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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
	READ		INC	PNM	P	
		IOG	PASS	S4	TAB	
			.			
			.			

- Load the Instruction Register before issuing an IOG. (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 0	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).
<p>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).</p>			

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 the occurrence of the IOG to ensure that the correct signals are sent to the desired SC (refer to paragraph 7-23). Do not modify IR or CNTR for 3 instructions following the IOG. 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 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
GO		IOG				T2
					CAB	T3
				I00	CAB	T4
		RTN		I00	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:

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

Considerations

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 address 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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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 JUMP	JMP		DEC	P	P 6	FIX P, EXIT TO HALT-OR- 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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.
			.			
			.			

7-31. 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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDRESS	COMMENTS
			.			
			.			
	IMM		CMHI	L	376B	L=LI* 0
	IMM		CMLD	S4	376B	S4=1
FAKESC		IOFF	IDR	S4	S4	S4=LI* 1
			IRCM	S4	S4	IR=LI* 1
REALSC			IDR	S4	S3	S4=LI* SC
			CNTR	S4	S4	IR=LI* SC
		IOG		M	S5	START I/O OPERATION
				S6	IOI	M=BUFFER ADDRESS
	WRTE	MPCK		TAB	S6	S6=DATA
						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.	<p>Check to see if I/O signal SFS is present (T3 to T5) <i>and</i> 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.</p> <p style="text-align: center;">— OR —</p> <p>Check to see if SFC signal is present (T3 to T5) and the I/O device's flag is clear.</p>
<p>NOTES:</p> <p>*This micro-order can also be used in the Special field of a word type IV (unconditional branch microinstruction).</p> <p>**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.</p>			

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 Q₄, 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 Q₅ 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 Q₄ signal has preceded this step by exactly one microcycle (i.e., Q₄, Q₅ 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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 1. Select opposite program map (does not change currently selected map per Q₅). 2. Can generate DMAFRZ to CPU.
Q ₄	<ol style="list-style-type: none"> 1. Set "Status Command" flag through next Control Processor cycle (defines Q₆ operation). 2. Reset to currently selected program map (nullifies Q₃). 3. Set "Enable Base Page Fence" Flag through next Control Processor cycle (partly defines Q₅ operation).
Q ₅	<ol style="list-style-type: none"> 1. Store S-bus into MEM (other than maps) <ol style="list-style-type: none"> 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 Q₄; 001 = Address Register; 010 = State Register. 3. If S-bus bit 15 = 1 then Memory Protect is disabled for the current microinstruction.
Q ₆	<ol style="list-style-type: none"> 1. Enable MEM data (other than maps) onto S-bus. <ol style="list-style-type: none"> a. Normally enables MEM Violation Register. b. If preceded by Q₄ signal microinstruction, Status Register enabled.
Q ₇	<ol style="list-style-type: none"> 1. No MEM (DMS) microinstruction specified (NOP state for MEM).
<p>Notes:</p> <ol style="list-style-type: none"> 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 $x = 0, 1, 2, \dots, 7$.
- 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 1000 E/F-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,EDITOR
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 *Lnn* (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 *ECnew* (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 Llist 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 a disc file. (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 interspersed 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:

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.

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 "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 is no longer used since the output file name is now specified in the run string.

```
$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 or a binary 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 binary 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 constraints 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

Pseudo-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	<i>expression</i>

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:

LOCATION	LABEL	OP/ BRCH	MOD/ SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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 pseudo-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.

Preparation

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
<i>label</i>	EQU	<i>expression</i>

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
<i>label</i>	DEF	<i>expression</i>

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:

Fields:	1	10	30
	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
<i>label</i>	ONES	—
<i>label</i>	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:

Fields:	1	10
	Field 1	Field 2
Content:	NEG 1	ONES

An example of using ZERO would be:

Character column:

Fields:	1	10	40
	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,L,T,C,R(or S)
$PAGE=title           | [ : [crlabel] ] [ ,REPLACE]
$LIST
$NOLIST
$PUNCH
$NOPUNCH
```

- Pseudo-microinstructions:

Columns	1-9	10	30-39
	LABEL	OP	OPERAND
	—	ORG	<i>expression</i>
	—	ALGN	—
	—	END	—
	<i>label</i>	EQU	<i>expression</i>
	<i>label</i>	DEF	<i>expression</i>
	<i>label</i>	ONES	—
	<i>label</i>	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



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 pseudomicroinstruction, 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.)

```
{ *ON
  *RU } ,MICRO,source input[.list output[.binary output[.line count[.options]]]]
:RU
```

source input Name of an FMGR file or a logical unit number of the device containing the Microassembler source code; this entry must conform to the format required by the FMGR *namr* parameter.

list output Choose one of the following:

- (minus symbol)
- FMGR file name
- logical unit number
- null (omitted)

If the minus symbol is specified, and the source file name begins with an ampersand, the ampersand is replaced with an apostrophe and the remaining source file name characters are used for the list file name. The list file is forced to reside on the same cartridge (cartridge reference code) as the source file. For example:

```
&FIL1      source file name
'FIL1      list file name
```

If an FMGR file name is specified, it must conform to the format required by the FMGR *namr* parameter. The list file is created if it does not exist. If the file does exist, the first character in the file name must be an apostrophe; otherwise, an error results.

If a logical unit number is specified, the listed output is directed to that logical device.

If this parameter is omitted, logical unit number 6 is assumed. Further, if subsequent parameters are specified, the comma must be used as a parameter placeholder.

binary output Choose one of the following:

- (minus symbol)
- FMGR file name
- logical unit number
- null (omitted)

If the minus symbol is specified, and the source file name begins with an ampersand, the ampersand is replaced with a percent symbol and the remaining source file name characters are used for the binary file name. This binary file is forced to reside on the same cartridge (cartridge reference code) as the source file. For example:

```
&FIL1      source file name
%FIL1      binary file name
```

If an FMGR file name is specified, it must conform to the format required by the FMGR *namr* parameter. The binary file is created if it does not exist. If the file exists, it is necessary that.

- a. the first character of the file's name be a percent sign (%).
- b. the existing file be of the type specified in the *namr* parameter (if the file type is not declared in *namr*, the file's type must be Type 5, relocatable binary).

If the above conditions are not met, a microassembler error will result.

If a logical unit number is specified, the binary output is directed to that logical device.

If this parameter is omitted, no binary object code is generated. Further, if the subsequent parameter is specified, the comma must be used as a parameter placeholder.

line count A decimal number which defines the number of lines per page for the list device.

Specification of this parameter is optional. If it is omitted, 56 lines per page are assumed. If a number less than 10 is specified, the compiler treats it as effectively infinite. The line count must be in the range $10 \leq \text{line count} \leq 999$.

options Up to five characters that select control function options. No commas are allowed within the option string. These characters are: R, S, L, T, and C. If specified, these options replace (override) the character options declared in the MICMXE control statement (see MIC Assembly Command).

Characters other than the above are ignored, except that any option specified in this parameter position negates all character options declared in the MICMXE control statement.

Examples:

```
*RU,MICRO,&PROGA,-,-
```

Schedules MICRO to microassemble source file &PROGA. Listed output is directed to list file 'PROGA and binary object code is directed to binary file %PROGA. The number of lines per list file page defaults to 56.

```
:RU,MICRO,&FIL1,'LIST
```

Schedules MICRO to microassemble source file &FIL1. Listed output is directed to list file 'LIST. No binary object code is generated. The number of lines per list file page defaults to 56.

```
:RU,MICRO,&ABCD
```

Schedules MICRO to microassemble source file &ABCD. Listed output defaults to LU 6. No binary object code is generated. The number of lines per list file page defaults to 56.

```
:RU,MICRO,&AAAA,-,-,28
```

Schedules MICRO to microassemble source file &AAAA. Listed output is directed to list file 'AAAA. Binary object code is directed to binary file %AAAA. The number of lines per list file page is 28.

```
:RU,MICRO,&SFIL,-,-,TSC
```

Schedules MICRO to microassemble source file &SFIL. Listed output is directed to list file 'SFIL. Binary object code is directed to binary file %SFIL. The number of lines per list file page defaults to 56. A symbol table will be produced, special format object code will be produced, and a cross reference table will be produced.

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 $5 + (2 \times 27)$: 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 having 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 (Type 5 files). Therefore, the object code can be stored from an input device into a disc file as a binary relocatable by the FMGR STORE command. If the microassembler includes a run string output file or LU as described in section 8, the microassembler automatically stores the object code into the specified file or LU.

The microassembler outputs non-standard HP ROM Simulator object code to the device if the "S" parameter is 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 specified 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  sole
RU,MXREF,input,list,lines  ole
```

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 microinstructions, 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.

Microassembling

(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
INEXIT	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 specified list device; informative messages are output to either the specified 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 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 completion 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	DUPLICATE 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 <i>filename</i> , <i>security</i> , or <i>crlabel</i> specification. (Refer to the <i>Batch and Spool Manual</i> .) 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
<p>NOTES:</p> <ol style="list-style-type: none"> 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=M-SERIES,2=E/F-SERIES  
TYPE(1 OR 2)?
```

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.

MDE

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 the microassembler.

Any microprograms residing in WCS that are overlaid 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
<i>lu1</i>	<i>xxxxx-yyyyy</i>	<i>z</i>
<i>lu2</i>	<i>xxxxx-yyyyy</i>	<i>z</i>
.		
.		
<i>lux</i>	<i>xxxxx-yyyyy</i>	<i>z</i>

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

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

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:

$$\text{RUN} \left[\begin{array}{l} ,105yyy \text{ B} \\ ,101zzz \text{ B} \end{array} \right]$$

where:

105yyyB and 101zzzB 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:

$$\text{SEt}[,p1[,p2\dots[p25]]]$$

MDE

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 (LD) 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(DEFAULT=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)

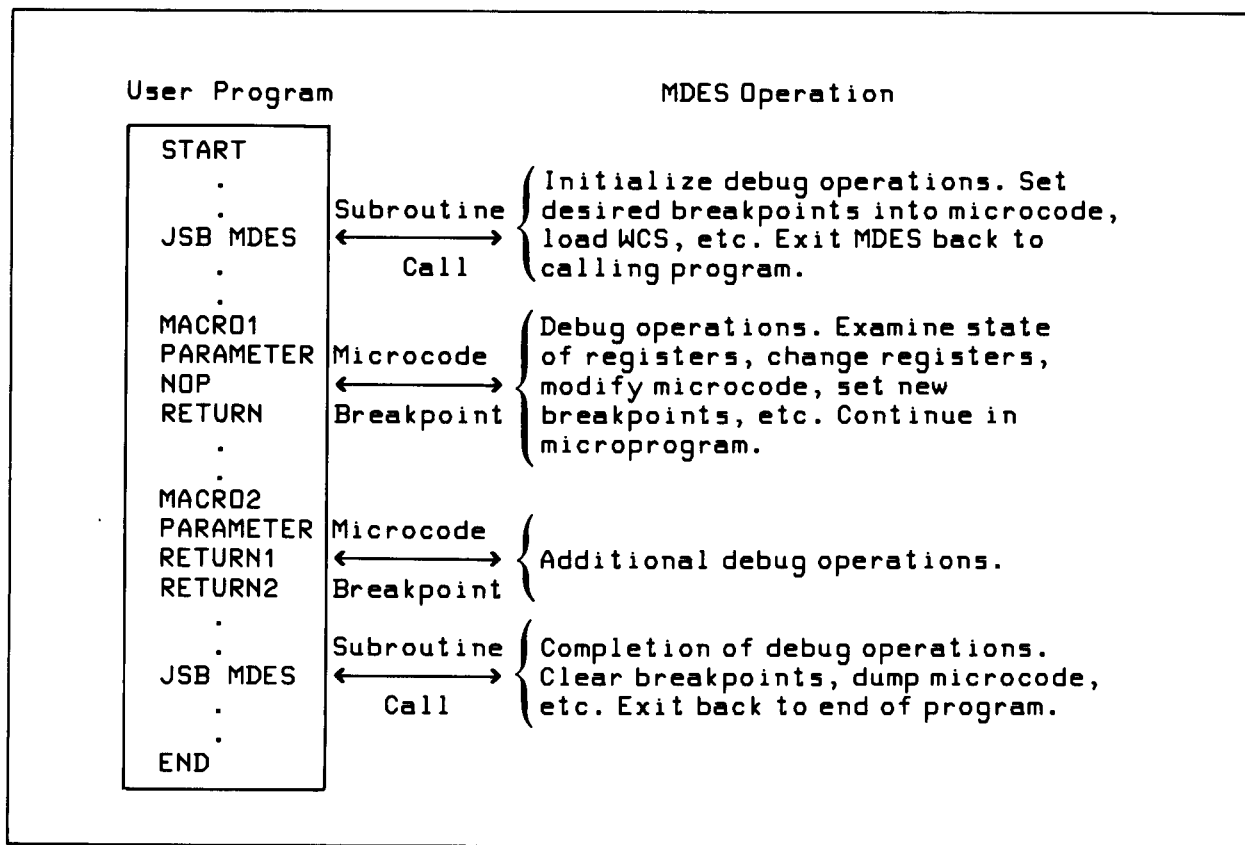
          CALL MDES(I1,I2,I3,I4,I5)

          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



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.

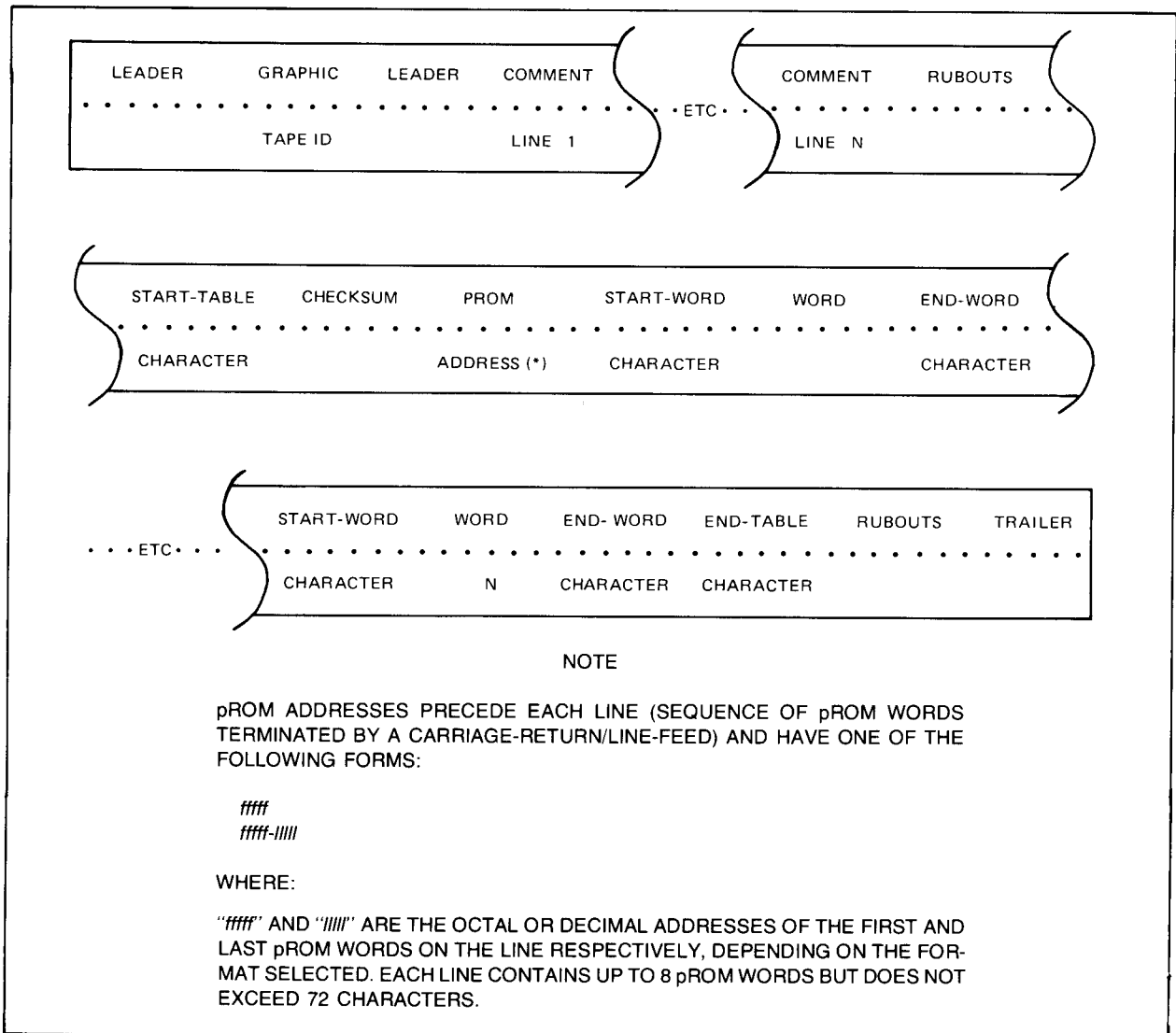


Figure 12-1. General Tape Format

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-aaaa (bb-bb)

where:

aaaaa-aaaa 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 start and end-table characters delimit the table, the checksum is punched immediately after the start-table character.

2.3 START-TABLE,END-TABLE CHARACTERS?

If start and 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 start and 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 high and 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:

<u>pROM PART</u>	<u>21MX</u>	<u>21MX E-SERIES</u>
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.

Generating pROM Tapes

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 1000 M/E/F-Series Firmware Installation and Reference 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 ($\overline{\text{BIO S}}$) 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 preceding 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)

SETPM			DEC	S3	P	SAVE P.
*		IOFF	INC	PNM	B	M = BUFFER ADDRESS,
*						P = NEXT BUFFER ADDRESS,
WAIT1	JMP	CNDX	H0I		INT1	HOLD OFF I/O INTERRUPTS.
SKPF1				PASS		EMERGENCY INTERRUPTS?
*				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	H0I		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.
			IDR	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
	JMP	ION	PASS	P	S3	FIX P, EXIT TO
					6B	TO H0RI ROUTINE
*						
*						
STCNTRL	IMM	L4	CMLD	L	303B	L=STC 0,C
	IMM		CMLD	S4	376B	S4=1
			IDR	S4	S4	S4=STC 1,C
FAKESC			PASS	IRCM	S4	IRCM=STC 1,C
			IDR	S4	X	S4=STC SC,C
REALSC			PASS	CNTR	S4	IRCM=STC SC,C
	RTN	IDG				
			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.
                                         105600 MAPS TO 34000B
                                         ORG                               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.
*

BRSTIN      JMP      ALGN      BRSTIN      SAVE ENTRY POINTS.
STCNTRL     IMM      L4        CMLD      L      303B      L=STC 0,C
            IMM                               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
BRSTADR     JMP      CNDX     E          BRSTDTA
            JMP      CNDX     HOI        INTADR      EMERGENCY INTERRUPTS?
            JMP      CNDX     PASS       INTERFACE FLAG SET?
            JMP      CNDX     SKPF      RJS        BRSTADR  NO, GO TO BRSTADR
            M          M          IOI        M = BUFFER ADDRESS.

*
BRSTDTA     JMP      CNDX     HOI        INTDTA      EMERGENCY INTERRUPTS?
            JMP      CNDX     PASS       INTERFACE FLAG SET?
            JMP      CNDX     SKPF      RJS        BRSTDTA  NO, GO TO BRSTDTA
            S4         S4         IOI        S4 = DATA
BRSTEND     WRTE      MPCK      TAB        S4         WRITE DATA INTO MEMORY.
            DEC      A          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      IMM                               ASG      PASS     P          S3         EXECUTE CLE OR CCE AND FIX P
            JMP      END                               6          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
$CODE=BI003
                                ORG          34000B
                                SPECIFIES E-SERIES OR F-SERIES
                                SAVE MICRO-OBJECT ON DISC.
                                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(S-0) = SC, IOFF HOLDS
*                                OFF I/O INTERRUPTS.
BURST1
                                IO          TAB          P          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         RTN         INC          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         END         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."
$\overline{\text{PP2SP}}$	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.

13-9. 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*):	1.59M words/second (maximum)
Output (256 words or less*):	1.36M words/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.

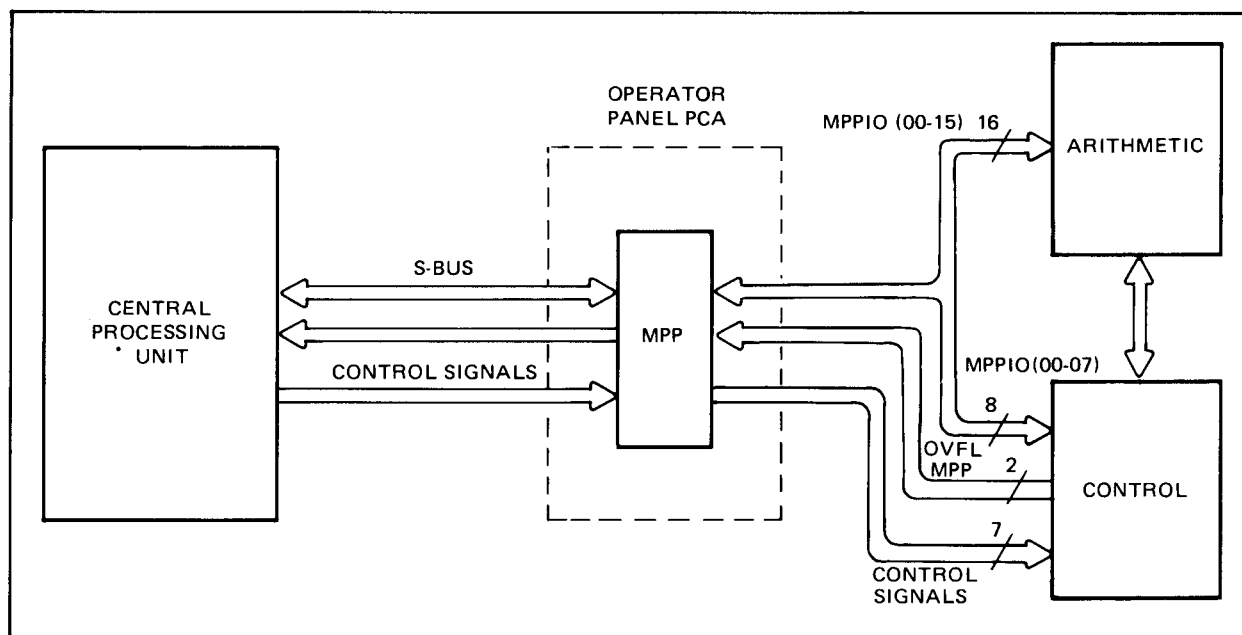
The Control section includes the hardware necessary to control the functions of the Arithmetic section.

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.



7700-220

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.

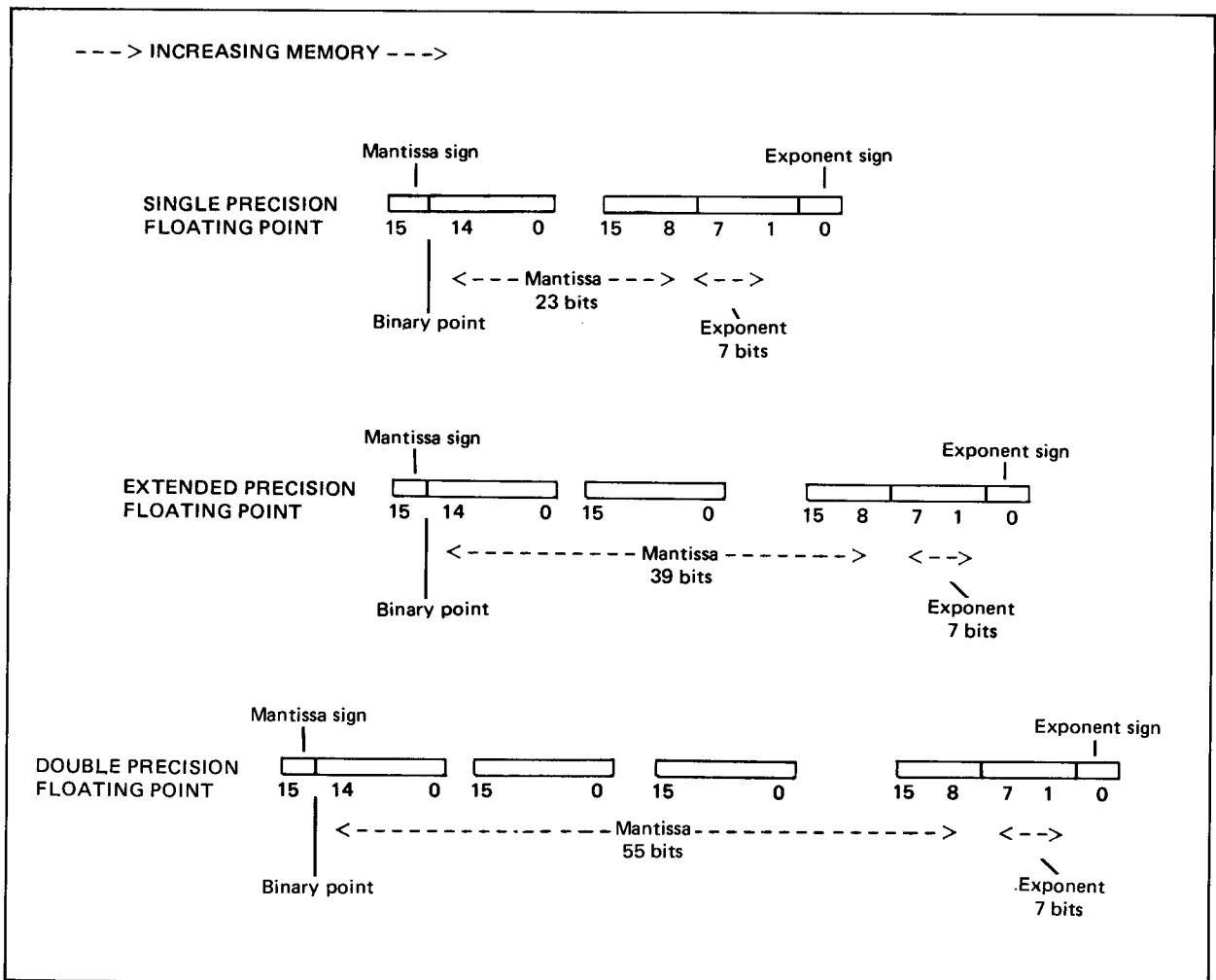


Figure 13-2. Floating Point Data Format

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}
Four-word	-2^{127} , $(1 - 2^{-55}) 2^{127}$	$-2^{-129} (1 + 2^{-54})$, 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}
Four-word	-2^{511} , $(1 - 2^{-55}) 2^{511}$	$-2^{-513} (1 + 2^{-54})$, 2^{-513}
<p>NOTE:</p> <ol style="list-style-type: none"> 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, three, and four-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 Four words 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, 48 bits, or 64 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, three, or four 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	Subtrahend
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

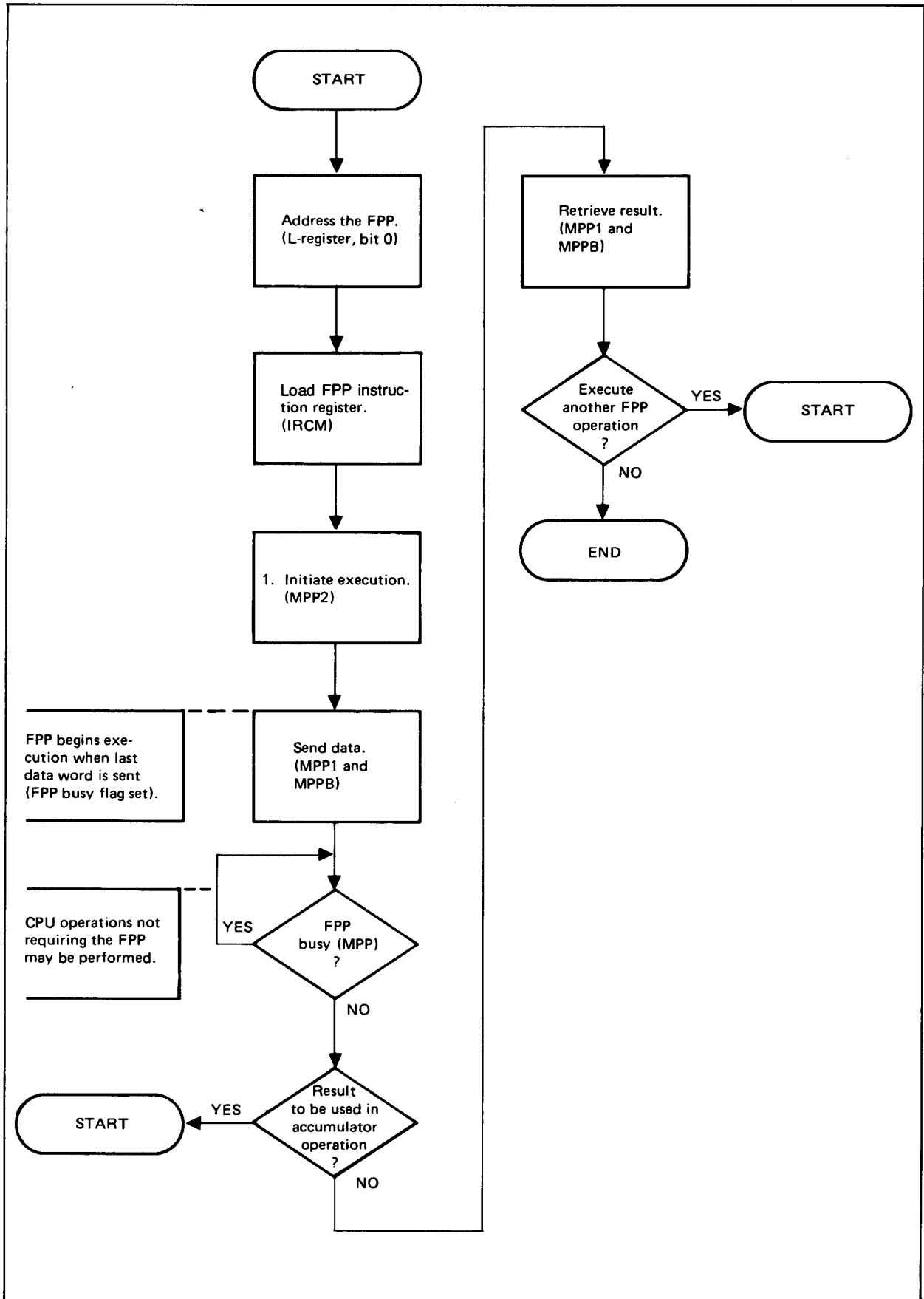
The following paragraphs describe the MPP micro-orders required to microprogram the FPP. Figure 13-4 illustrates the microprogramming sequence used to execute a typical FPP operation.

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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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.

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ 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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDR	COMMENT
			INC	PNM	P	GET RESULTANT DESTINATION ADDRESS.
	WRTE	MPP1		S4 TAB	MPPB S4	GET FIRST WORD OF RESULT FROM FPP. STORE FIRST WORD OF RESULT.
			INC	PNM	P	GET ADDRESS FOR SECOND WORD.
	WRTE	MPP1		S4 TAB	MPPB S4	GET SECOND WORD. STORE SECOND WORD.
			INC	PNM	P	
	WRTE	MPP1		S4 TAB	MPPB S4	GET LAST WORD.
			.			
			.			
			.			

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 (μ sec)		
	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
Double-precision Floating Point			
Add/Subtract	0.68	1.36	5.58
Multiply	3.55	4.08	4.62
Divide	3.68	6.60	9.51
Conversion to single integer	0.68	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:

LABEL	OP/ BRCH	SPCL	ALU/ MOD/ COND	STR	S-BUS/ ADDR	COMMENT
			.			
			.			
WAIT	IMM	CDV	CML0	S3	337B	SET WORD COUNT CONSTANT AND CLEAR OVERFLOW.
WAIT1	RTN	CNDX	MPP			IF FPP DONE, RETURN.
	JMP	CNDX	DEC	S3	S3	DECREMENT COUNTER
	JMP		AL15	RJS	WAIT1 ERRRUT	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 ADDR B	ADDRESS OF ARRAY B

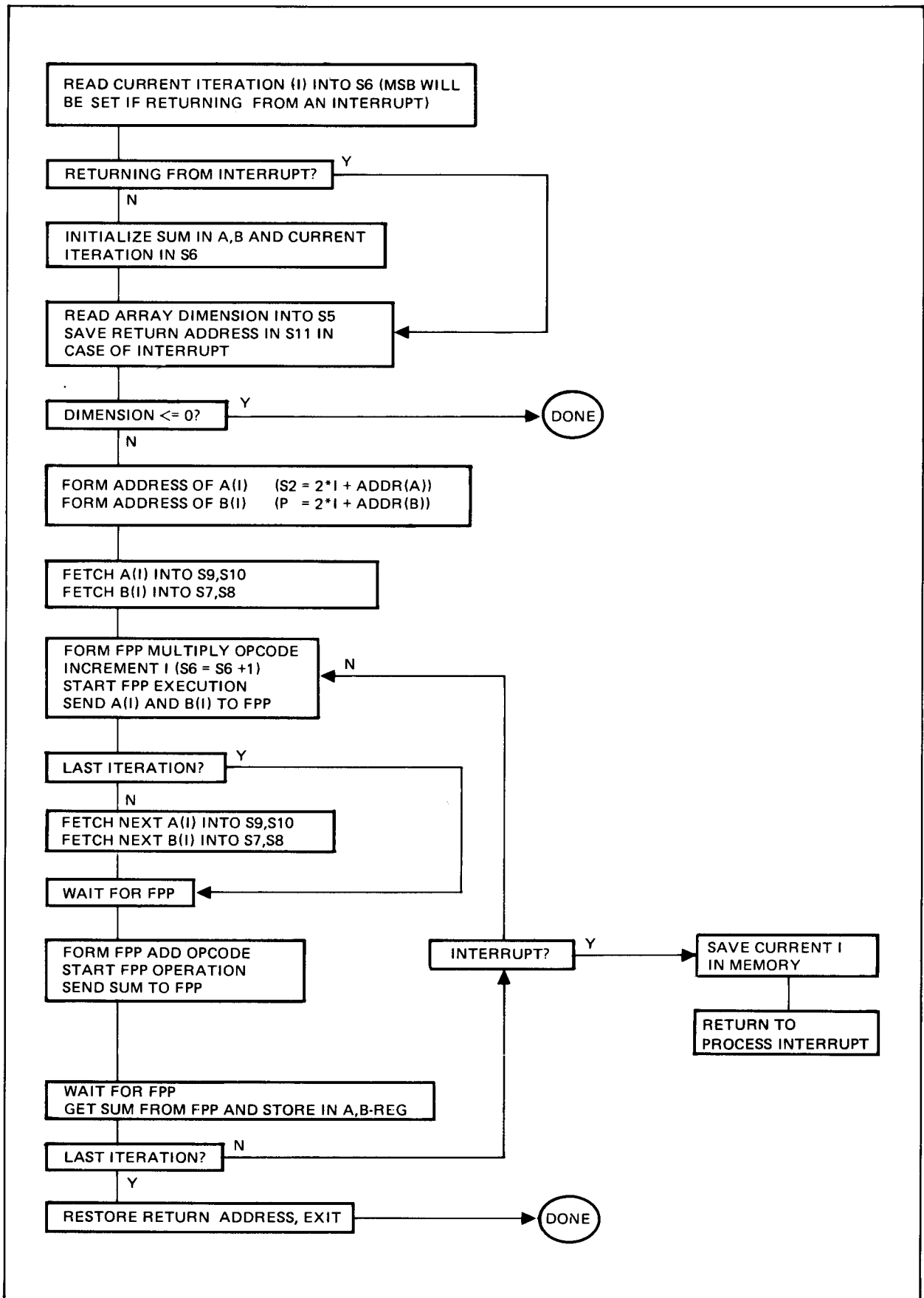
13-36. MICROPROGRAMMING THE FLOATING POINT PROCESSOR

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.

Special

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

```

0001          MICMXE,L
0002          #CODE='SUMAB:--48,REPLACE
0003          ORG          34000B
0004          HORI      EQU          00006B
0005          FETCH    EQU          00000B
0006          *
0007          *
0008          *
0009          * READ CURRENT ITERATION (I) INTO S6 *
0010          * (MSB WILL BE SET IF RETURNING FROM *
0011          * AN INTERRUPT) *
0012          *
0013          *
0014 34000 017 101254 SUMAB          SOV CMPS S6 TAB          S6 = CURRENT I
0015          *
0016          *
0017          * RETURNING FROM INTERRUPT? *
0018          *
0019          *
0020 34001 327 140242          JMP CNDX AL15 RJS REENT          JUMP IF RE-ENTERING
0021          *
0022          *
0023          * INITIALIZE SUM IN A,B AND CURRENT ITERATION IN S6 *
0024          *
0025          *
0026 34002 006 036147          ZERO A          INITIALIZE SUM
0027 34003 006 036207          ZERO B
0028 34004 006 037247          ZERO S6          INITIALIZE CURRENT I
0029          *
0030          *
0031          * READ ARRAY DIMENSION INTO S5 *
0032          * SAVE ADDRESS IN CASE OF INTERRUPT (S11) *
0033          *
0034          *
0035 34005 227 174707 REENT READ          INC PNM P          START READ OF IMAX
0036 34006 000 075507          DEC S11 P          SAVE P IN S11
0037 34007 307 005047          JSB          INDIRECT RESOLVE INDIRECTS
0038 34010 010 001207          PASS S5 TAB          S5 = IMAX
0039          *
0040          *
0041          * IMAX <= 0? *
0042          *
0043          *
0044 34011 327 104002          JMP CNDX AL15          DONE          FORGET IT IF IMAX<0
0045 34012 320 004002          JMP CNDX ALZ          DONE          DITTO IF IMAX=0
0047          *
0048          * FORM ADDRESS OF ARRAY A (S2 = 2*I + ADDR(A)) *
0049          * FORM ADDRESS OF ARRAY B (P = 2*I + ADDR(B)) *
0050          * NOTE THAT L0 = 0 WHEN ADDRESS IS FORMED *
0051          *
0052          *
0053 34013 227 174713          READ COV INC PNM P          START READ ON ADDR(A)
0054 34014 001 152507          DBLS L S6          SET L = 2 * I
0055 34015 307 005047          JSB          INDIRECT RESOLVE INDIRECTS
0056 34016 003 033047          ADD S2 M          S2 = ADDR(A(I))
0057 34017 227 174707          READ          INC PNM P          START READ ON ADDR(B)
0058 34020 307 005047          JSB          INDIRECT RESOLVE INDIRECTS
0059 34021 003 033707          ADD P M          P = ADDR(B(I))
0060          *
0061          *
0062          * FETCH B(I) INTO S7,S8 *
0063          * FETCH A(I) INTO S9,S10 *
0064          *
0065          *
0066 34022 227 174707          READ          INC PNM P          START READ ON B(I)
0067 34023 007 174707          INC PNM P          BUMP ADDR(B(I))
0068 34024 230 001307          READ          PASS S7 TAB          SAVE B(I) IN (S7 S8)
0069 34025 010 042647          PASS M S2
0070 34026 007 143047          INC S2 S2          BUMP ADDR(A(I))
0071 34027 230 001347          READ          PASS S8 TAB
0072 34030 010 042647          PASS M S2          SET M = ADDR(A(I))
0073 34031 007 143047          INC S2 S2          BUMP ADDR(A(I))
0074 34032 230 001407          READ          PASS S9 TAB          SAVE A(I) IN (S9 S10)
0075 34033 010 001447          PASS S10 TAB

```

```

0076
0077
0078
0079
0080
0081
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
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
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
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
0123
0124
0125 34060 306 044402 LOOP1 JSB CNDX MPP RJS WAIT WAIT FOR FPP
0126
0127
0128
0129
0130
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

```

Special

```

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 CMLD 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

```

```

0211          *
0212          *****
0213          * RESOLVE INDIRECT ADDRESSING *
0214          *****
0215          *
0216 34121 230 000647 INDIRECT READ      PASS M      TAB      SAVE ADDRESS IN M
0217 34122 367 140002          RTN  CNDX AL15 RJS      RETURN IF RESOLVED
0218 34123 323 145042          JMP  CNDX HOI  RJS  INDIRECT  KEEP RESOLVING
0219 34124 230 036747          READ                    BUT IF HOI AND
0220 34125 336 045042          JMP  CNDX NSNG RJS  INDIRECT  NO SINGLE STEP
0221 34126 000 065707          DEC  P      S11      RESTORE P AND
0222 34127 320 000307          JMP                    HORI      HANDLE INTERRUPT
0223          *
0224          *
0225          END

```

END OF PASS 2: NO ERRORS

PART IV

Microprogramming Examples

Section 14

MICROPROGRAMS



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 1000 E/F-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 your disc file, 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. Pseudo-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 that you designate 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=M-SERIES,2=E/F-SERIES
TYPE(1 OR 2)?2
$LU, 13
```

```
LU#      RANGE      STATUS
13 034000--034777 1
$RE, 34000B
34000 LGS STFL NAND SI 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
```

Microprograms

EXAMPLE 2: READ/WRITE MEMORY, CONSOLE RUN SHEET (Sheet 1 of 2)

*ON, FMGR
:RU, MDEP

COMPUTER TYPE: 1=M-SERIES, 2=E/F-SERIES
TYPE(1 OR 2)?2
SLU, 13

LU#	RANGE	STATUS			
13	034000--034777	1			
	<u>\$RE, 34000B, 34003B</u>				
34000	LGS	XOR	S3	X	
	<u>\$\$READ, NOP, PASS, L, A</u>				
34000	READ	PASS	L	A	
	<u>\$\$/</u>				
34001	STFL	CMPS	A	CNTR	
	<u>\$\$NOP, NOP, AND, S1, TAB</u>				
34001		AND	S1	TAB	
	<u>\$\$/</u>				
34002	STFL	PASS	S11	S1	
	<u>\$\$WRTE, MPCK, PASS, TAB, S1</u>				
34002	WRTE	MPCK	PASS	TAB	S1
	<u>\$\$/</u>				
34003	SRG1	CMPS		MEU	
	<u>\$\$READ, RTN, INC, PNM, P</u>				
34003	READ	RTN	INC	PNM	P
	<u>\$\$A</u>				
	<u>\$\$SH, 34000B, 34003B</u>				
34000	READ	PASS	L	A	
34001		AND	S1	TAB	
34002	WRTE	MPCK	PASS	TAB	S1
34003	READ	RTN	INC	PNM	P
	<u>\$\$E, A</u>				
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.

Microprograms

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     INTFGER 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          ASMR,L
0002 00000          NAM I2.1.7
0003*
0004*          SORT INTERFACE PROGRAM
0005*
0006          ENT SORT
0007          EXT .ENTR
0008 00000 000000  NMBR  RSS 1
0009 00001 000000  TABLE RSS 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**

```

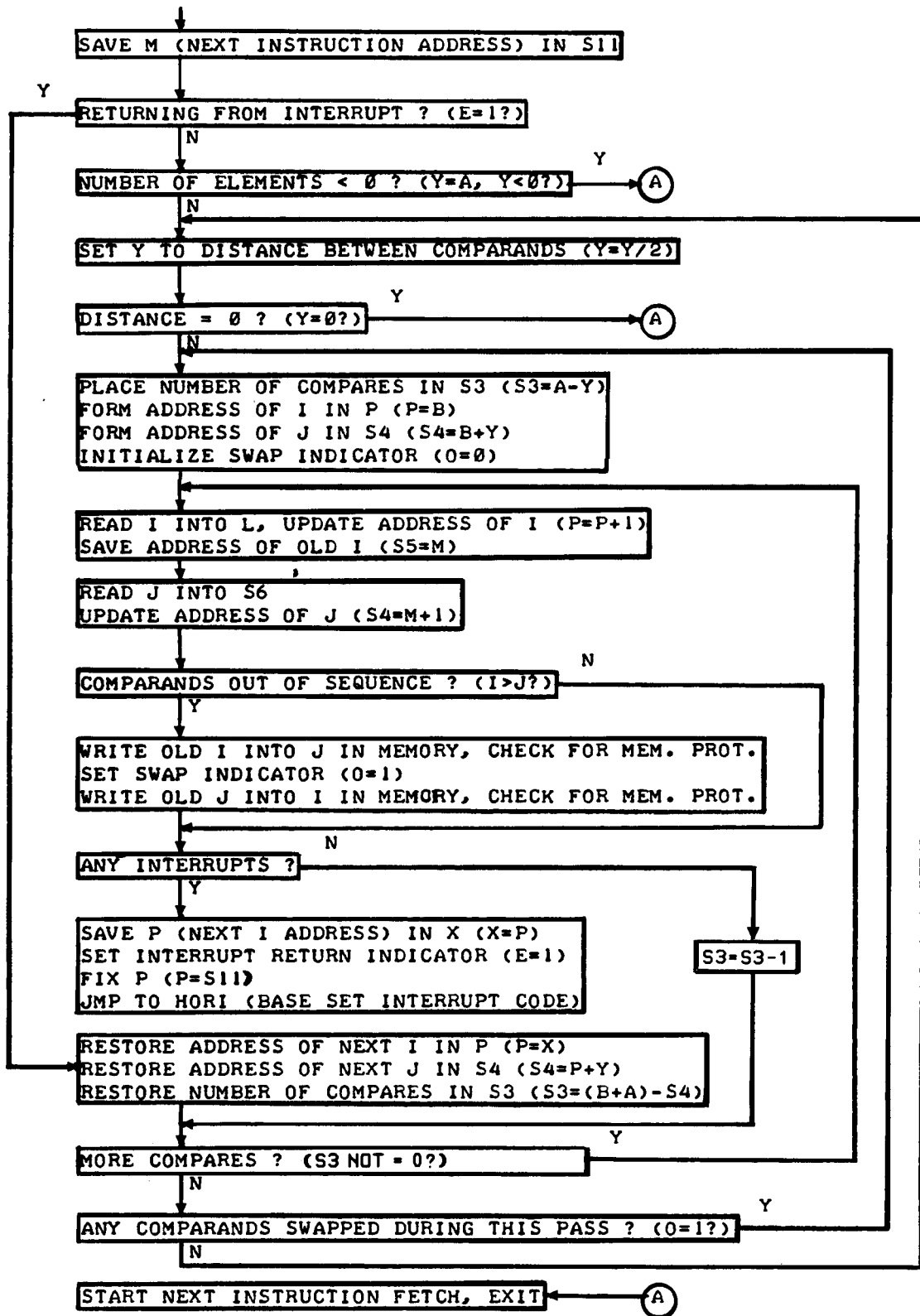


Figure 14-1. Example 3, Microprogrammed Shell Sort Flowchart

EXAMPLE 3: SHELL SORT; TEST, CONSOLE RUN SHEET (Sheet 1 of 2)

```

*ON,FMGR
:RU,EDITR ← CREATE MICROPROGRAM SOURCE FILE
SOURCE FILE?
/Δ
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

BODY OF MICROPROGRAM ← USER SELECTED MICROPROGRAM OBJECT FILENAME

/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
$SR,34052B,34072B ← LOCATE MDE BREAKPOINT MICROPROGRAM, AND
BREAK 1 34052 ← PROVIDE AN UNUSED ENTRY POINT FOR MDE USE,
BREAK 2 34072 ← BEFORE SETTING BREAKPOINTS
BREAK 3 0
$EX ← SET BREAKPOINT IN SWAP MICROINSTRUCTIONS, AND
START OF SORT ← SET BREAKPOINT AT END OF ONE COMPLETE PASS
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

```

Microprograms

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
136075 152742 016336 016440 023501 ← CORRECTLY SORTED DATA
$CL
BREAK 1 0
BREAK 2 0
BREAK 3 0
$EX
:EX
SEND FMGR
```

NOTE: THESE ARE NEGATIVE NUMBERS

BE SURE TO REMOVE BREAKPOINTS !

EXAMPLE 3: SHELL SORT, MICROPROGRAM (Sheet 1 of 3)

PAGE 0002 RTE MICRO-ASSEMBLER REV.A 760805

```

0001          MICMXE,L
0002
0003          ORG 34000B
0004          *****
0005          *
0006          *          LAB 2.1 MICROPROGRAM          *
0007          *
0008          * THIS MICROPROGRAM SORTS AN INTEGER ARRAY INTO
0009          * ASCENDING ORDER USING THE DIMINISHING INCREMENT
0010          * TECHNIQUE (I.E. SHELL SORT).
0011          * REF: ART OF COMPUTER PROGRAMMING, VOL 3.
0012          *
0013          * CALLING SEQUENCE
0014          *     LDA NMBR    + NUMBER OF SORT ELEMENTS
0015          *     LDB TABLE  ADDRESS OF FIRST ELEMENT
0016          *     CLE         E=(0=INITIAL ENTRY,
0017          *                   I=RETURN FROM INTERRUPT)
0018          *     OCT 105600  INVOKE SORT MICROPROGRAM
0019          *
0020          * AT END
0021          *     CONTENTS OF TABLE SORTED
0022          *     A,B UNALTERED  E,0 MAY BE ALTERED  X,Y 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          HORI      EQU 6B
0031 34000 327 001007  JMP          SORT          SAVE ENT POINTS
0032          ALGN
0033          *****
0034          * SAVE M (NEXT INSTRUCTION ADDRESS) IN S11 *
0035          *****
0036 34020 010 033507  SORT          S11 M          S11 = NEXT
0037          *****          INSTR ADDR
0038          * RETURNING FROM INTERRUPT ? (E=1?) *
0039          *****
0040 34021 334 103042  JMP CNDX E          INTRTN  YES, USE INTRTN
0041          *****
0042          * NUMBER OF ELEMENTS < 0 ? (Y=A, Y<0?) *
0043          *****
0044 34022 010 007647          Y      A          Y = A
0045 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

```

0047 *****
0048 * SET Y TO DISTANCE BETWEEN COMPARANDS (Y=Y/2) *
0049 *****
0050 34024 010 073664 SETY R1 Y Y Y = Y/2
0051 *****
0052 * DISTANCE = 0 ? (Y=0?) *
0053 *****
0054 34025 010 072747 Y
0055 34026 320 003602 JMP CNDX ALZ EXIT Y=0 ? YES, EXIT
0056 *****
0057 * PLACE NUMBER OF COMPARES IN S3 (S3=A-Y) *
0058 * FORM ADDRESS OF I IN P (P=B) *
0059 * FORM ADDRESS OF J IN S4 (S4=B+Y) *
0060 * INITIALIZE SWAP INDICATOR (O=0) *
0061 *****
0062 34027 010 072507 STRTPASS L Y
0063 34030 004 107107 SUB S3 A S3 = COMPARES
0064 34031 010 011707 P B P = ADDR OF I
0065 34032 003 011153 COV ADD S4 B S4 = ADDR OF J,
0066 * O=0
0067 *****
0068 * READ I INTO L, UPDATE ADDRESS OF I (P=P+1) *
0069 * SAVE ADDRESS OF OLD I (S5=M) *
0070 *****
0071 34033 227 174707 COMPARE READ INC PNM P READ I, UPDATE P
0072 34034 010 033207 S5 M S5 = ADDR OF I
0073 34035 010 000507 L TAB L = I
0074 *****
0075 * READ J INTO S6 *
0076 * UPDATE ADDRESS OF J (S4=M+1) *
0077 *****
0078 34036 230 046647 READ M S4 READ J
0079 34037 007 133147 INC S4 M S4 = NEXT J ADDR
0080 34040 010 001247 S6 TAB S6 = J
0081 *****
0082 * COMPARANDS OUT OF SEQUENCE ? (I>J?) *
0083 *****
0084 34041 014 152747 XOR S6 J SIGN = I SIGN?
0085 34042 327 142302 JMP CNDX AL15 RJS SUBTRACT YES, SUBTRACT
0086 34043 012 136747 PASL I SIGN = - ?
0087 34044 327 102602 JMP CNDX AL15 INTCHK YES, NO SWAP
0088 34045 327 002407 JMP SWAP NO, SWAP
0089 34046 004 152747 SUBTRACT SUB S6 J = I < 0 ?
0090 34047 327 142602 JMP CNDX AL15 RJS INTCHK NO, NO SWAP
0091 *****
0092 * WRITE OLD I INTO J IN MEMORY, CHECK FOR MEM. PROT. *
0093 * SET SWAP INDICATOR (O=1) *
0094 * WRITE OLD J INTO I IN MEMORY, CHECK FOR MEM. PROT. *
0095 *****
0096 34050 012 137307 SWAP PASL S7 S7 = OLD I
0097 34051 210 054036 WRTE MPCK TAB S7 J IN MEM = OLD I
0098 34052 007 150654 SOV INC M S5 M=ADDR OF I, O=1
0099 34053 210 052036 WRTE MPCK TAB S6 I IN MEM = OLD J
0100 * O=1

```

EXAMPLE 3: SHELL SORT, MICROPROGRAM (Sheet 3 of 3)

PAGE 0004 RTE MICRO-ASSEMBLER REV.A 760818

```

0102          *****
0103          * ANY INTERRUPTS ? *
0104          *****
0105 34054   323 143502 INTCHK   JMP  CNDX HOI  RJS  ENDCHK           NO, CHK PASS
0106          *****
0107          * SAVE P (NEXT I ADDRESS) IN X (X=P)           *
0108          * SET INTERRUPT RETURN INDICATOR (E=1)         *
0109          * FIX P (P=S11)                                 *
0110          * JMP TO HORI (BASE SET INTERRUPT CODE)        *
0111          *****
0112 34055   010 075607 INTEXIT          X    P           X = NEXT I ADDR
0113 34056   342 000607          IMM      LOW  IFCM 200B         IR(9-6)=1110=ELA
0114 34057   011 136761                   SRG1 ONE           I.E. SET E
0115 34060   010 065707                   P    S11           FIX P,
0116 34061   320 000307          JMP      HORI           JMP TO BASE SET
0117          *                                           INTERRUPT CODE
0118          *****
0119          * RESTORE ADDRESS OF NEXT I IN P (P=X)         *
0120          * RESTORE ADDRESS OF NEXT J IN S4 (S4=P+Y)     *
0121          * RESTORE NUMBER OF COMPARES IN S3 (S3=B+A-S4) *
0122          *****
0123 34062   010 071707 INTRTN          P    X           P = NEXT I ADDR
0124 34063   010 072507                   L    Y           S4 = NEXT J ADDR
0125 34064   003 075147          ADD   S4  P           S4 = NEXT J ADDR
0126 34065   010 006507                   L    A           S4 = NEXT J ADDR
0127 34066   003 011107          ADD   S3  B           S3 = B+A
0128 34067   010 046507                   L    S4           S3 = B+A
0129 34070   004 145107          SUB   S3  S3           S3 = (B+A)-S4 =
0130 34071   327 003547          JMP      *+2           COMPARES
0131          *****
0132          * MORE COMPARES ? (S3=S3-1, S3 NOT =0?) *
0133          *****
0134 34072   000 045107 ENDCHK          DEC   S3  S3           MORE COMPARES ?
0135 34073   320 041542          JMP  CNDX ALZ  RJS  COMPARE   YES, DO NEXT
0136          *****
0137          * ANY COMPAREANDS SWAPPED DURING THIS PASS ? (O=1?) *
0138          *****
0139 34074   335 101342          JMP  CNDX OVFL  STRTPASS     YES, REDO PASS
0140 34075   327 001207          JMP      SETY           NO, NEXT PASS
0141          *****
0142          * START NEXT INSTRUCTION FETCH, EXIT *
0143          *****
0144 34076   227 164700 EXIT    READ RTN  INC  PNM  S11   START NEXT
0145          END                                           INSTR FETCH
END OF PASS 2: NO ERRORS

```

Microprograms

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     INTFGER TABLE(125)
0005     EQUIVALENCE (CONS,P(1)),(NMBR,P(2)),(PRINT,P(3))
0006     DATA NAME/2HNS,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          ASMR,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*     LDR TABLE  ADDRESS OF FIRST ELEMENT *
0015*     CLE         NOT REQUIRED FOR THIS PROGRAM. *
0016*                INCLUDED FOR COMPATIRILITY WITH *
0017*                THE MICROPROGRAM CALL *
0018*     JSR 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

```

Microprograms

EXAMPLE 3: SHELL SORT, ASSEMBLER SORT (Sheet 2 of 2)

PAGE 0003 #01

```

0033 00000 000000  NMRR  BSS 1
0034 00001 000000  TABLE BSS 1
0035 00002 000000  SORT  NOP
0036 00003 016001X  JSB  .ENTR  GET PARAMETERS
0037 00004 000000R  DEF  NMRR
0038 00005 162000R  LDA  NMRR,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  LDH  NMRR,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  STR  IPTR  IPTR = ADDRESS OF "I"
0052 00023 046057R  ADB  DSTNC
0053 00024 076062R  STR  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  SUR  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
START OF SORT ← CONSOLE LU, NUMBER OF SORT ELEMENTS
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
START OF SORT ← CONSOLE LU, NUMBER OF SORT ELEMENTS
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 (DV A_{xx}) is shown as it would appear "normally", then the microprogram enhanced driver (DVM x_{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 IBUF(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,IBUF,IBUFL)
0012  C
0013  C DISPLAY INPUT DATA
0014  WRITE (CONS,100) IBUF
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 x_{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 x_{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 =B16777 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 EQ4 50
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  EX1  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**

```

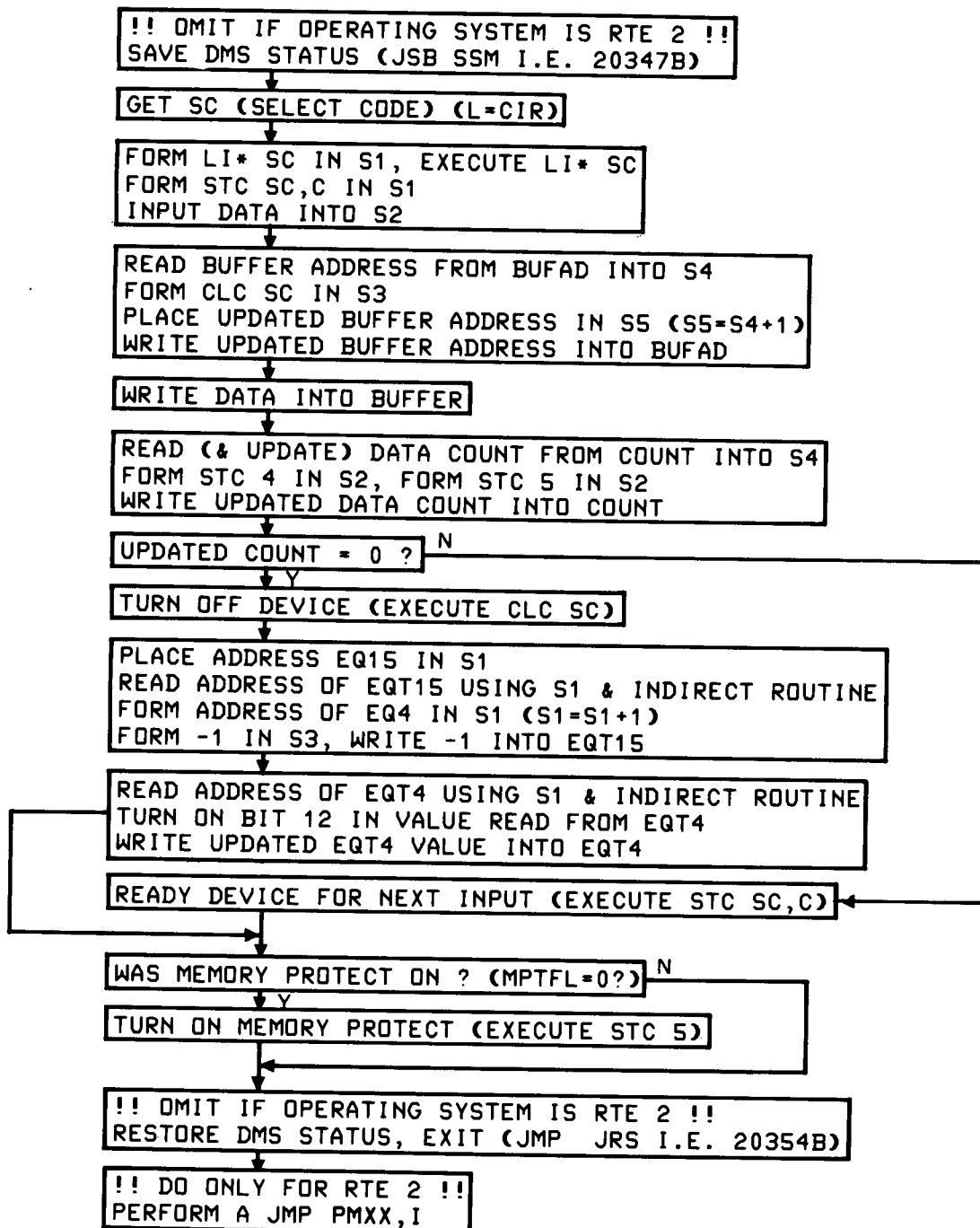


Figure 14-2. Example 4, Microprogrammed Privileged Section Flowchart

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

BODY OF MICROPROGRAM

/ELC&M3.1E ← USER SELECTED MICROPROGRAM SOURCE FILENAME
LS FILE 2 33
END OF EDIT
:RU,MICRO,&M3.1E,-6, ← 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 760818

```

0001          MICMXE,L                      21MX E/F-SERIES
0002          $CODE='MDRVH,REPLACE        OBJECT TO DISC
0003          *****
0004          *
0005          * SAMPLE PRIVILEGED SECTION MICROPROGRAM FOR DVMXX *
0006          *
0007          *****
0008          ORG 34000B                    105600 => 34000
0009          HORI EQU 6B
0010          INDIRECT EQU 251B
0011          SSM EQU 20345B
0012          JRS EQU 20352B
0013 34000 327 001007          JMP 34020B SAVE ENTRY
0014          ALGN                          POINTS
0015          *****
0016          * !! OMIT IF OPERATING SYSTEM IS RTE 2 !! *
0017          * SAVE DMS STATUS (JSH SSM I.E. 20347B) *
0018          *****
0019 34020 230 036747          READ SSM EXPECTS A
0020 34021 304 016247          JSH SSM READ OF DMSTS
0021 34022 000 075707          DEC P P SSM INC'S P 1
0022          ***** TOO MANY FOR US
0023          * GET SC (SELECT CODE) (L=CIR) *
0024          *****
0025 34023 010 024507          L CIR L = SELECT CODE
0026          *****
0027          * FORM LI* SC IN S1, EXECUTE LI* SC *
0028          * FORM STC SC,C IN S1 *
0029          * INPUT DATA INTO S2 *
0030          *****
0031 34024 357 175007          IMM CMHI S1 376B S1 = 400 = LI* 0
0032 34025 010 141007          IOR S1 S1 S1 = LI* SC
0033 34026 010 040607          INCM S1 EXECUTE LI* SC
0034 34027 010 036746          IOG
0035 34030 353 007023          IMM L4 CML0 S1 303B S1=1700=STC 0,C
0036 34031 010 141007          IOR S1 S1 S1 = STC SC,C
0037 34032 010 013047          S2 IOI S2 = DATA
0038          *****
0039          * READ BUFFER ADDRESS FROM BUFAD INTO S4 *
0040          * FORM CLC SC IN S3 *
0041          * PLACE UPDATED BUFFER ADDRESS IN S5 (S5=S4+1) *
0042          * WRITE UPDATED BUFFER ADDRESS INTO BUFAD *
0043          *****
0044 34033 227 174707          READ INC PNM P READ BUF ADDR
0045 34034 351 107123          IMM L4 CML0 S3 143B S3=4700=CLC 0
0046 34035 010 145107          IOR S3 S3 S3 = CLC SC
0047 34036 010 001147          S4 TAB S4 = BUF ADDR
0048 34037 007 147207          INC S5 S4 S5 = NEXT ADDR
0049 34040 210 050007          WRTE TAB S5 UPDATE BUF ADDR
0050          *****
0051          * WRITE DATA INTO BUFFER *
0052          *****
0053 34041 010 046647          M S4 M = BUF ADDR
0054 34042 210 042007          WRTE TAB S2 WRITE DATA

```

EXAMPLE 4: DRIVER MICROPROGRAMMED PRIVILEGED SECTION (Sheet 2 of 3)

PAGE 0003 RTE MICRO-ASSEMBLER REV.A 760818

```

0056
0057
0058
0059
0060
0061 34043 227 174707
0062 34044 350 073062
0063 34045 007 143047
0064 34046 007 101147
0065 34047 210 046007
0066
0067
0068
0069 34050 320 043402
0070
0071
0072
0073 34051 010 044607
0074 34052 010 036746
0075
0076
0077
0078
0079
0080
0081 34053 343 172507
0082 34054 004 175007
0083 34055 230 040647
0084 34056 300 012477
0085 34057 007 141007
0086 34060 343 177107
0087 34061 210 044007
0088
0089
0090
0091
0092
0093 34062 230 040647
0094 34063 300 012477
0095 34064 347 136507
0096 34065 011 001007
0097 34066 210 040007
0098 34067 327 003507
0099
0100
0101
0102 34070 010 040607
0103 34071 010 036746

*****
* READ (& UPDATE) DATA COUNT FROM COUNT INTO S4 *
* FORM STC 4 IN S2, FORM STC 5 IN S2 *
* WRITE UPDATED DATA COUNT INTO COUNT *
*****
      READ      INC  PNM  P      READ DATA COUNT
      IMM L1    CMLO S2  35B    S2 = 704 = STC 4
                               INC  S2  S2      S2 = 705 = STC 5
                               INC  S4  TAB    S4 = NEW COUNT
      WRTE      TAB  S4      WRITE NEW COUNT
*****
* UPDATED COUNT = 0 ? *
*****
      JMP  CNDX ALZ  RJS  STC      NO, STC SC,C
*****
* TURN OFF DEVICE (EXECUTE CLC SC) *
*****
                               IRCM S3      EXEC CLC SC
      IOG
*****
* 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 *
*****
      IMM      LOW  L    375B    L = 17775 = -3
                               SUB  S1  P      S1 = EQ15 ADDR
      READ      M      S1      GET EQT15 ADDR
      JSB  IOFF      INDIRECT
                               INC  S1  S1      S1 = ADDR OF EQ4
      IMM      LOW  S3  377B    S3 = 17777 = -1
      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 *
*****
      READ      M      S1      READ EQT4
      JSB  IOFF      INDIRECT
      IMM      HIGH L    357B    L = 16777
                               SONL S1  TAB    TURN ON BIT 12
      WRTE      TAB  S1      EQT4 BIT 12 = 1
      JMP      MPSTAT      CHK MEM. PROT.
*****
* READY DEVICE FOR NEXT INPUT (EXECUTE STC SC,C) *
*****
      STC      IRCM S1      EXEC STC SC,C
      IOG

```

EXAMPLE 4: DRIVER MICROPROGRAMMED PRIVILEGED SECTION (Sheet 3 of 3)

PAGE 0004 RTE MICRO-ASSEMBLER REV.A 760818

```

0105
0106
0107
0108 34072 227 174707 MPSTAT READ INC PNM P READ MPTFL
0109 34073 300 012477 JSB IOFF INDIRECT
0110 34074 010 000743 ION TAB MPTFL = 0 ?
0111 34075 320 043742 JMP CNDX ALZ RJS **2 NO, LEAVE
0112 * MEM. PROT. OFF
0113 *****
0114 * TURN ON MEMORY PROTECT (EXECUTE STC 5) *
0115 *****
0116 34076 010 042607 IRCM S2 EXEC STC 5
0117 34077 010 036746 IOG
0118 *****
0119 * !! OMIT IF OPERATING SYSTEM IS RTE 2 !! *
0120 * RESTORE DMS STATUS, EXIT (JMP JRS I.E. 20354B ) *
0121 *****
0122 34100 227 174707 READ INC PNM P JRS EXPECTS A
0123 34101 324 016507 JMP JRS READ OF DMSTS
0124 *****
0125 * !! DO ONLY FOR RTE 2 !! *
0126 * PERFORM A JMP PMXX,I *
0127 *****
0128 * INC P P P => DEF PMXX,I
0129 * READ INC PNM P READ PMXX ADDR
0130 * JSB IOFF INDIRECT
0131 * READ MPCK INC P M JMP PMXX,I
0132 * RTN ION
0133 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)
FEM	Firmware Expansion Module (12791A 8K 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 (micro-programming 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	X	X	ADDRESS	COMMENTS
	FIELD 1	FIELD 2	FIELD 3	FIELD 4	FIELD 5	FIELD 6	FIELD 7
	1	10	15	20	25	30	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
0001	10000	00100	00100	10010
23	19	14	9	4

Binary Object Code

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	ALO	CMHI	A		A
00100	NRM	**RJ30	OP8	LO		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 completeness 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:

$$\text{Inhibit Store} = \text{JTAB}[(\text{IR}14 + \text{IR}13 + \text{IR}12) \text{AL}15 + \text{IR}14 \cdot \text{IR}12 \cdot \text{IR}11 + \text{IR}14 \cdot \text{IR}13 \cdot \text{IR}12 + \text{IR}14 \cdot \text{IR}13 \cdot \text{IR}11]$$

- d. Turns on the Interrupt Enable flip-flop.

- e. Initializes the microsubroutine Save Stack address logic.
- f. Inhibits the clocking of the A- and B-Addressable flip-flops (AAF and BAF).

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 instruction 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 through 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 MAC0 (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 MACTABL1).

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.

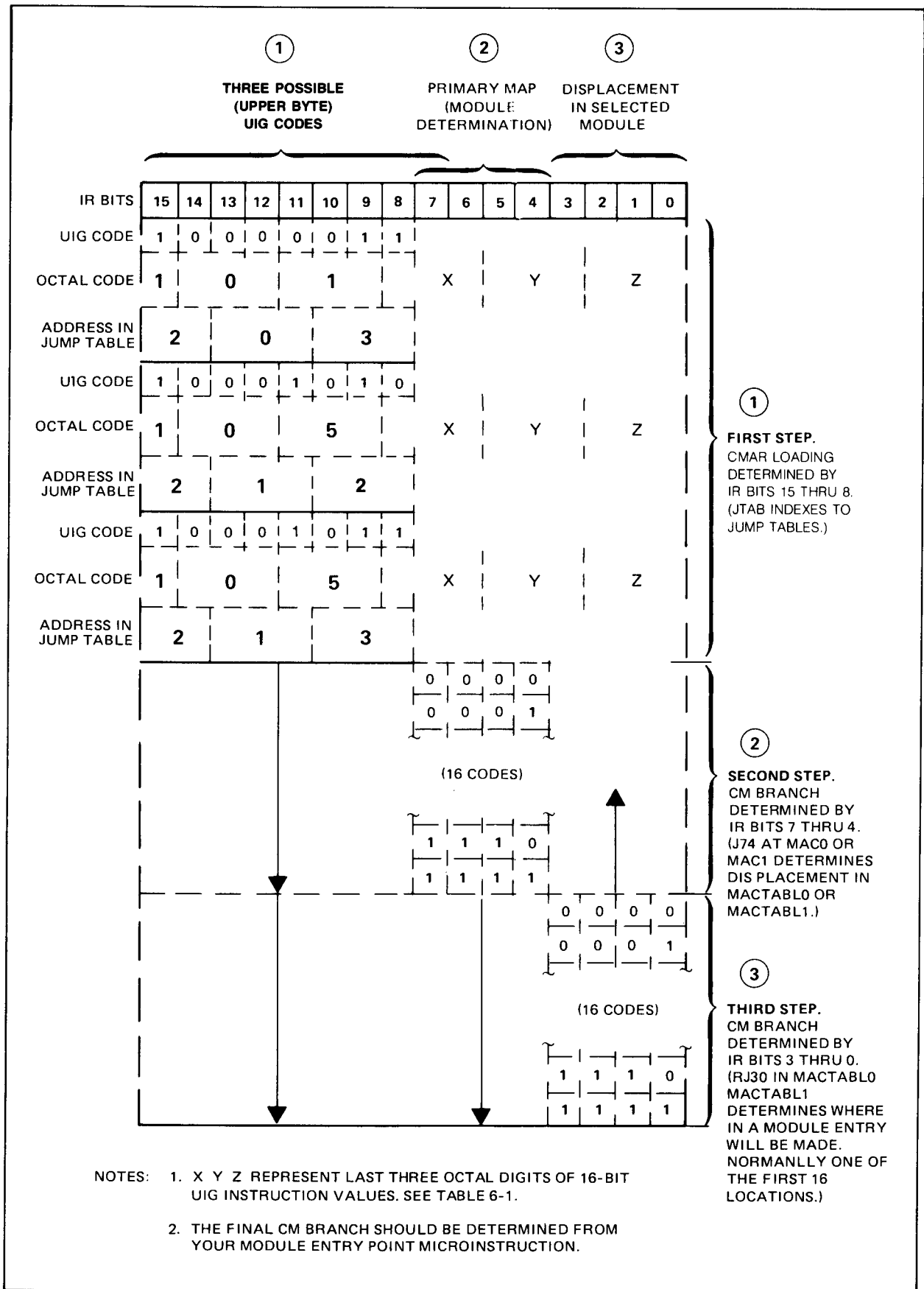


Figure C-1. UIG Instruction Bit Decoding

Appendix D
MICROPROGRAMMING FORM



Appendix E

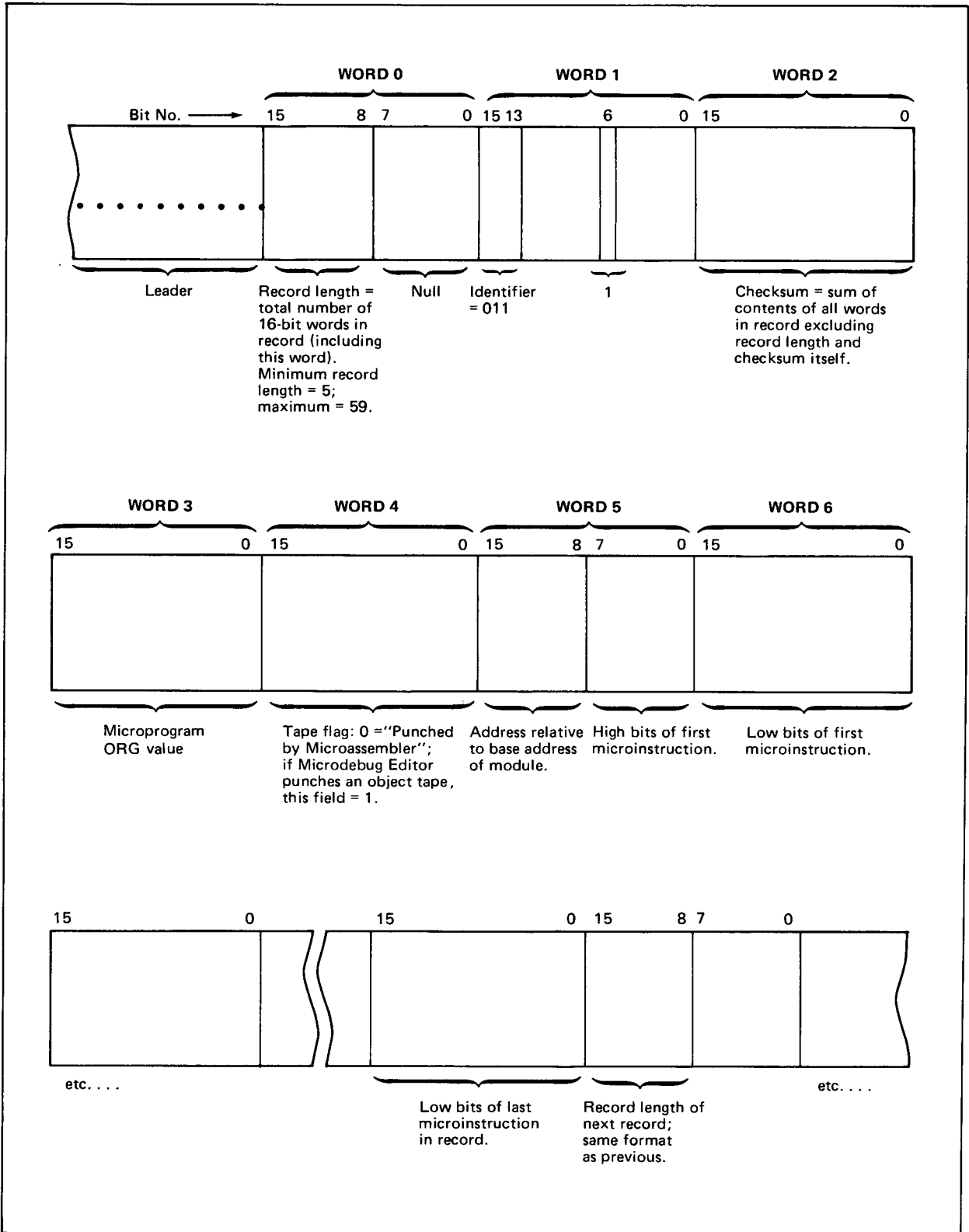
OBJECT TAPE FORMATS



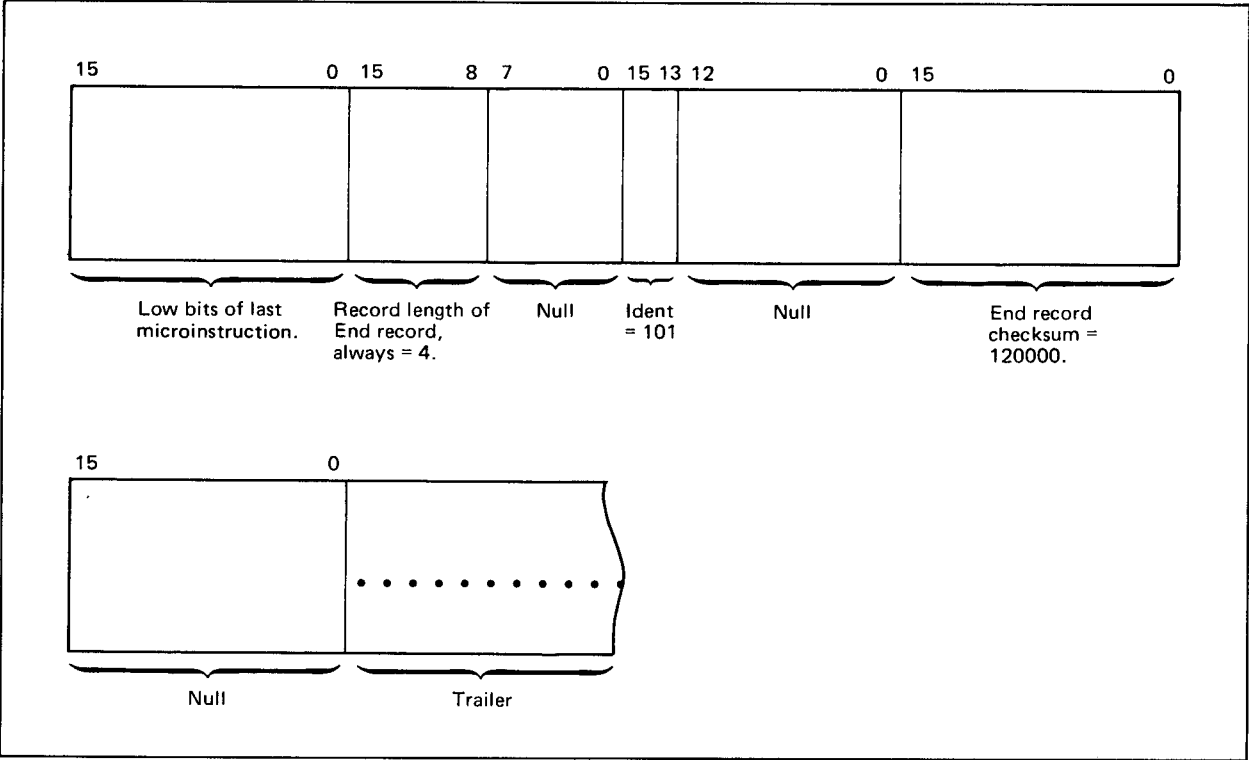
OBJECT TAPE FORMATS

APPENDIX

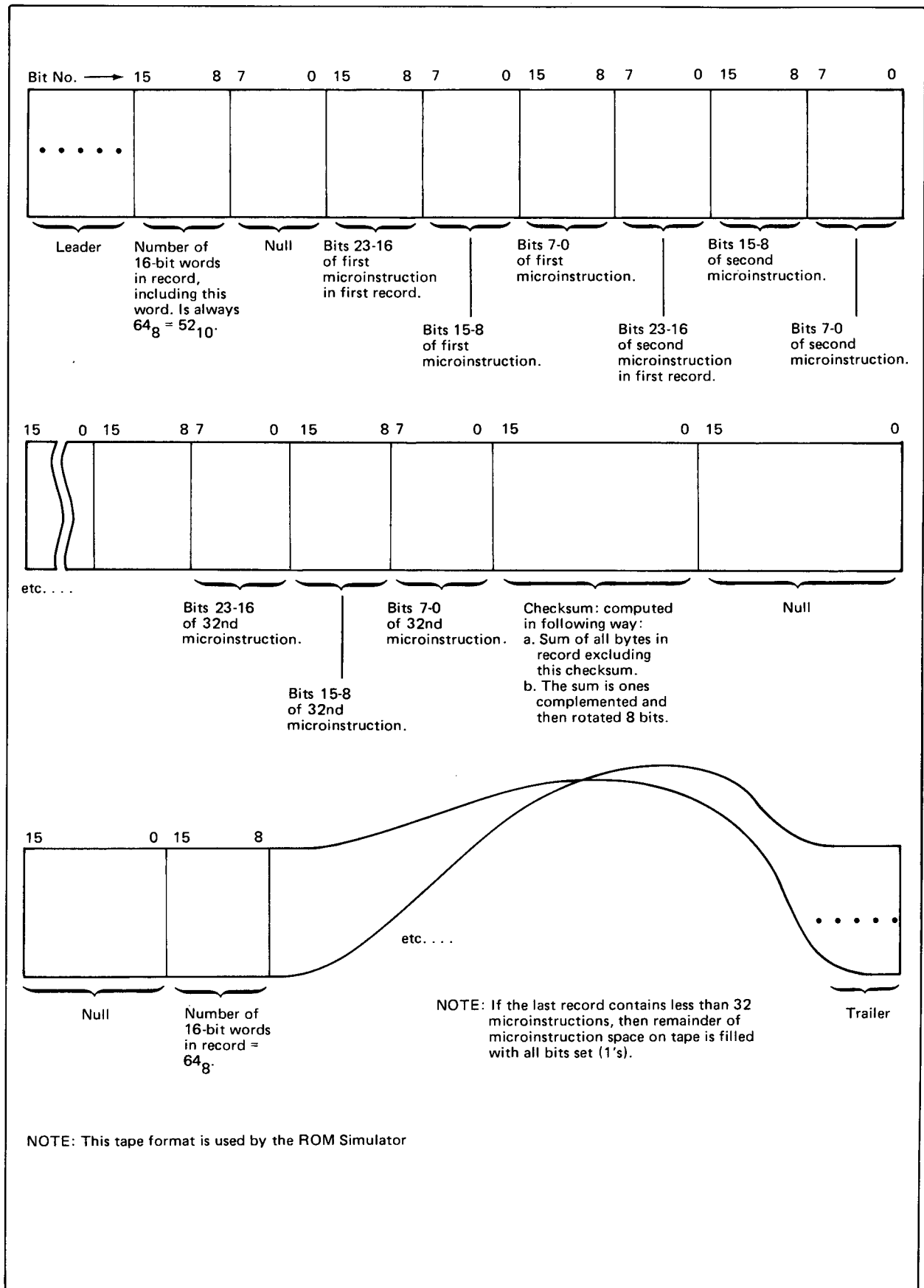
E



Appendix E



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		BRANCHSENSE 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	CMLP	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	CMLP	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=%E04 04,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          RTN          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

```

Appendix G

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        MACTABLO
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 STPL 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 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 PRESIFT 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 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

```

Appendix G

```

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          *
0225          *      EAU/MACTABLE 1 000 000 0
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          *
0238          *      UNIVERSAL INDIRECT OPERAND ROUTINE
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          *
0249          *      EAU/MACTABLE 1 000 001 0
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 POSITION 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

```

Appendix G

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 STOPE 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 CRG 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 UPDATAP 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

```

Appendix G

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          RPT 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 PCRT CONTROLLER WORD 1
0494 00603 300 030747          JSB CPTTEST 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=OFIG. 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      XCR 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     XNCR DIFFERENT PATTERN
0536 00646 320 076602      JMP  CNDX ALZ  RJS  FAILURE+2 TEST FOR NON-ZEROS
0537 00647 003 064747          ADD          S11     ADD UNLIKE PATTEPNS
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

```

Appendix G

```

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 CPTTEST TEST CENTRAL PROCESSOR
0544 *
0545 00655 006 037253 RIPPLMW 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 SRG1 PASS S DSPL LOADER FIND HOW MUCH MEMORY AVAILABLE
0558 00672 010 015761 PASS S3 M RESTORE S. CLEAR EXTEND
0559 00673 010 033107 JMP CNDX L15 TESTDMS S3 := TOP OF ENABLED MEMORY
0560 00674 322 134402 IMM CMLO S2 376B TEST FOR PRESENT MEMORY
0561 00675 353 175047 IMM LOW A 366B BACKGROUND PATTERN := 000001
0562 00676 343 154147 JSB RIPP32K TEST PATTERN := 177766
0563 00677 300 035107 IMM CMHI S2 177B TEST #1
0564 00700 355 177047 IMM CMLO A 366B BACKGROUND PATTERN := 100000
0565 00701 353 154147 JSB RIPP32K TEST PATTERN := 000011
0566 00702 300 035107 IMM LOW S2 377B TEST #2
0567 00703 343 177047 IMM CMHI A 177B BACKGROUND PATTERN := 177777
0568 00704 355 176147 JSB RIPP32K TEST PATTERN := 100000
0569 00705 300 035107 IMM PASS S2 S5 TEST #3
0570 00706 010 051047 JSB RIPP32K BACKGROUND PATTERN := S5 (DSPL)
0571 00707 300 035107 TESTDMS PASS MEU MEU TEST #4
0572 00710 010 022447 PASS MEU MEU ENABLE MEM STATUS REGISTER
0573 00711 010 022747 JMP CNDX CNES *-6 TEST IF DMS IS PRESENT
0574 00712 320 135002 INC S DSPL S := DISPLAY REGISTER
0575 00713 007 115747 IMM CMLO L 337B L := 40
0576 00714 353 076507 XOR DSPL S DISPLAY REGISTER := S
0577 00715 014 176307 JMP CNDX ALZ RJS DMSLOAD TEST FOR ALL MEMORY TESTED
0578 00716 320 073042 IMM HIGH MEU 100B DISABLE DMS MAPS
0579 00717 345 000447 SHLT PASS S S5 RESTORE S
0580 00720 010 051775 RTN DEC P S7 RESTORE P AND EXIT
0581 00721 360 055707

PAGE 0022 RTE MICRO-ASSEMBLER REV.A 760818
0583 00722 000 044735 RIPPLMW 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 RIPLOP-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                23420E    2000 ACCESS SYSTEM
0622 00761 324 161007 JMP                23420E    2000 ACCESS SYSTEM
0623 00762 325 140004 JMP RJ30                27000B
0624 00763 324 160004 JMP RJ30                23400E    2000 ACCESS SYSTEM
0625 00764 325 160004 JMP RJ30                27400E
0626 00765 326 000004 JMP RJ30                30000B
0627 00766 326 020004 JMP RJ30                30400E
0628 00767 326 040004 JMP RJ30                31000E
0629 00770 327 000004 JMP RJ30                34000E
0630 00771 327 020004 JMP RJ30                34400E
0631 00772 327 040004 JMP RJ30                35000B
0632 00773 327 060004 JMP RJ30                35400B
0633 00774 324 000004 JMP RJ30                20000E    DYNAMIC MAPPING SYSTEM
0634 00775 324 001004 JMP RJ30                20020E    DYNAMIC MAPPING SYSTEM
0635 00776 320 041004 JMP RJ30                EIG        EXTENDED INSTRUCTION GROUP
0636 00777 320 042004 JMP RJ30                EIG+20E    EXTENDED INSTRUCTION GROUP
0637
0638 01000 320 061557 MACTABLU JMP STFL        FAD        FLOATING POINT ADD
0639 01001 320 061547 JMP                FSE        FLOATING POINT SUBTRACT
0640 01002 320 065107 JMP                FMP        FLOATING POINT MULTIPLY
0641 01003 320 067047 JMP                FFDV       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 FJ30          36000B
0645 01007 327 140004 JMP FJ30          37000B
0646 01010 324 040004 JMP RJ30          21000E    FAST FCFTAN
0647 01011 324 060004 JMP RJ30          21400E    FAST FCFTAN
0648 01012 324 100004 JMP FJ30          22000B    HP RESERVED
0649 01013 324 120004 JMP RJ30          22400E    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 FJ30          26000E    HP RESERVED
0653 01017 325 120004 JMP RJ30          26400E    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 PTN                PASS Y    CAE        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                LEY        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       CES
0688 01055 320 056707 JMP                BITS       TBS
0689 01056 320 047407 JMP                CMW        CMW
0690 01057 320 050747 JMP                MVW        MVW

```

Appendix G

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  FJS      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 PTN  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/E
0749 01133 227 174700 READ RTN  INC PNM  P           M := P; P := P+1; FEAD
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; FEAD
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; FEAD
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 ZEPO
0770 01153 227 174700 READ RTN  INC PNM  P           M := P; P := P+1; FEAD
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; FEAD

```

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, FEAD
0789 01167 370 036776 RTN MPCK          CHECK JUMP TARGETS

```

Appendix G

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           A
0865 01266 010 047163      L4 PASS S4   S4          S4 := TERMINATION BYTE
0866 01267 150 011064 LSFBS 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 LSFBS     TEST FOR INTERRUPT PENDING
0875 01300 000 075707      DEC P         P           DECFEMENT 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         S3          RESTORE M AND PEAD
0883 01306 007 110207      INC B         B           BUMP BYTE ADDRESS
0884 01307 370 040147      RTN          S1          A := RIGHT JUSTIFIED BYTE
0885 *
0886 01310 344 000507 SBT   IMM           HIGH L    000B      L := 000377
0887 01311 010 033207      PASS S5      M           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

```

Appendix G

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 BYTES
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 PEAD      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  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
0956 01377 370 101007 SBS RTN      IOR S1  TAB      S1 := WORD WITH BITS SET

```

```

PAGE 0033 RTE MICRO-ASSEMBLER REV.A 760818
0958          CRG          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
0972 01402 353 141147          IMM  CMLO S4  $360  CLEAF LSL'S TO SHIFT INTO E
0973 01403 000 075707          DEC  P  P          SET EXPONENT FOR MAX INTEGER
0974 01404 320 073007          JMP          PACK          BECAUSE PACK BUMPS IT

PAGE 0034 RTE MICRO-ASSEMBLER REV.A 760818
0976          *
0977          *          ON ENTRY-- A,E = FLOATING POINT NUMBER
0978          *          FLAG = 1
0979          *
0980          *          ON EXIT      A = INTEGER      B = CHANGED (USUALLY = A,THOUGH)
0981          *
0982          *          USES A,E,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 ALZ  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 LEAD; LXIT
0999          *
1000 01423 010 040567          FIXOK2          RPT  PASS CNTR S1          CCOUNTER := $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 := LSE'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 HANGING
1009          *
1010 01432 227 106140          READ RTN  INC  A  A          ELSE ROUND UP AND RETURN

```

Appendix G

```

PAGE 0035 RTE MICRO-ASSEMBLER REV.A 760818
1012 * F A D / F S B -- FLOATING POINT ADD / SUBTRACT
1013 * -----
1014 * CN ENTRY--A,B = FIRST CPERAND
1015 * P = POINTER TO ALDFESS OF SECOND CPEFAND
1016 * FLAG = 1 MEANS ADD =0 MEANS SUBTRACT
1017 * ON EXIT-- A,E = (FIRST OPERAND) +(-) (SECOND CPEFAND)
1018 *
1019 * USES REGISTERS S1,S2,S3,S4,S5,S6
1020 *
1021 FSB EQU * ITS THE SAME AS FAD
1022 01433 230 036747 FAD READ
1023 *
1024 01434 300 012447 JSB INDIRECT GO CLEAR INDIFECTS IF NECESSARY
1025 01435 300 071607 JSB UNPACK GO UNPACK THE NUMBERS
1026 01436 010 042747 PASS S2 IS OP2 = C?
1027 01437 320 062042 JMP CNDX ALZ RJS *+2 SKIP IF NOT
1028 01440 342 001147 IMM LOW S4 %200 EXP(D) := -200
1029 *
1030 01441 010 010747 PASS B IS OP1 = C?
1031 01442 320 062202 JMP CNDX ALZ RJS *+2 SKIP IF NOT
1032 01443 342 001207 IMM LOW S5 %200 EXP(C) := -200
1033 *
1034 01444 334 022742 JMP CNDX FLAG DIFF SKIP AHEAD IF DOING AN ADD
1035 01445 017 143047 CMPS S2 S2 -ELSE NEGATE OP2
1036 01446 017 145107 CMPS S3 S3
1037 01447 007 145107 INC S3 S3
1038 01450 321 062742 JMP CNDX COUT RJS DIFR IF NO CARRY OUT, GO PROCEED
1039 01451 007 143047 INC S2 S2 -BUMP MSB'S
1040 01452 327 162742 JMP CNDX AL15 RJS DIFR IF POSITIVE, GO PROCEED
1041 01453 001 142747 DELS S2 -WAS IT 100...0?
1042 01454 320 062742 JMP CNDX ALZ RJS DIFF
1043 *
1044 01455 010 043064 R1 PASS S2 S2 YES, MAKE IT 010...0
1045 01456 007 147147 INC S4 S4 AND ADJUST EXPONENT

```


PAGE 0036 RTE MICRO-ASSEMBLER REV.A 760818

```

1047 *
1048 * FIND DIFFERENCE IN EXPONENTS--SMALLER EXPONENT GETS RIGHT-SHIFTED.
1049 *
1050 01457 010 046507 DIFR PASS L S4 L := EXP(D)
1051 01460 004 151007 SUB S1 S5 S1 := EXP(C) - EXP(D)
1052 01461 320 024102 JMP CNDX ALZ ADD2 IF 0,DO THE ADDITION (NO SHIFTS)
1053 *
1054 01462 327 163302 JMP CNDX AL15 RJS RVFS
1055 *
1056 * WE NEED TO SHIFT THE SECOND NUMBER.
1057 *
1058 01463 017 141007 CMPS S1 S1
1059 01464 007 141007 INC S1 S1 S1 := POSITIVE DIFFERENCE
1060 01465 320 063607 JMP SWAMPCHK GO CHECK IF ONE OF THEM>THE OTHER
1061 *
1062 * SWAP THE NUMEERS--WRONG ONE IS IN B,A
1063 *
1064 01466 010 042207 RVRS PASS B S2 B := S2
1065 01467 010 053047 PASS S2 S6
1066 01470 010 006507 PASS L A
1067 01471 010 044147 PASS A S3 A := S3
1068 01472 012 137107 PASL S3
1069 01473 010 051147 PASS S4 S5 S4 := LAPCER EXPONENT
1070 *
1071 * CHECK FOP ABS(EXP2 - EXP1) > 30 --IF SO, ADDING WILL DO NOTHING
1072 *
1073 01474 343 116507 SWAMPCHK IMM LOW L $347
1074 01475 003 040547 ADD CNTR S1 -- TEST (S1 - 30(B8))
1075 01476 327 172702 JMP CNDX AL15 RJS TOOBIG BUG OUT EARLY
1076 01477 010 036767 ALIGN RPT
1077 01500 030 010224 AFS R1 PASS B B ALIGN THE OPERAND FOR ADDING
1078 01501 326 163742 JMP CNDX CNT8 RJS ALICN IF NOT DONE LOOP
1079 01502 150 042522 ADD2 LWF L1 PASS L S2 SET UP TO ADD THE HI BITS
1080 01503 243 010207 ENV ADD B B ADD THE HIGH BITS
1081 01504 010 044507 PASS L S3 PREPARE FOR ADDING THE LO BITS
1082 01505 003 006147 ADD A A ADD THE LO BITS
1083 01506 321 064442 JMP CNDX COUT RJS NOCARY TEST CARRY OUT FROM LO BITS
1084 01507 345 176507 IMM HIGH L $177
1085 01510 247 110207 ENV INC B B BUMP B IF CARRY OUT OF LO BITS
1086 01511 335 173002 NOCARY JMP CNDX OVFL RJS PACK IF NO OVERFLOW GO PACK IT UP
1087 01512 334 064702 JMP CNDX FLAG RJS OFLOW IF SIGN POSITIVE HANDLE ODD CASE
1088 01513 345 176507 IMM HIGH L $177 SET UP L FOR OVFL TEST
1089 01514 014 110747 XOR B
1090 01515 320 133002 JMP CNDX ONES PACK IF UNIQUE CASE GO PACK IT UP
1091 *
1092 01516 150 010224 OFLOW LWF R1 PASS B B FULL WORD SHIFT;USE FLAG FOR SIGN
1093 01517 150 006164 LWF R1 PASS A A
1094 01520 007 147147 INC S4 S4 BUMP THE EXPONENT
1095 01521 320 073007 JMP PACK GO PACK IT UP

```

Appendix G

PAGE 0037 RTE MICRO-ASSEMBLER REV.A 760818

```

1097          *          F M P -- FLOATING POINT MULTIPLY
1098          *          -----
1099          *          ON ENTRY--A,E = C
1100          *          P = POINTER TO ADDRESS OF D
1101          *
1102          *          ON EXIT--A,E = RESULT
1103          *
1104          *          USES REGISTERS A,E,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  E          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

```

1156          *          F D V -- FLOATING POINT DIVIDE
1157          *          -----
1158          *
1159          *          ON ENTRY-- A,B = C
1160          *          P = POINTER TO ADDRESS OF D
1161          *
1162          *          ON EXIT-- A,B = RESULT
1163          *
1164          *          USES REGISTERS A,B,S1,S2,S3,S4,S5,S6
1165          *
1166 01561 230 036747 FDV      READ
1167 01562 300 012447 JSB      INDIRECT GO CLEAR INDIRECTS IF NECESSARY
1168 01563 300 071607 JSB      UNPACK GO UNPACK THE NUMBERS
1169          *
1170          *          GET SET TO FORM FIRST QUOTIENT OF THE APPROXIMATION (Q0).
1171          *
1172 01564 017 143253          COV CMPS S6 S2 S6 := NOT(MSB(D))
1173 01565 320 135542          JMP CNDX ONES OVERFLOW CHECK FOR DIVIDE BY ZERO!
1174 01566 327 127402          JMP CNDX AL15 *+2
1175 01567 007 153054          SOV INC S2 S6 S2 := ABS(MSB(D)); OVF := SIGN
1176 01570 000 046507          DEC L S4 S4 L := EXP(D) - 1
1177 01571 004 151147          SUB S4 S5 S4 := EXP(C)-EXP(D); CNTR := 1'S
1178 01572 030 010224          ARS R1 PASS B B PRESHIFT TO AVOID OVERFLOW
1179 01573 300 075707          JSB DIVX
1180 01574 010 007207          PASS S5 A S5 := Q0
1181 01575 010 010747          PASS B
1182 01576 321 170002          JMP CNDX AL0 RJS *+2
1183 01577 000 010207          DEC B B FIRST LEFT SHIFT FOR NEXT
1184 01600 006 036147          ZERO A
1185 01601 300 075707          JSB DIVX
1186 01602 010 007247          PASS S6 A
1187 01603 010 044224          FDIV71 R1 PASS B S3 B := LSB(D)/4
1188 01604 010 010224          R1 PASS B B
1189 01605 006 036147          ZERO A
1190 01606 300 075707          JSB DIVX
1191 01607 017 106147          CMPS A A
1192 01610 007 106147          INC A A
1193          *
1194 01611 010 050507          PASS L S5 L := Q0; COUNTER := ALL ONES
1195 01612 300 077047          JSB MPYX
1196          *

```

PAGE 0040 RTE MICRO-ASSEMBLER REV.A 760818

```

1198 01613 010 011047          FDIV81 PASS S2 B S2 := MSB(-Q0*Q2)
1199 01614 006 036207          ZERO B B := 0
1200 01615 010 052747          PASS S6 IF Q1
1201 01616 327 171002          JMP CNDX AL15 RJS *+2 NEGATIVE,
1202 01617 011 136207          ONE B B := ONES
1203 01620 010 042747          PASS S2 IF (-Q0*Q2)
1204 01621 327 171142          JMP CNDX AL15 RJS *+2 NEGATIVE,
1205 01622 000 010207          DEC B B B := B + (ALL ONES)
1206 01623 001 143062          L1 DBLS S2 S2 REORIENT PRODUCT (*4)
1207 01624 010 042507          PASS L S2
1208 01625 003 052147          ADD A S6 ADD TO Q1
1209 01626 321 071402          JMP CNDX COUT RJS *+2 IF THERE WAS A CARRY OUT,
1210 01627 007 110207          INC B E ADD IT TO THE HIGH BITS.
1211          *
1212 01630 070 010222          LGS L1 PASS B B
1213 01631 010 050507          PASS L S5
1214 01632 003 010207          ADD B E ADD Q0 TO MSB
1215 01633 320 073007          JMP PACK GC PACK IT UP

```

Appendix G

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 LSE(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 ADSEXPNT 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 E B < 0? FLAG := SIGN
1307 01737 327 176242 JMP CNDX AL15 RJS READY
1308 01740 017 110207 CMPS B E 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 E 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

```

Appendix G

```

PAGE 0045 RTE MICRO-ASSEMBLER REV.A 760818
1339                                ORG                                2000B
1340
1341                                *
1342                                *           ROM JUMP TABLE
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,PPL,MPY
1476	02201	000	000153	DEF	DIV	201
1477	02202	000	000227	DEF	JTBL1010	202 ASR,LSR,FRF
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,SF5
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

Appendix G

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						
BKGNDC	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			
CPTST	0512	0494	0543							
DECDMS	0340	0335								
DECM	0335	0290								

Appendix G

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							
DLI	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								
FTES	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						
RETNFP 2	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**							

Appendix G

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**																
UPDATM	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 0021 RTE MICRO-ASSEMBLER REV.A 760818

```

0731
0732
0733
0734
0735
0736 00760 320 100004 * MACTABL1 JMP RJ30 2000B 105400B, HP RESERVED
0737 00761 322 100004 * JMP RJ30 12000B 105420B, HP RESERVED 780831 CC
0738 00762 325 140004 * JMP RJ30 27000B 105440B, USER RESERVED
0739 00763 321 100004 * JMP RJ30 6000B 105460B, HP RESERVED
0740 00764 325 160004 * JMP RJ30 27400B 105500B, USER RESERVED
0741 00765 326 000004 * JMP RJ30 30000B 105520B, USER RESERVED
0742 00766 326 020004 * JMP RJ30 30400B 105540B, USER RESERVED
0743 00767 326 040004 * JMP RJ30 31000B 105560B, USER RESERVED
0744 00770 327 000004 * JMP RJ30 34000B 105600B, USER RESERVED
0745 00771 327 020004 * JMP RJ30 34400B 105620B, USER RESERVED
0746 00772 327 040004 * JMP RJ30 35000B 105640B, USER RESERVED
0747 00773 327 060004 * JMP RJ30 35400B 105660B, USER RESERVED
0748 00774 324 000004 * JMP RJ30 20000B DYNAMIC MAPPING SYSTEM
0749 00775 324 001004 * JMP RJ30 20020B DYNAMIC MAPPING SYSTEM
0750 00776 320 041004 * JMP RJ30 EIG EXTENDED INSTRUCTION GROUP
0751 00777 320 042004 * JMP RJ30 EIG+20B EXTENDED INSTRUCTION GROUP
0752
0753
0754
0755
0756
0757

```

PAGE 0023 RTE MICRO-ASSEMBLER REV.A 760818

```

0780
0781
0782
0783 01000 320 060004 * MACTABLO JMP RJ30 ASMD2345 FLOATING POINT ADD
0784 01001 320 060004 * JMP RJ30 ASMD2345 FLOATING POINT SUBTRACT
0785 01002 320 060004 * JMP RJ30 ASMD2345 FLOATING POINT MULTIPLY
0786 01003 320 060004 * JMP RJ30 ASMD2345 FLOATING POINT DIVIDE
0787 01004 343 130507 * IMM LOW L 354B L= COMPL OF 24B
0788 01005 320 061004 * JMP RJ30 XTSD2345 FLOATING POINT TO INTEGER
0789
0790 01006 327 100004 * JMP RJ30 36000B 105140B, USER RESERVED
0791 01007 327 140004 * JMP RJ30 37000B 105160B, USER RESERVED
0792 01010 324 040004 * JMP RJ30 21000B FAST FORTRAN
0793 01011 324 060004 * JMP RJ30 21400B FAST FORTRAN
0794 01012 324 100004 * JMP RJ30 22000B 105240B, EMA
0795 01013 321 000004 * JMP RJ30 4000B 105260B, HP RESERVED
0796 01014 324 140004 * JMP RJ30 23000B 105300B, DS
0797 01015 325 000004 * JMP RJ30 24000B 105320B, SCI. INSTRUCTION SET
0798 01016 322 000004 * JMP RJ30 10000B 105340B, HP RESERVED
0799 01017 322 040004 * JMP RJ30 11000B 105360B, HP RESERVED 780831 CC
0800 *NOLIST

```

Appendix G

```

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 LGOP 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 CN 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 := 1111111100000000
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

```

Appendix G

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 := 1111111110111111
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 RESET MAP STATE
0179 20143 010 022447 PASS MEU MEU START NEXT INST READ - EXIT
0180 20144 227 174700 READ RTN INC PNM P *****
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 RTN-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

```

Appendix G

```

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 := 000000011111111
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 Ll PASS A A FORM BYTE ADDRESS IN A
0247 20217 014 001152 MESP SANL S4 TAB S4 := AAAAAAAAA00000000
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 Ll PASS A A FORM BYTE ADDRESS IN A
0258 20231 014 001147 SANL S4 TAB S4 := AAAAAAAAA00000000
0259
0260 20232 230 010647 MB* READ PASS M B M := DESTINATION ADDRESS
0261 20233 010 010222 Ll PASS B B FORM FYTE ADDRESS IN B
0262 20234 012 000507 AND L TAB L := 00000000BBBBBBBB
0263 20235 010 147147 IOR S4 S4 S4 := AAAAAAAAABBBBBBBB
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 Ll 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 Ll PASS A A RESET A FOR EVEN BYTE ADDRESS
0274 20247 370 010222 RTN Ll 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 Rl PASS A A A := SOURCE WORD ADDRESS
0278 20252 010 010224 Rl PASS B B B := DESTINATION WORD ADDRESS
0279 20253 150 071624 LWF Rl 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 T-BUS := X
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 E 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 MWV 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 INTERRUPT 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 JMP R1 PASS S4 S4 S4 := 0010000001000000
0331 20316 324 015207 JMP MAPMOVE
0332 *****
0333 20317 342 077147 PB* IMM LOW S4 %237 S4 := 111111110011111
0334 20320 324 015107 JMP US*+1 L := 110111111111111
0335 * S4 := 0010000001100000
0336 *****
0337 20321 343 077147 US* IMM LOW S4 %337 S4 := 111111111011111
0338 20322 347 076507 IMM HIGH L %337 L := 110111111111111
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

```

Appendix G

```

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 WOPD INTO MEMORY
0370 20350   210 040036          WRTE MPCK PASS TAE  S1
0371 20351   324 010107          JMP          JS*EXIT
0372          *****
0373 20352   230 036747 JRS      READ
0374 20353   304 017377          JSB IOFF          OPGLT   GET STATUS WORD FROM ^.INSTR+1
0375 20354   150 001222          LWF LI   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 USEP SELECTED
0383 20364   345 004507          IMM          HIGH L   %102  IF SYS, L := 0100001011111111
0384 20365   232 047147          READ          AND  S4   S4     THEN S4 := 0100000111111111
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 CNTF
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
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.PESET 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
MAPMCVE 0340 0327 0331
MB*      0260 0248
MBF      0235 0046
MBI      0236 0045
MRW      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 MICRC CRSS-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.LCOP   0292 0237 0300
XC*      0182 0065
XFER     0155 0152
XFERLOCP 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
```

Appendix G

PAGE 0001 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM MON., 6 AUG.

0001 MICMXE,R

END OF PASS 1: NO ERRORS

PAGE 0002 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM MON., 6 AUG.

```

0001          MICMXE,L
0002          *****
0003          *
0004          *
0005          *   ENHANCED SCIENTIFIC INSTRUCTION SET MICROCODE
0006          *   FOR HP1000 F--SERIES COMPUTERS
0007          *
0008          *   JULY 31, 1979 CC,BG,CRG
0009          *
0010          *   NOTE: ESISREVN SHOULD BE DECREMENTED ON EVERY PROM CHANGE
0011          *           I.E. INCREMENT X REG. RTN (X=COMPL OF ESISREVN- SEE SELFTES
0012          *
0013          *****
0014          *
0015          $NOLIST
0020          $LIST
0021          FETCH      EQU          00000B
0022          HORI       EQU          00006B
0023          ESISREVN  EQU          00373B      1'S COMPLEMENT OF 4=REVN
0024          ORG        24000B
0025 24000 325 012047      JMP      TAN
0026 24001 325 040007      JMP      SQRT
0027 24002 325 023047      JMP      ALOG
0028 24003 325 027647      JMP      ATAN
0029 24004 325 015647      JMP      COS
0030 24005 325 015747      JMP      SIN
0031 24006 325 043447      JMP      EXP
0032 24007 325 027207      JMP      ALOGT
0033 24010 325 047147      JMP      TANH
0034 24011 325 052747      JMP      DPOLY
0035 24012 325 056547      JMP      /CMRT
0036 24013 325 065707      JMP      /ATLG
0037 24014 325 070207      JMP      .FPWR
0038 24015 325 070047      JMP      .TPWR
0039 24016 230 036740      READ RTN
0040 24017 325 035347      JMP      SELFTEST

```

PAGE 0003 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0041 *****
0042 *
0043 * SUBROUTINE FLUN
0044 *
0045 * ENTER: B = LOW PART OF FLOATING POINT NUMBER
0046 * RETURN: A = EXPONENT B = LOW MANTISSA
0047 *
0048 2402 0 340 000516 FLUN IMM CLFL LOW L 000B L = 177400B
0049 2402 1 154 010164 LWF RL SANL A B GET EXPONENT IN A
0050 2402 2 012 0102 07 AND B B PUT MANTISSA IN B
0051 2402 3 374 0402 02 RTN CNDX FLAG RJS RETURN IF EXP POSITIVE
0052 2402 4 342 000507 IMM LOW L 200B L = 177600B
0053 2402 5 370 106147 RTN IOR A A RETURN, EXTEND SIGN BIT
0054 *
0055 *****
0056 *
0057 * SUBROUTINE PWR2
0058 *
0059 * ENTER: FLOATING POINT NUMBER IN BOX
0060 * INTEGER IN S11
0061 * RETURN: (A B) = X**S11, P = P + 1
0062 *
0063 2402 6 306 043242 PWR2 JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0064 2402 7 010 020172 MPP1 PASS A MPPB GET A FROM BOX
0065 2403 0 320 002642 JMP CNDX ALZ PDONE EXIT IF X = 0
0066 2403 1 010 0202 32 MPP1 PASS B MPPB GET B FROM BOX
0067 2403 2 340 000516 IMM CLFL LOW L 000B L = 177400B
0068 2403 3 154 011024 LWF RL SANL S1 B S1 = EXPONENT
0069 2403 4 334 041742 JMP CNDX FLAG RJS PNEXT1 JUMP IF EXP POSITIVE
0070 2403 5 342 000507 IMM LOW L 200B L = 177600B
0071 2403 6 010 141007 PNEXT1 IOR S1 S1 EXTEND EXP SIGN BIT
0072 2403 7 010 040507 PASS L S1 PUT EXPONENT IN L
0073 2404 0 143 064762 LWF L1 ADD S11 SET NEW EXP SIGN BIT
0074 2404 1 143 065022 LWF L1 ADD S1 S11 S1 = NEW EXP
0075 2404 2 340 000507 IMM LOW L 000B L = 177400B
0076 *
0077 * CHECK FOR UNDER/OVERFLOW
0078 *
0079 2404 3 012 040753 COV AND S1
0080 2404 4 320 002402 JMP CNDX ALZ PNEXT2 JUMP IF NO OVERFLOW
0081 2404 5 011 040747 SONL S1
0082 2404 6 320 102402 JMP CNDX ONES PNEXT2 JUMP IF NO UNDERFLOW
0083 2404 7 370 036754 RTN SOV OVER/UNDERFLOW RETURN
0084 *
0085 2405 0 014 041007 PNEXT2 SANL S1 S1 S1 = 000EXP
0086 2405 1 012 010507 AND L B L = MAN000
0087 2405 2 010 1402 07 IOR B S1 B = MANEXP
0088 2405 3 227 174707 READ INC PNM P START READ
0089 2405 4 370 036747 RTN RETURN
0090 *
0091 2405 5 010 0202 32 PDONE MPP1 PASS B MPPB GET B
0092 2405 6 227 174707 READ INC PNM P START READ
0093 2405 7 370 036747 RTN RETURN
0094 *
0095 *****

```

Appendix G

```

0097
0098
0099
0100
0101
0102
0103
0104
0105 24060 306 043242
0106 24061 340 110607
0107 24062 010 036751
0108 24063 010 042432
0109 24064 370 044432
0110
0111
0112
0113
0114
0115
0116
0117
0118
0119
0120
0121
0122
0123
0124 24065 340 100547
0125 24066 323 104102
0126 24067 366 004102
0127 24070 010 036765
0128 24071 366 000742
0129 24072 326 143342
0130 24073 011 136154
0131 24074 355 165047
0132 24075 010 042607
0133 24076 010 052206
0134 24077 000 075707
0135 24100 227 174707
0136 24101 320 000007
0137
0138
0139
0140
0141
0142
0143
0144
0145 24102 336 043342
0146 24103 000 075732
0147 24104 010 050147
0148 24105 010 052207
0149 24106 320 000307
0150
0151

```

```

*****
*
* SUBROUTINE FMPY
*
* STARTS BOX MULTIPLY ON ACCUMULATOR
* AND REGISTERS (S2 S3)
*
FMPY JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
IMM LOW IRCM 044B BOX MPY OPCODE
MPP2 START BOX
MPP1 PASS MPPB S2 SEND OPI = (S2 S3)
RTN MPP1 PASS MPPB S3 RETURN
*
*****
*
* SUBROUTINE WAIT1
*
* WAITS FOR BOX TO COMPLETE EXECUTION
* LOOPS FOR COUNT OF 32, THEN ABORTS
* ON ABORT: A SET TO ALL ONES
* B RESTORED
* GENERATE MP INT
*
WAIT1 IMM LOW CNTR 040B CNTR=32
JMP CNDX HOI INTRT1 CHECK FOR INTERRUPTS
LOOP1 RTN CNDX MPP RETURN WHEN DONE
DCNT DECREMENT CNTR
RTN CNDX MPP RETURN WHEN DONE
JMP CNDX CNT8 RJS LOOP1 ELSE LOOP1 32 TIMES
SOV ONE A SET A = 17777B
IMM CMHI S2 172B SET IRCM = MIA 00
PASS IRCM S2
IOG PASS B S6 RESTORE B, MP INT
DEC P P SET P = ERROR ADDR
READ INC PNM P START READ
JMP FETCH RETURN
*
*****
*
* INTERRUPT ROUTINE
*
* RESTORES A,B,P AND RETURNS
*
INTRT1 JMP CNDX NSNG RJS LOOP1 RTN IF SINGLE STEP
MPP1 DEC P P RESET BOX, SET ADDR
PASS A S5 RESTORE A
PASS B S6 RESTORE B
JMP HORI
*
*****

```


PAGE 0005 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0153 *****
0154 *
0155 * SUBROUTINE FSUB2
0156 *
0157 * STARTS BOX SUB ON ACC AND (S2 S3)
0158 *
0159 24107 306 043242 FSUB2 JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0160 24110 340 060607 IMM LOW IRCM 030B BOX SUB OPCODE
0161 24111 010 036751 MPP2 START BOX
0162 24112 010 042432 MPP1 PASS MPPB S2 SEND OP1 = (S2 S3)
0163 24113 370 044432 RTN MPP1 PASS MPPB S3 RETURN
0164 *
0165 *****
0166 *
0167 * SUBROUTINE FADD
0168 *
0169 * STARTS BOX ADD ON ACC AND (S2 S3)
0170 *
0171 24114 306 043242 FADD JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0172 24115 340 020607 IMM LOW IRCM 010B BOX ADD OPCODE
0173 24116 010 036751 MPP2 START BOX
0174 24117 010 042432 MPP1 PASS MPPB S2 SEND OP1 = (S2 S3)
0175 24120 370 044432 RTN MPP1 PASS MPPB S3 RETURN
0176 *
0177 *****
0178 *
0179 * SUBROUTINE FDIV7
0180 *
0181 * STARTS BOX DIV ON (S7 S8) AND ACC
0182 *
0183 24121 306 043242 FDIV7 JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0184 24122 340 150607 IMM LOW IRCM 064B BOX DIV OPCODE
0185 24123 010 036751 MPP2 START BOX
0186 24124 010 054432 MPP1 PASS MPPB S7 SEND OP1 = (S7 S8)
0187 24125 370 056432 RTN MPP1 PASS MPPB S8 RETURN
0188 *
0189 *****
0190 *
0191 * SUBROUTINE XSQ
0192 *
0193 * SAVES ACC IN (S7 S8) AND STARTS ACC*ACC
0194 *
0195 24126 306 043242 XSQ JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0196 24127 340 100607 IMM LOW IRCM 040B BOX MPY OPCODE
0197 24130 010 021332 MPP1 PASS S7 MPPB SAVE IN (S7 S8)
0198 24131 010 021372 MPP1 PASS S8 MPPB
0199 24132 010 036751 MPP2 START BOX
0200 24133 010 054432 MPP1 PASS MPPB S7 SEND OP1 = (S7 S8)
0201 24134 010 056432 MPP1 PASS MPPB S8
0202 24135 010 054432 MPP1 PASS MPPB S7 SEND OP2 = (S7 S8)
0203 24136 370 056432 RTN MPP1 PASS MPPB S8 RETURN
0204 *
0205 *****

```

Appendix G

PAGE 0006 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

02 07          *****
02 08          *
02 09          *   SUBROUTINE FADA2
02 10          *
02 11          *   STARTS BOX ADD ON (A B) AND (S2 S3)
02 12          *
02 13 24137   306 043242 FADA2   JSB  CNDX MPP  RJS  WAIT1   WAIT FOR BOX
02 14 24140   340 000607          IMM      LOW  IRCM 000B   BOX ADD OPCODE
02 15 24141   010 036751 BAB23          MPP2          START BOX
02 16 24142   010 006432          MPP1 PASS MPPB A   SEND OP1 = (A B)
02 17 24143   010 010432          MPP1 PASS MPPB B
02 18 24144   010 042432          MPP1 PASS MPPB S2  SEND OP2 = (S2 S3)
02 19 24145   370 044432          RTN   MPP1 PASS MPPB S3  RETURN
02 20          *
02 21          *****
02 22          *
02 23          *   SUBROUTINE FDAVC2
02 24          *
02 25          *   STARTS BOX DIV ON ACC AND (S2 S3)
02 26          *
02 27 24146   306 043242 FDVAC2  JSB  CNDX MPP  RJS  WAIT1   WAIT FOR BOX
02 28 24147   340 150607          IMM      LOW  IRCM 064B   BOX DIV OPCODE
02 29 24150   010 036751          MPP2          START BOX
02 30 24151   010 042432          MPP1 PASS MPPB S2  SEND OP1 = (S2 S3)
02 31 24152   370 044432          RTN   MPP1 PASS MPPB S3  RETURN
02 32          *
02 33          *****
02 34          *
02 35          *   SUBROUTINE VMPY
02 36          *
02 37          *   STARTS BOX MPY ON ACC AND (S9 S10)
02 38          *
02 39 24153   306 043242 VMPY     JSB  CNDX MPP  RJS  WAIT1   WAIT FOR BOX
02 40 24154   340 110607          IMM      LOW  IRCM 044B   BOX MPY OPCODE
02 41 24155   010 036751          MPP2          START BOX
02 42 24156   010 060432          MPP1 PASS MPPB S9  SEND OP1 = (S9 S10)
02 43 24157   370 062432          RTN   MPP1 PASS MPPB S10  RETURN
02 44          *
02 45          *****
02 46          *
02 47          *   SUBROUTINE FSUB
02 48          *
02 49          *   STARTS SUB ON (S2 S3) AND ACC
02 50          *
02 51 24160   006 037047 NEGATE          ZERO S2          SET (S2 S3) = 0
02 52 24161   006 037107          ZERO S3
02 53          *
02 54          *
02 55 24162   306 043242 FSUB     JSB  CNDX MPP  RJS  WAIT1   WAIT FOR BOX
02 56 24163   340 050607          IMM      LOW  IRCM 024B   BOX SUB OPCODE
02 57 24164   010 036751          MPP2          START BOX
02 58 24165   010 042432          MPP1 PASS MPPB S2  SEND OP1 = (S2 S3)
02 59 24166   370 044432          RTN   MPP1 PASS MPPB S3
02 60          *
02 61          *****

```

PAGE 0007 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0263 *****
0264 *
0265 * SUBROUTINE REDUCE
0266 *
0267 24167 345 043047 FOPI IMM HIGH S2 121B SET (S2 S3 S4) TO 4/PI
0268 24170 341 170507 IMM LOW L 174B
0269 24171 012 043047 AND S2 S2
0270 24172 347 003107 IMM HIGH S3 301B
0271 24173 342 156507 IMM LOW L 267B
0272 24174 012 045107 AND S3 S3
0273 24175 344 117147 IMM HIGH S4 047B
0274 24176 340 004507 IMM LOW L 002B
0275 24177 012 047147 AND S4 S4
0276 *
0277 *
0278 24200 340 102607 REDUCE IMM LOW IRCM 041B BOX MPY OPCODE
0279 24201 340 000511 IMM MPP2 LOW L 000B SET L = 177400B
0280 24202 010 042432 MPP1 PASS MPPB S2 SEND OP1 = (S2 S3 S4)
0281 24203 010 044432 MPP1 PASS MPPB S3
0282 24204 010 046432 MPP1 PASS MPPB S4
0283 24205 010 006432 MPP1 PASS MPPB A SEND OP2 = (A B)
0284 24206 012 011307 AND S7 B SET S7 = MAN(B)
0285 24207 010 054432 MPP1 PASS MPPB S7
0286 24210 014 011347 SANL S8 B SET S8 = EXPO(B)
0287 24211 010 056432 MPP1 PASS MPPB S8
0288 24212 305 003247 JSB WAIT1 WAIT FOR BOX
0289 24213 010 021332 MPP1 PASS S7 MPPB SAVE IN (S7 S8 S9)
0290 24214 010 021372 MPP1 PASS S8 MPPB
0291 24215 010 021432 MPP1 PASS S9 MPPB
0292 24216 341 022607 IMM LOW IRCM 111B BOX FIX OPCODE
0293 24217 353 172511 IMM MPP2 CMLO L 375B SET L = 000002B
0294 24220 305 003247 JSB WAIT1 WAIT FOR BOX
0295 24221 010 020172 MPP1 PASS A MPPB SAVE INTEGER IN A
0296 24222 150 006164 LWF R1 PASS A A SET FLAG = BIT0
0297 24223 010 006162 L1 PASS A A SET BIT0 = 0
0298 24224 334 051402 JMP CNDX FLAG RJS RNXT
0299 24225 010 006747 PASS A TEST A
0300 24226 327 111402 JNP CNDX ALL5 RNXT JUMP IF A < 0
0301 24227 243 006147 ENV ADD A A SET A = A+2
0302 24230 341 042607 RNXT IMM LOW IRCM 121B BOX FLOAT OPCODE
0303 24231 010 036751 MPP2 START BOX
0304 24232 010 006432 MPP1 PASS MPPB A SEND OPA = A
0305 24233 305 003247 JSB WAIT1 WAIT FOR BOX
0306 24234 340 052607 IMM LOW IRCM 025B BOX SUBTRACT OPCODE
0307 24235 010 036751 MPP2 START BOX
0308 24236 010 054432 MPP1 PASS MPPB S7 SEND OP1 = (S7 S8 S9)
0309 24237 010 056432 MPP1 PASS MPPB S8
0310 24240 370 060432 RTN MPP1 PASS MPPB S9
0311 *
0312 *
0313 *****

```

Appendix G

```

0315 *****
0316 *
0317 * TANGENT ROUTINE
0318 *
0319 *
0320 24241 010 007213 TAN COV PASS S5 A SAVE A,B FOR
0321 24242 010 011247 PASS S6 B INTRT ROUTINE
0322 24243 305 007347 JSB FOPI X = 4X/PI, REDUCE
0323 24244 335 115242 JMP CNDX OVFL TANERR REDUCE ERROR
0324 24245 010 006164 RI PASS A A SET FLAG = BIT1(N)
0325 24246 150 006764 LWF RI PASS A A
0326 24247 305 005307 JSB XSQ GET X, SQUARE
0327 24250 346 177047 IMM HIGH S2 277B SET (S2 S3) = C4 =
0328 24251 343 1416507 IMM LOW L 363B -4.0030956
0329 24252 012 043047 AND S2 S2
0330 24253 345 045107 IMM HIGH S3 122B
0331 24254 340 014507 IMM LOW L 006B
0332 24255 012 045107 AND S3 S3
0333 24256 306 043242 JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0334 24257 010 021432 MPP1 PASS S9 MPPB SAVE IN XSQ (S9 S10)
0335 24250 010 021472 MPP1 PASS S10 MPP3
0336 24261 305 004647 JSB FADD+1 Z = Z + C4
0337 24262 346 141047 IMM HIGH S2 260B SET (S2 S3) = C3 =
0338 24263 340 016507 IMM LOW L 007B -1279.5424
0339 24264 012 043047 AND S2 S2
0340 24265 345 045107 IMM HIGH S3 122B
0341 24266 340 054507 IMM LOW L 026B
0342 24267 012 045107 AND S3 S3
0343 24270 305 006307 JSB FDVAC2 Z = C3/Z
0344 24271 010 061047 PASS S2 S9 SET (S2 S3) = XSQ
0345 24272 010 063107 PASS S3 S10
0346 24273 305 004607 JSB FADD Z = Z + XSQ
0347 24274 345 003047 IMM HIGH S2 101B SET (S2 S3) = C2 =
0348 24275 341 150507 IMM LOW L 164B .0019974806
0349 24276 012 043047 AND S2 S2
0350 24277 354 053107 IMM CMHI S3 025B
0351 24300 353 142507 IMM CMLO L 361B
0352 24301 017 045107 NOR S3 S3
0353 24302 305 003007 JSB FMPY Z = C2 * Z
0354 24303 345 027047 IMM HIGH S2 113B SET (S2 S3) = C1 =
0355 24304 340 164507 IMM LOW L 072B .146926953
0356 24305 012 043047 AND S2 S2
0357 24306 354 005107 IMM CMHI S3 002B
0358 24307 353 172507 IMM CMLO L 375B
0359 24310 017 045107 NOR S3 S3
0360 24311 305 004607 JSB FADD Z = Z + C1
0361 24312 010 055047 PASS S2 S7 SET (S2 S3) = X
0362 24313 010 057107 PASS S3 S8
0363 24314 305 003007 JSB FMPY Z = X * Z
0364 24315 334 055042 JMP CNDX FLAG RJS EXIT1

```

```

0366 24316 355 177047 IMM CMHI S2 177B SET (S2 S3) = -1.0
0367 24317 006 037107 ZERO S3
0368 24320 305 006307 JSB FDVAC2 Z = -1/Z
0369 24321 305 003247 EXIT1 JSB WAIT1 WAIT FOR BOX
0370 24322 227 174707 READ INC PNM P START READ
0371 24323 010 020172 MPP1 PASS A MPPB PUT ANSWER IN (A B)
0372 24324 370 020232 RTN MPP1 PASS B MPPB RETURN
0373 *
0374 *
0375 24325 340 162514 TANERR IMM SOV LOW L 071B SET (A B) TO
0376 24326 344 140147 IMM HIGH A 060B ASCII 090R
0377 24327 012 006147 AND A A
0378 24330 341 044507 IMM LOW L 122B
0379 24331 345 036207 IMM HIGH B 117B
0380 24332 000 075732 ERRET MPP1 DEC P P SET P = ERROR ADDR
0381 24333 227 174707 READ INC PNM P START READ
0382 24334 372 010207 RTN AND B B ERROR RETURN
0383 *
0384 *
0385 *****

```

```

0387 *****
0388 *
0389 *
0390 *      SINE ROUTINE
0391 *
0392 *
0393 24335 353 173007 COS      IMM      CMLO S1 375B      SET J=2
0394 24336 325 016007      JMP      SIN+1
0395 *
0396 *
0397 24337 006 037007 SIN      ZERO S1      SET J=0
0398 24340 010 007213      COV PASS S5      A      SAVE A,B FOR
0399 24341 010 011247      PASS S6      B      INTRT ROUTINE
0400 24342 305 007347      JSB      FOPI      X = 4X/PI, REDUCE
0401 24343 335 122542      JMP CNDX OVFL SINERR REDUCE ERROR
0402 24344 010 040507      PASS L      S1      SET L=J
0403 24345 003 006164      R1 ADD A      A      N = (N+J)/2
0404 24346 150 007024      LWF R1 PASS S1 A      SET FLAG = BIT1(N)
0405 24347 305 005307      JSB      XSQ      GET X, SQUARE
0406 24350 334 060242      JMP CNDX FLAG RJS SINAG SIN OR COS ?
0407 *
0408 24351 343 127107 COSAG IMM      LOW S3 353B      SET (S2 S3) = CC4 =
0409 24352 344 012507      IMM      HIGH L 005B      -.00031957
0410 24353 012 045107      AND S3 S3
0411 24354 346 131047      IMM      HIGH S2 254B
0412 24355 340 164507      IMM      LOW L 072B
0413 24356 012 043047      AND S2 S2
0414 24357 306 043242      JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0415 24360 010 021432      MPP1 PASS S9 MPPB SAVE VSQR IN (S9 S10)
0416 24361 010 021472      MPP1 PASS S10 MPPB
0417 24362 305 003047      JSB      FMPY+1      Z = Z*CC4
0418 24363 345 000507      IMM      HIGH L 100B      SET (S2 S3) = CC3 =
0419 24364 343 133047      IMM      LOW S2 355B      .015851077
0420 24365 012 043047      AND S2 S2
0421 24366 354 036507      IMM      CMHI L 017B
0422 24367 353 157107      IMM      CMLO S3 367B
0423 24370 017 045107      NOR S3 S3
0424 24371 305 004607      JSB      FADD      Z = Z+CC3
0425 24372 305 006547      JSB      VMPY      Z = Z*VSQR
0426 24373 356 142507      IMM      CMHI L 261B      SET (S2 S3) = CC2 =
0427 24374 350 027047      IMM      CMLO S2 013B      -.30842483
0428 24375 017 043047      NOR S2 S2
0429 24376 344 045107      IMM      HIGH S3 022B
0430 24377 305 004607      JSB      FADD      Z = Z+CC2
0431 24400 305 006547      JSB      VMPY      Z = Z*VSQR
0432 24401 356 177047      IMM      CMHI S2 277B      SET (S2 S3) = CC1 =
0433 24402 353 173107      IMM      CMLO S3 375B      1.0
0434 24403 305 004607      JSB      FADD      Z = Z+CC1
0435 24404 325 022347      JMP      CKSGN      TEST SIGN OF ANSWER

```

Appendix G

PAGE 0011 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0437 24405 342 067047 SINAG IMM LOW S2 233B SET (S2 S3) = C4 =
0438 24406 346 150507 IMM HIGH L 264B -.000035950439
0439 24407 012 043047 AND S2 S2
0440 24410 355 002507 IMM CMHI L 101B
0441 24411 353 113107 IMM CMLO S3 345B
0442 24412 017 045107 NOR S3 S3
0443 24413 306 043242 JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0444 24414 010 021432 MPP1 PASS S9 MPPB SAVE VSQR IN (S9 S10)
0445 24415 010 021472 MPP1 PASS S10 MPPB
0446 24416 305 003047 JSB FMPY+1 Z = VSQR*C4
0447 24417 345 042507 IMM HIGH L 121B SET (S2 S3) = C3 =
0448 24420 342 057047 IMM LOW S2 227B .002490001
0449 24421 012 043047 AND S2 S2
0450 24422 356 110507 IMM CMHI L 244B
0451 24423 353 143107 IMM CMLO S3 361B
0452 24424 017 045107 NOR S3 S3
0453 24425 305 004607 JSB FADD Z = Z+C3
0454 24426 305 006547 JSB VMPY Z = Z*VSQR
0455 24427 346 132507 IMM HIGH L 255B SET (S2 S3) = C2 =
0456 24430 341 043047 IMM LOW S2 121B -.0807454325
0457 24431 012 043047 AND S2 S2
0458 24432 354 044507 IMM CMHI L 022B
0459 24433 353 167107 IMM CMLO S3 373B
0460 24434 017 045107 NOR S3 S3
0461 24435 305 004607 JSB FADD Z = Z+C2
0462 24436 305 006547 JSB VMPY Z = Z*VSQR
0463 24437 355 110507 IMM CMHI L 144B SET (S2 S3) = C1 =
0464 24440 352 017047 IMM CMLO S2 207B .78539816
0465 24441 017 043047 NOR S2 S2
0466 24442 354 045107 IMM CMHI S3 022B
0467 24443 305 004607 JSB FADD Z = Z+C1
0468 24444 010 055047 PASS S2 S7 SET (S2 S3) = V
0469 24445 010 057107 PASS S3 S8
0470 24446 305 003007 JSB FMPY Z = Z * V
0471 *
0472 24447 010 040747 CKSGN PASS S1 TEST SIGN
0473 24450 321 155042 JMP CNDX AL0 RJS EXIT1 EXIT IF BIT2 (N) = 1
0474 24451 305 007007 JSB NEGATE Z = -Z
0475 24452 325 015047 JMP EXIT1 EXIT ROUTINE
0476 *
0477 *
0478 24453 344 140514 SINERR IMM SOV HIGH L 060B SET (A B) TO
0479 24454 340 152147 IMM LOW A 065B ASCII 050R
0480 24455 012 006147 AND A A
0481 24456 345 036507 IMM HIGH L 117B
0482 24457 341 044207 IMM LOW B 122B
0483 24460 325 015507 JMP ERRET ERROR RETURN
0484 *
0485 *
0486 *****

```

PAGE 0012 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0488 *****
0489 *
0490 *      NATURAL LOGARITHM ROUTINE
0491 *
0492 24461 010 011253 ALOG      COV PASS S6 B      SAVE A,B FOR
0493 24462 010 007207          PASS S5 A      INTRT ROUTINE
0494 24463 320 026702          JMP CNDX ALZ LOGERR  ERROR IF X = 0
0495 24464 327 126702          JMP CNDX AL15 LOGERR  ERROR IF X < 0
0496 24465 305 001007          JSB          FLUN    UNPACK MANT AND EXP
0497 24466 357 062507          IMM          CMHI L  331B  TEST FOR BITS 14-7
0498 24467 003 050747          ADD          S5      OF X > 264B
0499 24470 327 123602          JMP CNDX AL15 LGNXT  JUMP IF GREATER
0500 24471 000 006147          DEC A        A      DECREMENT EXPO(X)
0501 24472 007 110207          INC B        B      SET EXPO(MAN(X)) = 1
0502 24473 007 110207          INC B        B
0503 24474 341 040607 LGNXT  IMM     LOW  IRCM 120B  BOX FLOAT OPCODE
0504 24475 356 177051          IMM MPP2 CMHI S2 277B  SET (S2 S3) = 1.0
0505 24476 010 006432          MPP1 PASS MPPB A      SEND OP1 = EXPO(X)
0506 24477 010 050147          PASS A       S5      SET (A B) = MAN(X)
0507 24500 353 173107          IMM CMLO S3 375B
0508 24501 306 043242          JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0509 24502 010 021032          MPP1 PASS S1 MPPB  SAVE CHAR IN (S1 S11)
0510 24503 010 021532          MPP1 PASS S11 MPPB
0511 24504 340 040607          IMM     LOW  IRCM 020B  BOX SUB OPCODE
0512 24505 305 006047          JSB          BAB23  Z = MAN(X) - 1.0
0513 24506 305 003247          JSB          WAIT1  WAIT FOR BOX
0514 24507 010 021332          MPP1 PASS S7 MPPB  SAVE IN Y (S7 S8)
0515 24510 010 021372          MPP1 PASS S8 MPPB
0516 24511 305 006007          JSB          FADA2+1 Z = MAN(X) + 1.0
0517 24512 305 005047          JSB          FDIV7  Z = Z/Y
0518 24513 305 003247          JSB          WAIT1  WAIT FOR BOX
0519 24514 010 021432          MPP1 PASS S9 MPPB  SAVE IN W (S9 S10)
0520 24515 010 021472          MPP1 PASS S10 MPPB
0521 24516 340 130607          IMM     LOW  IRCM 054B  BOX Z*Z OPCODE
0522 24517 345 125051          IMM MPP2 HIGH S2 152B  START BOX
0523 24520 340 020507          IMM     LOW  L    010B  SET (S2 S3) = C =
0524 24521 012 043047          AND S2       S2      1.6567626301
0525 24522 345 115107          IMM     HIGH S3 146B
0526 24523 340 004507          IMM     LOW  L    002B
0527 24524 012 045107          AND S3       S3
0528 24525 305 004347          JSB          FSUB2  Z = Z-C
0529 24526 346 126507          IMM     HIGH L   253B  SET (S2 S3) = MB =
0530 24527 342 015047          IMM     LOW  S2 206B  -2.6398577035
0531 24530 012 043047          AND S2       S2
0532 24531 345 023107          IMM     HIGH S3 111B
0533 24532 340 010507          IMM     LOW  L    004B
0534 24533 012 045107          AND S3       S3
0535 24534 305 006307          JSB          FDVAC2  Z = MB/Z
0536 24535 345 044507          IMM     HIGH L   122B  SET (S2 S3) = A =
0537 24536 342 141047          IMM     LOW  S2 260B  1.2920070987
0538 24537 012 043047          AND S2       S2
0539 24540 344 177107          IMM     HIGH S3 077B
0540 24541 340 004507          IMM     LOW  L    002B
0541 24542 012 045107          AND S3       S3

```

Appendix G

PAGE 0013 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0543 24543 305 004607      JSB          FADD      Z = Z+A
0544 24544 305 006547      JSB          VMPY      Z = Z*W
0545 24545 010 041047          PASS S2      S1        SET (S2 S3) = CHAR
0546 24546 010 065107          PASS S3      S11
0547 24547 305 004607      JSB          FADD      Z = Z+CHAR
0548 24550 355 060507      IMM          CMHI L    130B   SET (S2 S3) = LE2 =
0549 24551 352 163047      IMM          CMLO S2   271B   .6931471806
0550 24552 017 043047          NOR S2      S2
0551 24553 357 147116      IMM CLFL CMHI S3   363B
0552 24554 305 003007      JSB          FMPY      Z = Z*LE2
0553 24555 325 015047      JMP          EXIT1    EXIT ROUTINE
0554
0555
0556 24556 344 140514      LOGERR      IMM SOV HIGH L    060B   SET (A B) TO
0557 24557 340 144157      IMM STFL LOW A     062B   ASCII 02UN
0558 24560 012 006147          AND A       A
0559 24561 345 052507      IMM          HIGH L    1253
0560 24562 341 034207      IMM          LOW B     116B
0561 24563 325 015507      JMP          ERRET    ERROR RETURN
0562
0563
0564
0565
0566
0567
0568
0569 24564 305 023047      ALOGT      JSB          ALOG      COMPUTE LN(X)
0570 24565 374 023042      RTN        CNDX FLAG   ERROR RETURN
0571
0572 24566 000 075707          DEC P       P          SET RETURN ADDRESS
0573 24567 355 136507      IMM          CMHI L    157B   SET (S2 S3) = LOG(E) =
0574 24570 350 133047      IMM          CMLO S2   055B   .43429228
0575 24571 017 043047          NOR S2      S2
0576 24572 347 131107      IMM          HIGH S3   354B
0577 24573 305 003047      JSB          FMPY+1
0578 24574 325 015047      JMP          EXIT1    EXIT ROUTINE
0579
0580
0581

```


PAGE 0014 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0583          *****
0584          *
0585          *      ARCTANGENT ROUTINE
0586          *
0587          *
0588 24575 340 000607 ATAN   IMM      LOW  IRCM 000B      BOX ADD OPCODE
0589 24576 150 010764      LWF    R1   PASS  B          SET FLAG = BIT0 (B)
0590 24577 010 011253      COV    PASS S6  B          SAVE A,B FOR
0591 24600 010 007207      PASS  S5  A          INTRT ROUTINE
0592 24601 320 035202      JMP    CNDX ALZ  ZERO1      RETURN ZERO IF X = 0
0593 24602 334 030242      JMP    CNDX FLAG  POS       JUMP IF ABS(X) < .5
0594 24603 327 170242      JMP    CNDX AL15 RJS POS    JUMP IF X > 0
0595 24604 340 040607      IMM    LOW  IRCM 020B      BOX SUBTRACT OPCODE
0596 24605 006 037051      POS    MPP2  ZERO S2       SET S2 = 0
0597 24606 010 042432      MPP1  PASS  MPPB S2       SEND OP1 = 0
0598 24607 010 042432      MPP1  PASS  MPPB S2
0599 24610 010 006432      MPP1  PASS  MPPB A        SEND OP2 = (A B)
0600 24611 010 010432      MPP1  PASS  MPPB B
0601 24612 334 032042      JMP    CNDX FLAG  ATAN3    JUMP IF ABS(X) < .5
0602 24613 345 111007      IMM    HIGH S1  144B      SET (S1 S11) = PI/4
0603 24614 342 016507      IMM    LOW  L   207B
0604 24615 012 041007      AND   S1  S1
0605 24616 354 045507      IMM    CMHI S11 022B
0606 24617 356 177047      IMM    CMHI S2  277B      SET (S2 S3) = 1.0
0607 24620 353 173107      IMM    CMLO S3  375B
0608 24621 350 006507      IMM    CHLO L   003B      SET L = 000374B
0609 24622 012 010747      AND   B
0610 24623 320 031402      JMP    CNDX ALZ  ATAN1    JUMP IF ABS(X) < 2
0611 24624 007 165507      INC   S11 S11           SET (S1 S11) = PI/2
0612 24625 007 165507      INC   S11 S11
0613 24626 305 006307      JSB   FDVAC2           X = 1/X
0614 24627 325 032047      JMP   ATAN3           CONTINUE
0615 24630 306 043242      ATAN1 JSB  CNDX MPP RJS WAIT1  WAIT FOR BOX
0616 24631 010 020172      MPP1  PASS  A   MPPB      SAVE X IN (A B)
0617 24632 010 020232      MPP1  PASS  B   MPPB
0618 24633 305 007147      JSB   FSUB+1           Z = 1 - X
0619 24634 305 003247      JSB   WAIT1           WAIT FOR BOX
0620 24635 010 021332      MPP1  PASS  S7  MPPB      SAVE (1-X) IN (S7 S8)
0621 24636 010 021372      MPP1  PASS  S8  MPPB
0622 24637 305 006007      JSB   FADA2+1         Z = 1 + X
0623 24640 305 005047      JSB   FDIV7           Z = (1-X)/(1+X)
0624 24641 305 003247      ATAN3 JSB  WAIT1       WAIT FOR BOX
0625 24642 010 021332      MPP1  PASS  S7  MPPB      SAVE NEW X IN (S7 S8)
0626 24643 010 021372      MPP1  PASS  S8  MPPB
0627 24644 340 130613      IMM   COV  LOW  IRCM 054B  BOX Z*Z OPCODE
0628 24645 345 001051      IMM   MPP2 HIGH S2  100B  SET (S2 S3) = C4 =
0629 24646 342 136507      IMM    LOW  L   257B      2.0214656
0630 24647 012 043047      IMM    AND  S2  S2
0631 24650 347 063107      IMM    HIGH S3  331B
0632 24651 340 010507      IMM    LOW  L   004B
0633 24652 012 045107      IMM    AND  S3  S3

```

Appendix G

PAGE 0015 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0635 24653 306 043242 JSB CNDX MPP RJS WAIT1 WAIT FOR BOX
0636 24654 010 021432 MPP1 PASS S9 MPPB SAVE XSQ IN (S9 S10)
0637 24655 010 021472 MPP1 PASS S10 MPPB
0638 24656 305 004647 JSB FADD+1 Z = Z + C4
0639 24657 346 151047 IMM HIGH S2 264B SET (S2 S3) = C3 =
0640 24660 340 144507 IMM LOW L 062B -4.7376165
0641 24661 012 043047 AND S2 S2
0642 24662 346 157107 IMM HIGH S3 267B
0643 24663 340 014507 IMM LOW L 006B
0644 24664 012 045107 AND S3 S3
0645 24665 305 006307 JSB FDVAC2 Z = C3/Z
0646 24666 010 061047 PASS S2 S9 SET (S2 S3) = XSQ
0647 24667 010 063107 PASS S3 S10
0648 24670 305 004607 JSB FADD Z = Z + XSQ
0649 24671 345 037047 IMM HIGH S2 117B SET (S2 S3) = C2 =
0650 24672 340 016507 IMM LOW L 007B .154357652
0651 24673 012 043047 AND S2 S2
0652 24674 357 157107 IMM CMHI S3 367B
0653 24675 353 172507 IMM CMLO L 375B
0654 24676 017 045107 NOR S3 S3
0655 24677 305 003007 JSB FMPY Z = C2 * Z
0656 24700 345 057047 IMM HIGH S2 127B SET (S2 S3) = C1 =
0657 24701 340 116507 IMM LOW L 047B 1.3617611
0658 24702 012 043047 AND S2 S2
0659 24703 344 061107 IMM HIGH S3 030B
0660 24704 340 004507 IMM LOW L 002B
0661 24705 012 045107 AND S3 S3
0662 24706 305 004607 JSB FADD Z = C1 + Z
0663 24707 305 005047 JSB FDIV7 Z = X/Z
0664 24710 334 034742 JMP CNDX FLAG EXIT2 EXIT IF ABS(X) < .5
0665 24711 010 041047 PASS S2 S1 SET (S2 S3) = PI/N
0666 24712 010 065107 PASS S3 S11
0667 24713 305 004347 JSB FSUB2 Z = Z - PI/N
0668 24714 010 050747 PASS S5 TEST MAN(X)
0669 24715 327 134742 JMP CNDX AL15 EXIT2 EXIT IF X < 0
0670 24716 305 007007 JSB NEGATE Z = - Z
0671 *
0672 24717 305 003247 EXIT2 JSB WAIT1 WAIT FOR BOX
0673 24720 000 075707 DEC P P SET RETURN ADDR
0674 24721 227 174707 READ INC PNM P START READ
0675 24722 010 020172 MPP1 PASS A MPPB SAVE ANS IN (A B)
0676 24723 370 020232 RTN MPP1 PASS B MPPB RETURN
0677 *
0678 *
0679 24724 000 075707 ZEROL DEC P P SET RETURN ADDR
0680 24725 227 174707 READ INC PNM P START READ
0681 24726 370 036747 RTN RETURN
0682 *
0683 *
0684 *****

```

PAGE 0016 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0686 *****
0687 *
0688 *
0689 * SELF-TEST ROUTINE
0690 *
0691 *
0692 24727 343 076347 SELFTEST IMM LOW DSPI 337B TURN ON "S" DSPI LED
0693 24730 355 167747 IMM CMHI S 173B SET S = 102 001B
0694 24731 007 177747 INC S S
0695 24732 010 076307 PASS DSPL S SEND TO PANEL
0696 24733 356 176147 IMM CMHI A 277B SET (A B) = 4.0
0697 24734 353 162207 IMM CMLO B 371B
0698 24735 305 040007 JSB SQRT CALCULATE SQRT(4)
0699 24736 007 177747 INC S S SET S = 102 002B
0700 24737 335 136402 JMP CNDX OVFL DISPLAY JUMP IF OVERFLOW
0701 24740 356 176507 IMM CMHI L 277B CHECK ANSWER
0702 24741 014 106747 XOR A A
0703 24742 320 076402 JMP CNDX ALZ RJS DISPLAY JUMP IF A WRONG
0704 24743 353 166507 IMM CMLO L 373B
0705 24744 014 110747 XOR B B
0706 24745 320 076402 JMP CNDX ALZ RJS DISPLAY JUMP IF B WRONG
0707 24746 353 000507 IMM CMLO L 300B
0708 24747 010 177747 IOR S S SET S = 102 077B
0709 24750 353 171604 DISPLAY IMM RJ30 CMLO X ESISREVN REVISION #.
0710 24751 370 076307 RTN PASS DSPL S SEND TO PANEL, RETURN
0711 *
0712 *
0713 *****

```

PAGE 0017 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0715                ORG                25000B
0716                *
0717                *****
0718                *
0719                *      SQUARE ROOT ROUTINE
0720                *
0721 25000 010 011253 SQRT          COV  PASS S6  B      SAVE A,B FOR
0722 25001 010 007207                PASS S5  A      INTRT ROUTINE
0723 25002 320 012442                JMP  CNDX ALZ  EXIT3  EXIT IF X = 0
0724 25003 327 103142                JMP  CNDX AL15 SQRERR  ERROR IF X < 0
0725 25004 305 001007                JSB                FLUN  UNPACK EXP AND MAN
0726 25005 150 006762                LWF  L1  PASS  A      ARITHMETIC SHIFT
0727 25006 150 007524                LWF  R1  PASS S11  A      A-REG RIGHT
0728 25007 321 101202                JMP  CNDX AL0  ODD   JUMP IF EXP ODD
0729                *
0730 25010 010 050147 EVEN          PASS A    S5      RESTORE A
0731 25011 345 027047                IMM  HIGH S2  113B  SET (S2 S3) = A2 =
0732 25012 342 024507                IMM  LOW  L   212B  .5901621
0733 25013 012 043047                AND  S2  S2
0734 25014 356 041107                IMM  CMHI S3  220B
0735 25015 340 100607                IMM  LOW  IRCM 040B  BOX MPY OPCODE
0736 25016 305 006047                JSB                BAB23  Z = F*A2
0737 25017 345 125047                IMM  HIGH S2  152B  SET (S2 S3) = B2 =
0738 25020 343 050507                IMM  LOW  L   324B  .4173076
0739 25021 012 043047                AND  S2  S2
0740 25022 346 131107                IMM  HIGH S3  254B
0741 25023 325 042047                JMP  BOTH
0742                *
0743 25024 010 050147 ODD          PASS A    S5      RESTORE A
0744 25025 345 125047                IMM  HIGH S2  152B  SET (S2 S3) = A1 =
0745 25026 343 050507                IMM  LOW  L   324B  .8346152
0746 25027 012 043047                AND  S2  S2
0747 25030 355 045107                IMM  CMHI S3  122B
0748 25031 340 100607                IMM  LOW  IRCM 040B  BOX MPY OPCODE
0749 25032 305 006047                JSB                BAB23  Z = F*A1
0750 25033 345 027047                IMM  HIGH S2  113B  SET (S2 S3) = B1 =
0751 25034 342 024507                IMM  LOW  L   212B  .5901621
0752 25035 012 043047                AND  S2  S2
0753 25036 356 041107                IMM  CMHI S3  220B
0754 25037 007 110207                INC  B      3      SET F = 2 * F
0755 25040 007 110207                INC  B      3
0756                *
0757 25041 305 004607 BOTH          JSB                FADD  Z = Z+B
0758 25042 010 007307                PASS S7  A      SET (S7 S8) = F =
0759 25043 010 011347                PASS S8  B      MAN(X)
0760 25044 305 003247                JSB                WAIT1  WAIT FOR BOX
0761 25045 010 021072                MPP1 PASS S2  MPPB  SAVE IN P (S2 S3)
0762 25046 010 021132                MPP1 PASS S3  MPPB
0763 25047 305 005107                JSB                FDIV7+1  Z = F/P
0764 25050 353 172507                IMM  CMLO L   375B  SET L = 2
0765 25051 003 057347                ADD  S8  S8      SET F = 4 * F
0766 25052 003 057347                ADD  S8  S8

```

PAGE 0018 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0768 25053 305 004607                JSB                FADD  Z = Z+P
0769 25054 305 003247                JSB                WAIT1  WAIT FOR BOX
0770 25055 010 021072                MPP1 PASS S2  MPPB  SAVE IN P (S2 S3)
0771 25056 010 021132                MPP1 PASS S3  MPPB
0772 25057 305 005107                JSB                FDIV7+1  Z = F/P
0773 25060 305 004607                JSB                FADD  Z = Z+P
0774 25061 004 165507                SUB  S11  S11  DEC EXPONENT BY 2
0775 25062 325 001307                JMP                PWR2  Z = Z**S11, RETURN
0776                *
0777                *
0778 25063 344 140514 SQRERR  IMM  SOV  HIGH L   060B  SET (A B) TO
0779 25064 340 146147                IMM  LOW  A   063B  ASCII 03UN
0780 25065 012 006147                AND  A    A
0781 25066 345 052507                IMM  HIGH L   125B
0782 25067 341 034207                IMM  LOW  B   116B
0783 25070 325 015507                JMP                ERRET  ERROR RETURN
0784                *
0785                *
0786                *****

```

Appendix G

```

0788 *****
0789 *
0790 *           EXPONENTIATION ROUTINE
0791 *
0792 25071 010 007213 EXP          COV  PASS S5  A          SAVE A,B
0793 25072 010 011247          PASS S6  B          FOR INTRT ROUTINE
0794 25073 345 070507          IMM   HIGH L  134B        SET (S2 S3 S4) =
0795 25074 341 053047          IMM   LOW  S2  125B        2/LE2
0796 25075 012 043047          AND  S2  S2
0797 25076 344 073107          IMM   HIGH S3  035B
0798 25077 342 050507          IMM   LOW  L  224B
0799 25100 012 045107          AND  S3  S3
0800 25101 346 135147          IMM   HIGH S4  256B
0801 25102 340 010507          IMM   LOW  L  004B
0802 25103 012 047147          AND  S4  S4
0803 25104 305 010007          JSB          REDUCE
0804 25105 150 006762          LWF  L1  PASS  A          ARITH RT SHIFT
0805 25106 150 006164          LWF  R1  PASS  A          ARITH RT SHIFT
0806 25107 007 107507          INC  S11 A          INC, SAVE IN S11
0807 *
0808 *           PERFORM BOUNDS CHECKS
0809 *
0810 25110 010 050747          PASS  S5          TEST X
0811 25111 327 144702          JMP  CNDX AL15 RJS POSCHK  JUMP IF X > 0
0812 *
0813 25112 351 176507          NEGCHK IMM   CMLO L  177B        SET L = 128
0814 25113 003 064747          ADD          S11          INT < -128 ?
0815 25114 327 106502          JMP  CNDX AL15          ZERO2   YES, ANS = 0
0816 25115 325 045107          JMP          EXPNXT       NO, CONTINUE
0817 *
0818 25116 335 106642          POSCHK JMP  CNDX OVFL          EXPERR  REDUCE ERROR
0819 25117 342 000507          IMM   LOW  L  200B        SET L = -128
0820 25120 003 006747          ADD          A          INT > 128?
0821 25121 327 146642          JMP  CNDX AL15 RJS          EXPERR  YES, ERROR
0822 *
0823 25122 305 005307          EXPNXT JSB          XSQ          GET X, SQUARE
0824 25123 356 023047          IMM   CMHI S2  211B        SET (S2 S3) = C2 =
0825 25124 355 143107          IMM   CMHI S3  161B        .05761803
0826 25125 353 162507          IMM   CMLO L  371B
0827 25126 017 045107          IMM   NOR  S3  S3
0828 25127 305 003007          JSB          FMPY          Z = C2 * Z
0829 25130 010 055047          PASS S2  S7          SET (S2 S3) = F
0830 25131 010 057107          PASS S3  S8
0831 25132 305 004347          JSB          FSUB2        Z = Z - F
0832 25133 345 071047          IMM   HIGH S2  134B        SET (S2 S3) = C1 =
0833 25134 341 052507          IMM   LOW  L  125B        5.7708162
0834 25135 012 043047          AND  S2  S2
0835 25136 345 007107          IMM   HIGH S3  103B
0836 25137 340 014507          IMM   LOW  L  006B
0837 25140 012 045107          AND  S3  S3
0838 25141 305 004607          JSB          FADD          Z = C1 + Z

```

```

0840 25142 305 005047          JSB          FDIV7        Z = F/Z
0841 25143 356 177047          IMM   CMHI S2  277B        SET (S2 S3) = .5
0842 25144 006 037107          ZERO S3
0843 25145 305 004607          JSB          FADD          Z = Z + .5
0844 25146 305 001307          JSB          PWR2         Z = Z**S11
0845 25147 375 141302          RTN  CNDX OVFL RJS          RETURN IF PWR2 OK
0846 25150 010 050747          PASS  S5          TEST X
0847 25151 327 146642          JMP  CNDX AL15 RJS          EXPERR  ERROR IF X > 0
0848 *
0849 *
0850 25152 227 174732          ZERO2 READ MPP1 INC  PNM  P          START READ
0851 25153 006 036153          COV  ZERO A          SET ANS = 0
0852 25154 366 036207          RTN  ZERO B          RETURN
0853 *
0854 *
0855 25155 344 140514          EXPERR IMM  SOV  HIGH L  060B        SET (A B) TO
0856 25156 340 156147          IMM   LOW  A  067B        ASCII 070F
0857 25157 012 006147          AND  A  A
0858 25160 345 036507          IMM   HIGH L  117B
0859 25161 341 014207          IMM   LOW  B  106B
0860 25162 325 015507          JMP          ERRET        ERROR RETURN
0861 *
0862 *
0863 *****

```

```

0865 *****
0866 *
0867 *   TANH ROUTINE
0868 *
0869 25163 010 007213 TANH      COV  PASS S5  A      SAVE A,B FOR
0870 25164 010 011247          PASS S6  B      INTRT ROUTINE
0871 *
0872 *   PERFORM BOUNDS CHECKS
0873 *
0874 25165 321 110342          JMP  CNDX AL0  TANH1    JU4P IF ABS(X) < .5
0875 25166 350 016507          IMM  CMLO L   007B    SET L = 000370B
0876 25167 012 010747          AND  B       B
0877 25170 320 052202          JMP  CNDX ALZ  RJS  TANH2    JUMP IF ABS(X) > 8
0878 *
0879 25171 007 110207          INC  B       B      SET X = 2*X
0880 25172 007 110207          INC  B       B
0881 25173 305 043547          JSB                    EXP+2  GET EXP(2*X)
0882 25174 000 075707          DEC  P      P      SET RETURN ADDRESS
0883 25175 356 177047          IMM  CMHI S2 277B    SET (S2 S3) = 1.0
0884 25176 353 173107          IMM  CMLO S3 375B
0885 25177 340 040607          IMM  LOW  IRCH 020E  BOX SUB OPCODE
0886 25200 305 006047          JSB                    BAB23  Z = EXP(2*X) - 1
0887 25201 305 003247          JSB                    WAIT1   WAIT FOR BOX
0888 25202 010 021332          MPP1 PASS S7 MPPB    SAVE IN (S7 S8)
0889 25203 010 021372          MPP1 PASS S3 MPPB
0890 25204 305 006007          JSB                    FADA2+1 Z = EXP(2*X) + 1
0891 25205 305 005047          JSB                    FDIV7   Z = TANH(X)
0892 25206 325 034747          JMP                    EXIT2   EXIT ROUTINE
0893 *
0894 25207 340 100607          TANH1 IMM  LOW  IRCH 040B  BOX MPY OPCODE
0895 25210 010 007047          PASS S2  A      SET (S2 S3) = X
0896 25211 010 011107          PASS S3  B
0897 25212 305 006047          JSB                    BAB23  Z = X * X
0898 25213 345 041047          IMM  HIGH S2 120B    SET (S2 S3) = C3 =
0899 25214 340 112507          IMM  LOW  L   045B    2.5045337
0900 25215 012 043047          AND  S2     S2
0901 25216 344 111107          IMM  HIGH S3 044B
0902 25217 340 010507          IMM  LOW  L   004B
0903 25220 012 045107          AND  S3     S3
0904 25221 305 004607          JSB                    FADD     Z = Z + C3
0905 25222 345 005307          IMM  HIGH S7 102B    SET (S7 S8) = C2 =
0906 25223 343 116507          IMM  LOW  L   347B    2.0907609
0907 25224 012 055307          AND  S7     S7
0908 25225 346 007347          IMM  HIGH S8 203B
0909 25226 340 010507          IMM  LOW  L   004B
0910 25227 012 057347          AND  S8     S8
0911 25230 305 005047          JSB                    FDIV7   Z = C2 / Z

```

```

0913 25231 345 051047          IMM  HIGH S2 124B    SET (S2 S3) = C1 =
0914 25232 342 054507          IMM  LOW  L   226B
0915 25233 012 043047          AND  S2     S2
0916 25234 355 035107          IMM  CMHI S3 116B
0917 25235 353 172507          IMM  CMLO L   375B
0918 25236 017 045107          NOR  S3     S3
0919 25237 305 004607          JSB                    FADD     Z = C1 + Z
0920 25240 010 007047          PASS S2  A      SET (S2 S3) = X
0921 25241 010 011107          PASS S3  B
0922 25242 305 003007          JSB                    FMPY    Z = Z * X
0923 25243 325 034747          JMP                    EXIT2   EXIT ROUTINE
0924 *
0925 *
0926 25244 000 075707          TANH2 DEC  P      P      SET RETURN ADDRESS
0927 25245 010 006747          PASS  A      TEST X
0928 25246 327 112542          JMP  CNDX AL15 NEG     JUMP IF X < 0
0929 25247 356 176147          IMM  CMHI A   277B    SET (A B) = 1.0
0930 25250 353 172207          IMM  CMLO B   375B
0931 25251 227 174707          EXIT3 READ  INC  PNM  P      START READ
0932 25252 370 036747          RTN                    RETURN
0933 *
0934 *
0935 25253 355 176147          NEG  IMM  CMHI A   177B    SET (A B) = -1.0
0936 25254 006 036207          ZERO B
0937 25255 227 174707          READ  INC  PNM  P      START READ
0938 25256 370 036747          RTN                    RETURN
0939 *
0940 *
0941 *****

```

Appendix G

PAGE 0023 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0943      *      DPOLY EVALUATES A RATIONAL FORM USING
0944      *      TRIPLE-PRECISION OPERATIONS.
0945      *
0946      *      CALLING SEQUENCE:
0947      *
0948      *      OPCODE
0949      *      DEF *+6 OR OCT FLAGS+100000B
0950      *      DEF <RESULT>
0951      *      DEF <ARG>
0952      *      DEF <COEFS>
0953      *      DEF <M> ORDER OF NUMERATOR
0954      *      DEF <N> ORDER OF DENOMINATOR
0955      *
0956      *      THE CONSTANTS SHOULD BE ARRANGED FROM HIGHEST TO
0957      *      LOWEST ORDER, NUMERATOR THEN DENOMINATOR. THE
0958      *      FIRST (I.E. HIGHEST-ORDER) DENOMINATOR COEFFICIENT
0959      *      IS ASSUMED TO BE 1.0 AND MUST NOT BE IN THE LIST.
0960      *
0961      *      THERE ARE M+1 NUMERATOR COEFFICIENTS AND N
0962      *      DENOMINATOR COEFFICIENTS. IF N=0, TRNL IS A
0963      *      POLYNOMIAL EVALUATOR. M MUST BE GREATER THAN ZERO.
0964      *
0965      *      THE FLAGS ARE INTERPRETED AS FOLLOWS:
0966      *
0967      *      BIT 0: IF 0, MULTIPLY NUM BY X.
0968      *      BIT 14: IF 1, SUBTRACT NUM FROM DENOM. (N>0)
0969      *      BIT 15: IF 1, EVALUATE IN X**2 & ENABLE FLAGS.
0970      *
0971      *      *****
0972      *
0973      *      ENTRY. SET A = FLAGS, INITIALIZE.
0974      *
0975 25257 010 000157 DPOLY STFL PASS A TAB GET FLAG WORD.
0976 25260 327 113102 JMP CNDX AL15 TRNL1 IF ENABLED.
0977 25261 344 000156 IMM CLFL HIGH A 0 ELSE SET DEFAULT.
0978 25262 000 033613 TRNL1 COV DEC X M X=ADDR INST
0979 25263 007 175547 INC SP P SP=INST+3
0980 25264 353 166507 IMM CML0 L 373B SET L = 4
0981 25265 003 067647 ADD Y SP SET Y = INST+7
0982 25266 305 077007 JSB RDPARAM READ ARG
0983 25267 305 073447 JSB RD14
0984 25270 010 001147 PASS S4 TAB
0985 25271 334 054242 JMP CNDX FLAG RJS TRNL1A IF NO X**2
0986 25272 305 074347 JSB BOXIN14C SQUARE X
0987 25273 305 074547 JSB BOXIN14A
0988 25274 010 041347 PASS S8 S1 COPY X
0989 25275 010 043407 PASS S9 S2
0990 25276 010 045447 PASS S10 S3
0991 25277 010 047507 PASS S11 S4
0992 25300 305 077007 JSB RDPARAM READ P[M]
0993 25301 305 074007 JSB RD57
0994 25302 305 075707 JSB BOXOUT14 SAVE XSQ
0995 25303 340 114607 IMM LOW IRCM 046B FOR XSQ*P[M]
0996 25304 325 054407 JMP TRNL1B

```

PAGE 0024 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

0998          *          COMPUTE NUMERATOR.
0999          *
1000 25305 305 074347 TRNLI1A JSB          BOXIN14C SEND X TO BOX
1001 25306 305 077007          JSB          RDPARAM  READ P[M]
1002 25307 305 074007          JSB          RD57
1003 25310 305 075047 TRNLI1B JSB          BOXIN57A P[M]
1004 25311 305 077007          JSB          RDPARAM  READ M
1005 25312 010 001747                   PASS S   TAB          S=M
1006 25313 305 056147          JSB          LOOPADD  FINISH NUMERATOR
1007 25314 227 174707 READ      INC   PNM   P          READ Q[N-1]
1008 25315 305 074007          JSB          RD57
1009          *
1010          *          CONDITIONALLY MULTIPLY BY X.
1011          *
1012 25316 340 114647          IMM          LOW M   046B   FOR X*NUMERATOR
1013 25317 010 006747                   PASS      A          TEST BIT 0 FLAG.
1014 25320 301 175342          JSB  CNDX AL0 RJS  BOXIN811 IF A<0>=0, MPY
1015 25321 305 077007          JSB          RDPARAM  READ N
1016 25322 010 001747                   PASS S   TAB          S=N.
1017 25323 320 015702          JMP  CNDX ALZ          NZERO   IF N=0.
1018 25324 305 076147          JSB          BXOUT811 SAVE NUMERATOR.
1019          *
1020          *          EVALUATE DENOMINATOR AND DIVIDE.
1021          *
1022 25325 340 004647          IMM          LOW M   002B   FOR X+Q[N-1]
1023 25326 305 074407          JSB          BOXIN14   X
1024 25327 305 075047          JSB          BOXIN57A DO X+Q[N-1]
1025 25330 305 056407          JSB          LOOP3     FINISH DENOMINATOR
1026 25331 340 064647          IMM          LOW M   032B   FOR DENOM-NUM
1027 25332 001 106747                   DBLS      A          TEST BIT 14 FLAG.
1028 25333 307 135342          JSB  CNDX AL15  BOXIN811 IF A<14>, DO IT.
1029 25334 340 154647          IMM          LOW M   066B   FOR NUM/DENOM
1030 25335 305 075347          JSB          BOXIN811 DIVIDE
1031          *
1032          *          STORE & EXIT.
1033          *
1034 25336 007 171547 NZERO          INC   SP   X          SET SP=INST+2
1035 25337 007 167556          CLFL INC  SP   SP
1036 25340 325 067047          JMP          EXITB   WRITE & EXIT
1037          *
1038          *          MAIN LOOP: ACC _ CONST + X * ACC
1039          *
1040 25341 340 114647 LOOP          IMM          LOW M   046B   FOR X*ACC
1041 25342 305 074407          JSB          BOXIN14   DO IT
1042 25343 227 174707 LOOPADD READ      INC   PNM   P          READ NEXT COEF
1043 25344 305 074007          JSB          RD57
1044 25345 323 156342          JMP  CNDX HOI RJS  LOOP2   INT CHECK
1045 25346 336 037342          JMP  CNDX NSNG          INT
1046 25347 305 074747 LOOP2          JSB          BOXIN57  CONST+X*ACC
1047 25350 000 077747 LOOP3                   DEC   S     S          LOOP COUNTER
1048 25351 320 056042          JMP  CNDX ALZ RJS  LOOP
1049 25352 370 036747          RTN

```

Appendix G

PAGE 0025 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

1051      *           /CHRT MULTIPLIES A 4-WORD NUMBER
1052      *           BY A CONSTANT (4/PI OR 2/LN(2)) AND
1053      *           SUBTRACTS THE NEAREST EVEN INTEGER FROM THE
1054      *           RESULT. IF CANCELLATION OCCURS WHICH WOULD
1055      *           AFFECT THE CALLER'S RESULT, ADDITIONAL
1056      *           PRECISION IS USED.
1057      *
1058      *           CALLING SEQUENCE:
1059      *
1060      *           <A = FLAGS>
1061      *           OPCODE
1062      *           DEF <RESULT>
1063      *           DEF <CONST>
1064      *           DEF <ARG>
1065      *           <ERROR RETURN>
1066      *           <NORMAL RETURN> B=N LOWER.
1067      *
1068      *           THE CONSTANTS ARE THE 4-WORD VERSION AND THE
1069      *           THE 4-WORD VALUE WHICH IS THE CONSTANT MINUS
1070      *           THE FIRST 28 BITS OF THE CONSTANT.
1071      *
1072      *           THE FLAGS ARE:  EXP: -2  TANH: -1  TAN:  0
1073      *                           SIN: 4,8  COS: 2,6
1074      *
1075      *           THE ERROR RETURN IS TAKEN IF N DOES NOT FIT
1076      *           IN TWO WORDS.
1077      *
1078      *           METHOD: TWO METHODS ARE USED, DEPENDING ON
1079      *           THE AMOUNT OF CANCELLATION. IF THE ARGUMENT
1080      *           IS IN THE RANGE [-16,16) OR TANH IS INDICATED,
1081      *           IT IS MULTIPLIED BY C AND THE NEAREST EVEN
1082      *           INTEGER (N) IS FOUND AND SUBTRACTED. IF THE
1083      *           CANCELLATION DOES NOT EXCEED ABOUT 4 BITS OR
1084      *           THE RESULT WILL BE FOR COS, TANH OR EXP, THE
1085      *           RESULT IS RETURNED.
1086      *           IF TOO MUCH CANCELLATION OCCURS OR THE
1087      *           ARGUMENT IS OUTSIDE [-8,8), THE ARGUMENT
1088      *           AND THE CONSTANT ARE BROKEN INTO TWO PARTS
1089      *           AND THE PARTIAL PRODUCTS ARE TAKEN. SINCE .
1090      *           THE FIRST PARTIAL PRODUCT IS EXACT, N MAY BE
1091      *           SUBTRACTED SAFELY AND THE SIGNIFICANT BITS IN
1092      *           THE OTHER PRODUCTS ARE NOT LOST IN THE SUM.

```


PAGE 0026 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

1094          *      READ X AND C, DETERMINE IF EASY OR HARD.
1095          *
1096 25353 000 033616 /CMRT      CLFL DEC X      M      X=ADDR INST
1097 25354 010 075547          PASS SP      P      SP=INST+2
1098 25355 305 077007          JSB          RDPARAM  START READ C
1099 25356 305 074007          JSB          RD57     FINISH READ C
1100 25357 007 167647          INC Y      SP      Y=ERROR RETURN
1101 25360 010 033747          PASS S      M      S=ADDR C[4]
1102 25361 305 077007          JSB          RDPARAM  READ X
1103 25362 305 073447          JSB          RD14
1104 25363 010 001153          COV PASS S4  TA3
1105 25364 321 117602          JMP CNDX AL0  CMRT3  IN [-.5,+.5)
1106 25365 010 006147          PASS A      A      TANH ?
1107 25366 320 117602          JMP CNDX ONES CMRT3  YES, EASY.
1108 25367 340 000507          IMM      LOW L      0      L=8-BIT MASK.
1109 25370 014 047547          SANL SP     S4      EXPONENT*2
1110 25371 343 160507          IMM      LOW L      370B   -8
1111 25372 003 067547          ADD SP      SP      EXPONENT*2-8
1112 25373 327 161542          JMP CNDX AL15 RJS CMRT6  OUTSIDE [-8,+8)
1113          *
1114          *      EASY WAY. DO X*C-N.
1115          *
1116 25374 305 074347          CMRT3  JSB          BOXIN14C X IN X*C
1117 25375 010 077707          PASS P      S      P=ADDR C[4]
1118 25376 305 075107          JSB          BOXIN57B  C
1119 25377 305 076147          JSB          3XOUT811  SAVE X*C
1120 25400 341 024607          IMM      LOW IRCM 112B  FIX
1121 25401 343 174511          IMM      MPP2 LOW L  376B  L=1-BIT MASK
1122 25402 305 076447          JSB          WAIT+1
1123 25403 010 020232          MPP1 PASS B  MPP3  B=N LOWER
1124 25404 327 120302          JMP CNDX AL15  CMRT4  NEAREST EVEN
1125 25405 267 110207          ENVE      INC B      B
1126 25406 341 044607          CMRT4  IMM      LOW IRCM 122B  FLOAT
1127 25407 012 010211          MPP2 AND B      B      N
1128 25410 010 010432          MPP1 PASS MPP3 B
1129 25411 335 127542          JMP CNDX OVFL  EXITD  IF OFL (TANH ONLY)
1130 25412 340 054647          IMM      LOW M      026B  FOR X*C-N
1131 25413 305 075347          JSB          BOXIN811  DO IT.

```

Appendix G

```

1133          *          DETERMINE IF CANCELLATION MATTERS.
1134          *
1135 25414 010 006147          PASS A      A      CHECK FLAGS
1136 25415 327 125602          JMP  CNDX AL15  CMRT8  EXP, TANH: NO.
1137 25416 320 021142          JMP  CNDX ALZ  CMRT5  TAN: YES.
1138 25417 010 010507          PASS L      B      N
1139 25420 003 006164          R1  ADD  A      A      (N+(S:4,C:2))/2
1140 25421 010 006147          PASS A      A      SIN ?
1141 25422 321 125602          JMP  CNDX AL0  CMRT8  NO.
1142          *
1143          *          SEE WHETHER EXCESSIVE CANCELLATION.
1144          *
1145 25423 010 047147          CMRT5  PASS S4   S4      X IN [-.5,+.5) ?
1146 25424 321 125602          JMP  CNDX AL0  CMRT8  YES, DONE.
1147 25425 340 000517          IMM  STFL LOW  L      0      L=8-BIT MASK
1148 25426 305 076147          JS3          BXOUT811
1149 25427 321 165602          JMP  CNDX AL0  RJS  CMRT8  OUT [-.5,+.5)
1150 25430 007 064507          OPI  L      S11  EXPONENT*2+2
1151 25431 004 167547          SUB  SP      2*(EXP DIF) - 10
1152 25432 327 125602          JMP  CNDX AL15  CMRT8  LOST < 5 BITS
1153          *
1154          *          HARD WAY. DO (XU*CU-N) + (X-XU)*C * XU*CL .
1155          *
1156 25433 340 104613          CMRT6  IMM  COV  LOW  IRCM 042B  FOR XU*CU
1157 25434 343 160511          IMM  MPP2 LOW  L      370B  MASK FOR XU, CU
1158 25435 010 040432          MPP1 PASS MPPB S1      XU
1159 25436 012 043356          CLFL AND  S8  S2
1160 25437 012 053507          AND  S11 S6
1161 25440 010 056432          MPP1 PASS MPPB S8
1162 25441 006 037407          ZERO S9
1163 25442 010 060432          MPP1 PASS MPPB S9
1164 25443 340 000507          IMM  LOW  L      0
1165 25444 014 047347          SANL S8  S4
1166 25445 010 056432          MPP1 PASS MPPB S8
1167 25446 010 050432          MPP1 PASS MPPB S5      CU
1168 25447 227 176707          READ  INC  PNM  S      READ C[4]
1169 25450 010 064432          MPP1 PASS MPPB S11
1170 25451 010 050432          MPP1 PASS MPPB S9
1171 25452 014 001347          SANL S8  TAB
1172 25453 010 056432          MPP1 PASS MPPB S8
1173 25454 343 060507          IMM  LOW  L      330B  L=-40
1174 25455 010 006147          PASS A      A
1175 25456 327 163002          JMP  CNDX AL15  RJS  *+2  IF SIN/COS/TAN, -40.
1176 25457 343 160507          IMM  LOW  L      370B  IF EXP/TANH, -8.
1177 25460 003 067547          ADD  SP      SP      CHECK EXPONENT.
1178 25461 327 167542          JMP  CNDX AL15  RJS  EXITD IF OUT OF RANGE.
1179 25462 305 076147          JSB          BXOUT811  SAVE XU*CU
1180 25463 341 034607          IMM  LOW  IRCM 116B  FIX
1181 25464 343 174511          IMM  MPP2 LOW  L      376B  1-BIT MASK
1182 25465 305 076447          JSB          WAIT+1
1183 25466 010 021572          MPP1 PASS SP  MPPB  N UPPER
1184 25467 010 020232          MPP1 PASS B  MPPB  N LOWER
1185 25470 010 067547          PASS SP      SP
1186 25471 327 123642          JMP  CNDX AL15  CMRT7  NEAREST EVEN
1187 25472 007 110207          INC  B      B
1188 25473 321 063642          JMP  CNDX COUT RJS  CMRT7  PROPOGATE CARRY
1189 25474 267 167547          ENVE  INC  SP      SP

```

1190	25475	341	054607	CMRT7	IMM	LOW	IRCM	126B	FLOAT
1191	25476	012	010211		MPP2	AND	B	B	
1192	25477	010	066432		MPP1	PASS	MPPB	SP	
1193	25500	010	010432		MPP1	PASS	MPPB	B	
1194	25501	335	127542		JMP	CNDX	OVFL	EXITD	IF OFL
1195	25502	340	054647		IMM	LOW	M	026B	FOR XU*CU-N
1196	25503	305	075347		JSB			BOXIN811	
1197	25504	305	076147		JSB			BXOUT811	SAVE
1198	25505	340	044607		IMM	LOW	IRCM	022B	FOR X-XU
1199	25506	343	160511		IMM	MPP2	LOW	L	370B
1200	25507	305	074547		JSB			BOXIN14A	X
1201	25510	010	040432		MPP1	PASS	MPPB	S1	XU
1202	25511	012	043047			AND	S2	S2	
1203	25512	010	042432		MPP1	PASS	MPPB	S2	
1204	25513	006	037107			ZERO	S3		
1205	25514	010	044432		MPP1	PASS	MPPB	S3	
1206	25515	340	000507		IMM	LOW	L	0	
1207	25516	014	047147			SANL	S4	S4	
1208	25517	010	046432		MPP1	PASS	MPPB	S4	
1209	25520	306	076402		JSB	CNDX	MPP	RJS	WAIT
1210	25521	340	124607		IMM	LOW	IRCM	052B	FOR (X-XU)*C
1211	25522	305	075047		JSB			BOXIN57A	PASS C, START.
1212	25523	230	074647		READ	PASS	M	P	READ CL
1213	25524	305	074007		JSB			RD57	
1214	25525	340	014647		IMM	LOW	M	006B	ADD TO SUM
1215	25526	305	075347		JSB			BOXIN811	
1216	25527	305	076147		JSB			BXOUT811	SAVE SUM
1217	25530	305	074347		JSB			BOXIN14C	XU IN XU*CL
1218	25531	305	075047		JSB			BOXIN57A	CL
1219	25532	340	014647		IMM	LOW	M	006B	ADD TO SUM
1220	25533	305	075347		JSB			BOXIN811	
1221	25534	007	173647	CMRT8		INC	Y	Y	SKIP ERROR RETURN.
1222	25535	325	067007		JMP			EXITA	WRITE & EXIT.

Appendix G

```

1224 * /ATLG - COMMON CODE FOR .ATAN & .LOG .
1225 *
1226 * /ATLG COMPUTES THE EXPRESSION X _ (1-X)/(1+X)
1227 *
1228 * CALLING SEQUENCE:
1229 *
1230 * OPCODE
1231 * DEF <X> (DIRECT)
1232 *
1233 *
1234 25536 000 033607 /ATLG
1235 25537 230 000647 READ DEC X M X=ADDR INST
1236 25540 010 075647 PASS M TAB READ X
1237 25541 305 074007 PASS Y P Y=ERROR RETURN.
1238 25542 306 076402 JSB RD57
1239 25543 340 044607 JSB CNDX MPP RJS WAIT WAIT FOR BOX.
1240 25544 356 177011 IMM LOW IRCM 022B FOR 1-X
1241 25545 006 037047 IMM MPP2 CMHI S1 277B S1-S4 = 1.0
1242 25546 006 037107 ZERO S2
1243 25547 353 173147 ZERO S3
1244 25550 305 074447 IMM CMLO S4 375B
1245 25551 305 075047 JSB BOXIN14B
1246 25552 340 004647 JSB BOXIN57A
1247 25553 305 076147 IMM LOW M 002B FOR 1+X
1248 25554 305 074447 JSB BXOUT811 SAVE 1-X
1249 25555 305 075047 JSB BOXIN14B
1250 25556 340 154656 JSB BOXIN57A
1251 25557 305 075347 IMM CLFL LOW M 066B FOR (1-X)/(1+X)
1252 JSB BOXIN811
1253 *
1254 * COMMON CODE TO WRITE RESULT AND EXIT.
1255 *
1256 25560 007 171547 EXITA INC SP X SP = ADDR ADDR RESULT.
1257 25561 305 077007 EXITB JSB RDPARAM RESOLVE IND
1258 25562 314 076142 EXITC JSB CNDX FLAG RJS BXOUT811 COPY RESULT
1259 25563 210 056036 WRTE MPCK PASS TAB S8 WRITE RESULT
1260 25564 007 133707 INC P M
1261 25565 007 174707 INC PNM P
1262 25566 210 060036 WRTE MPCK PASS TAB S9
1263 25567 007 174707 INC PNM P
1264 25570 210 062036 WRTE MPCK PASS TAB S10
1265 25571 010 074647 PASS M P
1266 25572 210 064036 WRTE MPCK PASS TAB S11
1267 25573 227 172707 EXITD READ INC PNM Y PREPARE TO EXIT.
1268 25574 375 141642 RTN CNDX OVFL RJS RTN IF NO OVFL
1269 25575 230 036777 READ IOFF TURN OF ALL BUT MP.
1270 25576 323 170002 JMP CNDX HOI RJS *+2 RTN IF NO MP INT
1271 25577 230 036753 READ COV COV IF MP AND OVFL
1272 25600 366 036143 RTN ION ZERO A RTN, CLR A, INTS ON

```

PAGE 0029 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

1274      *      .FPWR & .TPWR - EXPONENTIATION.
1275      *
1276      *      .FPWR & .TPWR RAISE A 2 OR 4-WORD NUMBER TO A POSITIVE
1277      *      POWER IN THE RANGE [2,65535]. FOR COMPATABILITY WITH
1278      *      THE SOFTWARE VERSIONS, POWERS ABOVE 32768 SHOULD NOT
1279      *      BE USED.
1280      *
1281      *      CALLING SEQUENCE:
1282      *
1283      *          LDA POWER
1284      *          OPCODE
1285      *          DEF <ARG>
1286      *
1287      *      FOR .FPWR, THE RESULT IS RETURNED IN (A,B). FOR .TPWR,
1288      *      THE RESULT OVERWRITES THE ARGUMENT.
1289      *
1290      *      OVERFLOW: MAX POS NUMBER RETURNED, OFL SET.
1291      *      UNDERFLOW: ZERO RETURNED, OFL CLEAR.
1292      *      OTHERWISE: OFL CLEAR.
1293      *
1294      *      *****
1295      *
1296      *      INITIALIZATION. START FIRST SQUARE.
1297      *
1298 25601 340 105516 .TPWR IMM CLFL LOW S11 042B 4-WD MPY OPCODE.
1299 25602 010 075547          PASS SP P SKIP RESULT ADDR.
1300 25603 325 070307 JMP          *+3
1301 25604 340 101517 .FPWR IMM STFL LOW S11 040B 2-WD MPY OPCODE.
1302 25605 010 033547          PASS SP 1 SP = ADDR ADDR ARG.
1303 25606 000 033613          COV DEC X 1 X = RESTART ADDR.
1304 25607 305 077007 JSB          RDPARAM READ ADDR.
1305 25610 305 073447 JSB          RD14
1306 25611 010 064607          PASS IRCM S11 SEND BOX OPCODE.
1307 25612 010 001151 MPP2 PASS S4 TAB (START BOX)
1308 25613 334 070702 JMP CNDX FLAG RJS FPWR1 IF 4-WORD.
1309 25614 010 041107          PASS S3 S1 SET UP 2ND ARG 2-WD.
1310 25615 010 043147          PASS S4 S2
1311 25616 305 074547 FPWR1 JSB          BOXIN14A 4-WD: 1ST / 2-WD: BOTH.
1312 25617 314 074542 JSB CNDX FLAG RJS BOXIN14A 2ND ARG 4-WD.
1313 25620 010 067647          PASS Y SP Y = RETURN ADDR.
1314 25621 010 041007          PASS S1 S1 IF BASE=0, RETURN ZERO.
1315 25622 320 033202 JMP CNDX ALZ FPWR5
1316 25623 353 146507 IMM          CMLO L 363B L=14B
1317 25624 010 165507          IOR S11 S11 SQUARE OPCODE.

```

Appendix G

PAGE 0030 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

1319          *      LEFT-JUSTIFY POWER, COUNT # BITS.
1320          *
1321 25625 010 007247 FPWRNORM      PASS S6  A      SAVE POWER FOR INT.
1322 25626 320 033202          JMP  CNDX ALZ      FPWR5      IF POWER=0, RETURN (CPND**2)
1323 25627 010 006207                   PASS B  A      SET UP NORMALIZE.
1324 25630 070 010224          LGS  R1  PASS B  3      UNNORMALIZE POWER
1325 25631 070 010224          LGS  R1  PASS B  B
1326 25632 340 000567          IMM  RPT  LOW  CNTR 0      NORMALIZE.
1327 25633 110 036762          NRM  L1  PASS
1328 25634 353 136525          IMM  DCNT  CNLO L  357B      L=16., CNTR=#BITS-16
1329 25635 003 027207                   ADD  S5  CNTR      S5 = # BITS - 2.
1330 25636 010 052147                   PASS A  S6      RESTORE A.
1331 25637 340 114647          IMM          LOW  M  0463      4-WD ACC*ARG OP.
1332 25640 001 110207                   DBLS B  B      SKIP SIGN (=0).
1333          *
1334          * THE FOLLOWING CHECK IS NECESSARY BECAUSE DMA ACTIVITY MAY
1335          * CAUSE THE NORMALIZE OP TO TERMINATE INCORRECTLY!!!
1336 25641 327 132302          JMP  CNDX AL15      FPWR3      IF MSB SET, NORM WAS OK
1337 25642 325 071247          JMP          FPWRNORM NO, GO TRY AGAIN
1338          *
1339          *      LOOP.  SQUARE, THEN MPY BY ARG IF EXP BIT SET.
1340          *
1341 25643 306 076402          FPWR2  JSB  CNDX MPP  RJS  WAIT      SQUARE ACC:
1342 25644 010 054607                   PASS  IRCM S11
1343 25645 340 114651          IMM  MPP2  LOW  H  046B      4-WD ACC*ARG OP.
1344 25646 001 110207          FPWR3          DBLS B  B      EXAMINE NEXT EXP BIT.
1345 25647 327 172742          JMP  CNDX AL15  RJS  FPWR4      IF NOT SET.
1346 25650 314 074402          JSB  CNDX FLAG  RJS  BOXIN14     USE "BOXIN14" FOR 4-WD.
1347 25651 334 072742          JMP  CNDX FLAG  RJS  FPWR4
1348 25652 306 076402          JSB  CNDX MPP  RJS  WAIT      DO 2-WD INLINE.
1349 25653 340 110607          IMM          LOW  IRCM 044B
1350 25654 010 036751          MPP2
1351 25655 010 040432          MPPL  PASS  MPP3 S1
1352 25656 010 042432          MPPL  PASS  MPP3 S2
1353 25657 000 051207          FPWR4          DEC  S5  S5      DEC LOOP CNT (#BITS-2)
1354 25660 327 133202          JMP  CNDX AL15      FPWR5      DONE.
1355 25661 323 172142          JMP  CNDX HOI  RJS  FPWR2      LOOP IF NO INTERRUPT.
1356 25662 336 072142          JMP  CNDX NSNG RJS  FPWR2
1357 25663 325 077347          JMP          INT      PROCESS. (RESTART)
1358          *
1359          *      RETURN RESULT.
1360          *
1361 25664 334 067002          FPWR5  JMP  CNDX FLAG  RJS  EXITA     IF 4-WD, WRITE RESULT.
1362 25665 007 172707                   INC  PNM  Y      SET UP RTN ADDR
1363 25666 306 076402          JSB  CNDX MPP  RJS  WAIT      2-WD. WAIT FOR BOX.
1364 25667 230 020172          READ MPPL  PASS A  MPPB      (A,B) = RESULT.
1365 25670 370 020232          RTN  MPPL  PASS B  MPPB      RETURN

```

PAGE 0031 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

1367          *          COMMON UTILITY ROUTINES.
1368          *****
1369          *          RD14 - FINISH READING TO S1-S4.
1370          *
1371 25671 010 001007 RD14          PASS S1  TAB
1372 25672 007 133707          INC P    M
1373 25673 227 174707          READ     INC PNM  P
1374 25674 007 174707                   INC PNM  P
1375 25675 230 001047          READ     PASS S2  TAB
1376 25676 007 174707                   INC PNM  P
1377 25677 230 001100          READ RTN PASS S3  TAB
1378          *****
1379          *          RD57 - FINISH READING TO S5-S7.
1380          *
1381 25700 010 001207 RD57          PASS S5  TAB
1382 25701 007 133707          INC P    M
1383 25702 227 174707          READ     INC PNM  P
1384 25703 007 174707                   INC PNM  P
1385 25704 230 001247          READ     PASS S6  TAB
1386 25705 007 174707                   INC PNM  P
1387 25706 230 001300          READ RTN PASS S7  TAB
1388          *****
1389          *          BOXIN14 - START BOX: OP=M, OPND=S1-14.
1390          *
1391 25707 340 104647 BOXIN14C IMM     LOW M    042B
1392 25710 306 076402 BOXIN14 JSB CNDX MPP RJS WAIT
1393 25711 010 032607 BOXIN14B          PASS IRCM M
1394 25712 010 036751          MPP2
1395 25713 010 040432 BOXIN14A MPP1 PASS MPPB S1
1396 25714 010 042432          MPP1 PASS MPPB S2
1397 25715 010 044432          MPP1 PASS MPPB S3
1398 25716 370 046432          RTN MPP1 PASS MPPB S4
1399          *****
1400          *          BOXIN57 - ISSUE MPP2, SEND S5-S7 & (d) TO BOX.
1401          *
1402 25717 306 076402 BOXIN57 JSB CNDX MPP RJS WAIT
1403 25720 340 014607          IMM     LOW IRCM 006B   SET UP ADD.
1404 25721 000 075711 BOXIN57A MPP2 DEC P    P
1405 25722 010 050432 BOXIN57B MPP1 PASS MPPB S5
1406 25723 227 174707          READ     INC PNM  P
1407 25724 010 052432          MPP1 PASS MPPB S6
1408 25725 010 054432          MPP1 PASS MPPB S7
1409 25726 370 000432          RTN MPP1 PASS MPPB TAB
1410          *****
1411          *          BOXIN811 - START BOX: OP=M, OPND=S8-S11.
1412          *
1413 25727 306 076402 BOXIN811 JSB CNDX MPP RJS WAIT
1414 25730 010 032607          PASS IRCM M
1415 25731 010 036751          MPP2
1416 25732 010 056432          MPP1 PASS MPPB S8
1417 25733 010 060432          MPP1 PASS MPPB S9
1418 25734 010 062432          MPP1 PASS MPPB S10
1419 25735 370 064432          RTN MPP1 PASS MPPB S11

```

Appendix G

PAGE 0032 RTE MICRO-ASSEMBLER REV. 1926 790308 10:40 AM TUE., 27 MAR.

```

1421          *          BOXOUT14 - COPY RESULT TO S1-S4.
1422          *
1423 25736 306 076402 BOXOUT14 JSB  CNDX MPP  RJS  WAIT
1424 25737 010 021032          MPP1 PASS S1  MPPB
1425 25740 010 021072          MPP1 PASS S2  MPPB
1426 25741 010 021132          MPP1 PASS S3  MPPB
1427 25742 370 021172          RTN  MPP1 PASS S4  MPPB
1428          *****
1429          *          BXOUTS11 - COPY RESULT TO S3-S11.
1430          *
1431 25743 306 076402 BXOUT811 JSB  CNDX MPP  RJS  WAIT
1432 25744 010 021372          MPP1 PASS S8  MPPB
1433 25745 010 021432          MPP1 PASS S9  MPPB
1434 25746 010 021472          MPP1 PASS S10 MPPB
1435 25747 370 021532          RTN  MPP1 PASS S11 MPPB
1436          *****
1437          *          BOX WAIT ROUTINE.
1438          *
1439 25750 366 000402 WAIT      RTN  CNDX MPP          RETURN IF DONE.
1440 25751 340 100547          IMM          LOW  CNTR 0403  COUNTER=32
1441 25752 366 002002 WAIT2     RTN  CNDX MPP          RETURN IF DONE
1442 25753 010 036765          DCNT          DECR COUNTER
1443 25754 366 000742          RTN  CNDX MPP          RETURN IF DONE
1444 25755 326 176502          JMP  CNDX CNT8 RJS  WAIT2     LOOP 32 TIMES
1445 25756 007 173707          INC  P          Y          ** BOX HUNG **
1446 25757 325 003547          JMP          BAILOUT  USE SIS BOX HUNG CODE.
1447          *****
1448          *          INDIRECT RESOLVER.
1449          *
1450 25760 230 066647 RDPARAM READ          PASS M  SP
1451 25761 007 167547          INC  SP  SP
1452 25762 230 000647 INDIRECT READ          PASS M  TAB          NEXT ADDRESS
1453 25763 367 140002          RTN  CNDX AL15 RJS          RETURN IF 15 POS.
1454 25764 323 177102          JMP  CNDX HOI  RJS  INDIRECT  LOOP IF NO INT
1455 25765 230 036747          READ          MUST REREAD
1456 25766 336 077102          JMP  CNDX NSNG RJS  INDIRECT  IGNORE SNGL STEP
1457          *****
1458          *          INTERRUPT HANDLING.
1459          *
1460 25767 227 170732 INT      READ MPP1 INC  PNM  X          REDO INSTRUCTION.
1461 25770 010 036753          COV          CLEAR OVERFLOW
1462 25771 320 000007          JMP          FETCH
1463          *
1464          *
1465          END

```

END OF PASS 2: NO ERRORS

PAGE 0035 MICRO XREF REV. 1926 790322 10:40 AM TUE., 27 MAR.

FPWR2	1341	1355	1356
FPWR3	1344	1336	
FPWR4	1353	1345	1347
FPWR5	1361	1315	1322 1354
FPWRNORM	1321	1337	
FSUB	0255	0618	
FSUE2	0159	0528	0667 0831
HORI	0021	0149	
INDIRECT	1452	1454	1456
INT	1460	1045	1357
INIBT1	0145	0125	
LGNXT	0503	0499	
LOGERR	0556	0494	0495
LOCF	1040	1048	
LOOP1	0126	0129	0145
LOOP2	1046	1044	
LOOP3	1047	1025	
LOOPADD	1042	1006	
NEG	0935	0928	
NEGATE	0251	0474	0670
NEGCHK	0813	**NOT REFERENCED**	
NZERO	1034	1017	
ODD	0743	0728	
PDONE	0091	0065	
PNEXT1	0072	0069	
PNEXT2	0085	0080	0082
POS	0596	0593	0594
POSCHK	0818	0811	

Appendix G

PAGE 0036 MICRO XREF REV. 1926 790322 10:40 AM TUE., 27 MAR.

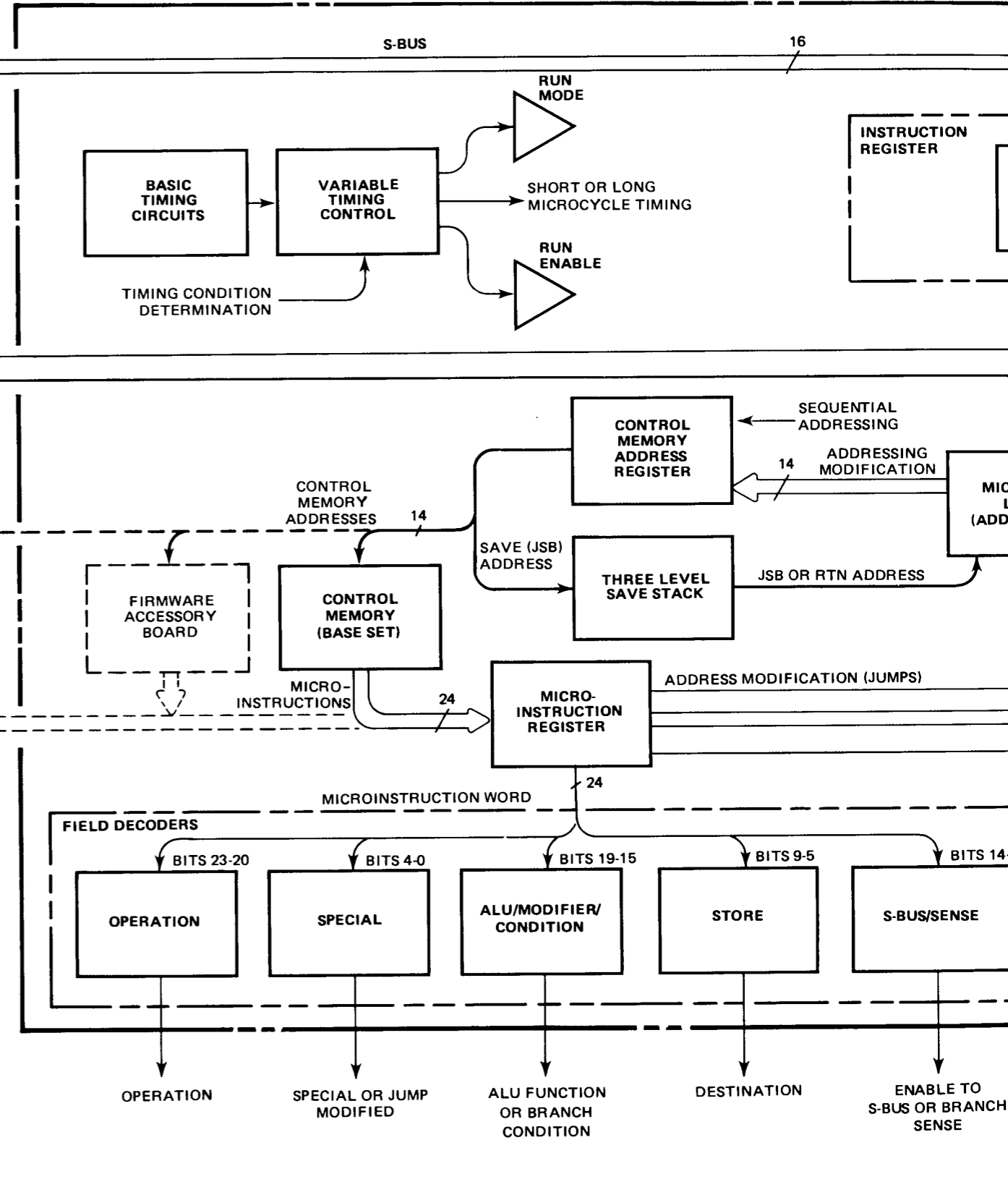
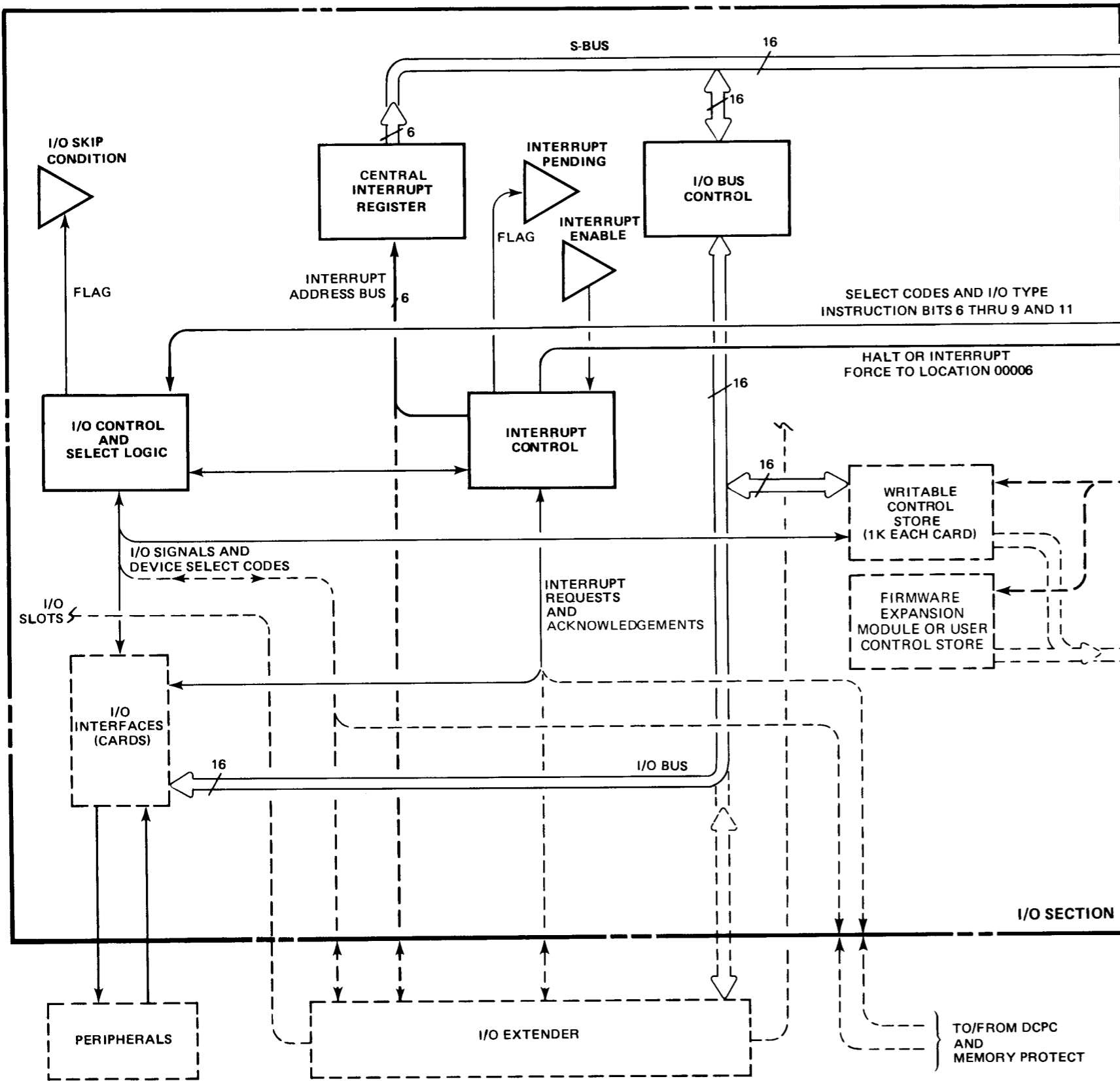
PWR2	0063	0775	0844							
RE14	1371	0983	1103	1305						
RD57	1381	0993	1002	1008	1043	1099	1213	1237		
RDPARAM	1450	0982	0992	1001	1004	1015	1098	1102	1257	1304
REDUCE	0278	0803								
RNXT	0302	0293	0300							
SELFTEST	0692	0039								
SIN	0397	0029	0394							
SINAG	0437	0406								
SINERR	0478	0401								
SQERR	0778	0724								
SQRT	0721	0025	0698							
TAN	0320	0024								
TANERR	0375	0323								
TANH	0869	0032								
TANH1	0894	0874								
TANH2	0926	0877								
TRNL	0975	0033								
TRNL1	0978	0976								
TRNL1A	1000	0985								
TRNL1B	1003	0996								
VMPY	0239	0425	0431	0454	0462	0544				
WAIT	1439	1122	1182	1209	1238	1341	1348	1363	1392	1402
	1413	1423	1431							
WAIT1	0124	0063	0105	0159	0171	0183	0195	0213	0227	0239
	0255	0288	0294	0305	0333	0369	0414	0443	0508	0513
	0518	0615	0619	0624	0635	0672	0760	0769	0887	
WAIT2	1441	1444								
XSQ	0195	0326	0405	0823						
ZERO1	0679	0592								

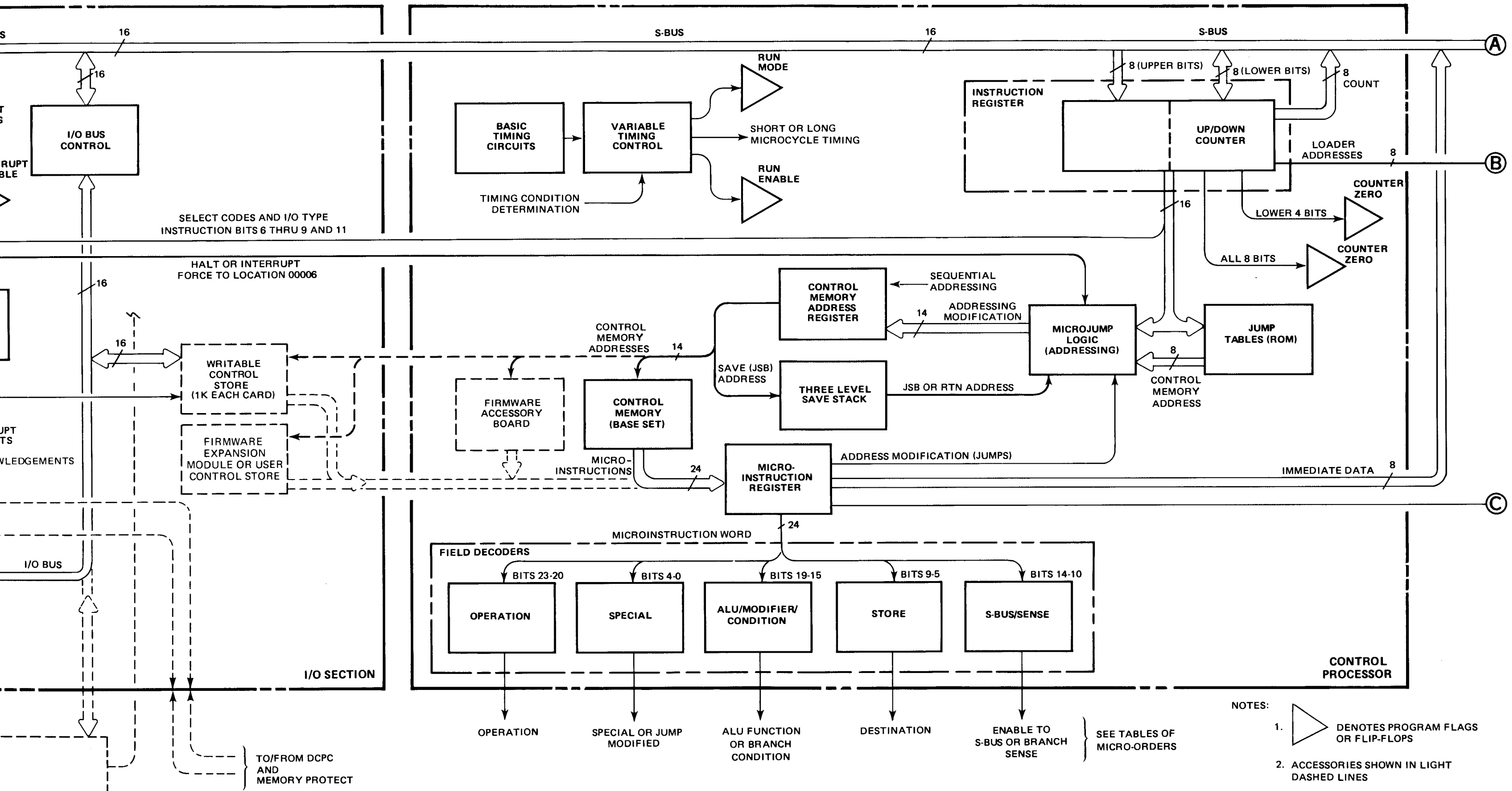
1
PAGE 0037 MICRO XREF REV. 1926 790322 10:40 AM TUE., 27 MAR.

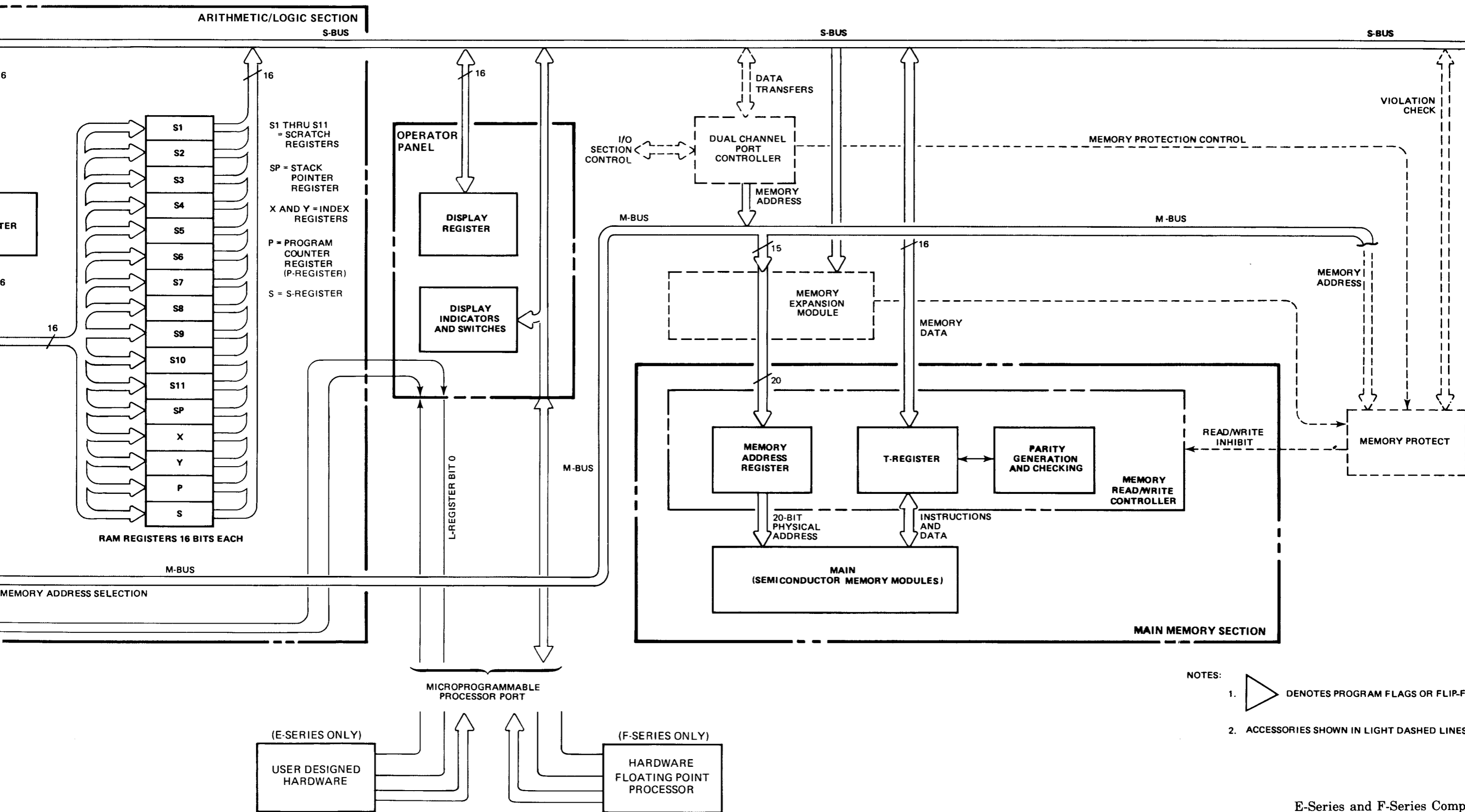
ZERO2 0850 0815

Appendix H
FUNCTIONAL BLOCK DIAGRAM









A

A (Store or S-bus Field), Table 4-1
 A-Addressable Flip-Flop (AAF), Table 2-1
 A-Register, Table 2-1
 Abbreviations, Appendix A
 Accessing A Parameter, 6-9
 Activity Profile, 1-2
 ADD, Table 4-1
 Address Field Word Type III, Table 4-1
 Address Field Word Type IV, Table 4-1
 Address Field Microassembler, 8-17
 AL0, Table 4-1
 AL15, Table 4-1
 ALGN Pseudo-Microinstruction, 8-21
 ALU Field Word Type, Table 4-1
 ALZ, Table 4-1
 Analysis Method, 1-2
 AND, Table 4-1
 Arithmetic Logic Unit (ALU), Table 2-1, 2-3
 Arithmetic/Logic Operations, 5-9
 ASG, Table 4-1, 7-8
 Assembler Interface Program, 14-3
 Assembler Procedure, 6-8
 Assembly Language Instruction, 2-12, 2-17

B

B (Store or S-bus Field), Table 4-1
 B-Addressable Flip-Flop (BAF), Table 2-1
 B-Register, Table 2-1
 Base Set Listing, Appendix G
 Base Set Modules, 2-15
 Base Set Operation, 2-14
 Binary Field Micro-Order Summary, Appendix C
 Binary Microcode Format, Appendix B
 Binary Object Code, 9-4
 Binary Structures, 4-1, Appendix C
 BIOI, 13-1
 BIOO, 13-1
 BIOS, 13-1
 Block Diagram, Appendix H
 Block I/O, 13-1
 Block I/O Address/Data Burst Input, 13-3
 Block I/O Byte Packing Burst Input, 13-2
 Block I/O Data Transfers, 13-1
 Block I/O Word Burst Output, 13-6
 Branch Field Word Type III, Table 4-1
 Branch Field Word Type IV, Table 4-1
 Branches, Control Memory, 5-10
 BREAKPOINT Command (MDE), 10-11
 Bus System, Table 2-1, Appendix H

C

CAB (Store or S-bus Field), Table 4-1
 Calling MDE, 10-19
 Calling Microprograms from FORTRAN, 6-15
 Central Interrupt Register (CIR), Table 2-1
 CIR, Table 4-1
 CLEAR Command (MDE), 10-12
 CLFL, Table 4-1
 CM/Main Memory Linkage, 6-10
 CMHI, Table 4-1
 CMLO, Table 4-1
 CMPL, Table 4-1
 CNDX, 4-5
 CNDX, Table 4-1
 CNT4, Table 4-1
 CNT8, Table 4-1
 CNTR (Store or S-bus Field), Table 4-1, 7-11
 CNTR Micro-Order, 7-10
 Comment Field, 8-18
 Computer Functions, 2-1, 2-10
 Conditional and Invalid Operations, 7-5
 Conditional Field Word Type III, Table 4-1
 Conditional Flags, Table 2-1
 Conditional Microbranches, 4-5
 Considerations, Section 7
 Constant Storage, 4-4
 Contributed Library Catalog, 1-2
 Control Commands, Microassembler, 8-8
 Control Memory (CM), Table 2-1, 2-3, 2-13
 Control Memory Address Register (CMAR), Table 2-1
 Control Memory Boards, 3-2
 Control Memory Mapping Method, 6-1
 Control Memory Maps, 2-13
 Control Processor Block Diagram, 2-2
 Control Processor, 2-1, 2-10, 2-15
 Controllable Functions, 2-1, 2-10
 Conventional Control Section, 2-10
 COUT, Table 4-1
 COV, Table 4-1
 Cross Reference Generator, 9-6
 CRS, Table 4-1

D

DBLS, Table 4-1
 DCNT, Table 4-1
 DCPC, See Dual Channel Port Controller
 DEC, Table 4-1
 DEF Pseudo-Microinstruction, 8-24
 Definitions and Abbreviations, Appendix A
 Definitions and Timing Points, Table 2-1, 2-11
 DELETE Command (MDE), 10-8

Index

DES, Table 4-1
Display Indicator (DSPI), Table 2-1
Display Register (DSPL), Table 2-1
DIV, Table 4-1
DMA, See Dual Channel Port Controller
DMS (MEU) Instruction Listing, Appendix G
DMS Considerations, 7-33
Driver DVR36, 3-7
DSP1 (Store or S-bus Field), Table 4-1
DSPL (Store or S-bus Field), Table 4-1
Dual Channel Port Controller (DCPC), 2-9, 7-21
DUMP Command (MDE), 10-5
DVR36 and WLOAD Use Summary, 3-7, 11-2
Dynamic Mapping System (DMS), 2-8, 3-2, 7-33

E

E, Table 4-1
E-/F- to M-Series Comparison Summary, Appendix F
E-Intervals, 5-2
END Pseudo-Microinstruction, 8-22
Entry Points, 6-2
ENV, Table 4-1
ENVE, Table 4-1
Environment, 3-1
EQU Pseudo-Microinstruction, 8-23
Error Messages, 9-10, 10-17, 12-5
Examples, Section 14
Executing Microprogrammed I/O Instructions, 7-30
Execution, 2-17
Execution, Assembly Language Instruction, 2-17
Execution Command, Microassembler, 9-2
Execution Times, FPP, 13-32
EXIT Command (MDE), 10-4
Extend Register, Table 2-1

F

Fetching, 2-16
Field Decoders, Table 2-1
Field Template, 8-4
Fields, 8-4
Fields, Binary, 4-1
Fields, Microassembler, 4-2
Firmware Accessory Board (FAB), 1-3
Firmware Expansion Module (FEM), 1-3
FLAG, Table 4-1
Flip-Flops, Addressable, Table 2-1
Floating Point Processor (FPP), 13-10
Floating Point Processory, Hardware, 13-10
Format, Binary Microcode, Appendix B
Formats, Microassembler, 4-2, Appendix B
FORTRAN, Calling Microprograms From, 6-15
FTCH, Table 4-1, 7-14, Appendix C
FPP, 13-10
FPP Accumulator Operations, 13-20
FPP Addressing, 13-23
FPP Complete Test, 13-28
FPP Controllable Functions, 13-11
FPP Data Formats, 13-12

FPP Data Operations, 13-18
FPP Execution Process, 13-33
FPP Execution Times, 13-32
FPP Exponent Format, 13-14
FPP Fix and Float Operations, 13-19
FPP Instruction Execution, 13-24
FPP Instruction Store, 13-22
FPP Instruction Word Format, 13-13
FPP Interrupt Considerations, 13-34
FPP Microprogram Example, 13-35
FPP Microprogramming Rules, 13-36
FPP MPP Micro-Orders, 13-21
FPP Operand Length, 13-17
FPP Operand Source, 13-16
FPP, Operand to, 13-25
FPP Operation, 13-15
FPP Overflow Detection, 13-29
FPP Result to CPU, 13-26
Freeze, 5-6
Functions, Computer, 2-1, 2-10, 2-12
Functions, Control Processor, 2-2, 2-10, 2-15

G

General Tape Format, 12-2
Guidelines for Writing Loaders, 7-34

H

Hardware Floating Point Processor (FPP), 13-10
HIGH, Table 4-1
HOI, Table 4-1
HP 1000 E-Series and F-Series Microinstructions, 8-14

I

IAK, Table 4-1, 7-19
ICNT, Table 4-1
IMM, Table 4-1, 4-4
INCI, Table 4-1, 7-16, Appendix C
Indirect Reference Resolution, 6-9
Initialize Phase, 12-2
Input/Output Section, 2-5
Instruction Register (IR), Table 2-1
Interrelated Functions, 2-12
Interrupt Control, Table 2-1
Interrupt Handling (IA) bus, Table 2-1, 7-29
Interrupt Handling, 7-29
Interrupt, FPP, 13-34
I/O bus, Table 2-1
I/O Control, Table 2-1, 7-25
I/O Input, 7-27
I/O Instructions, Microprogrammed, 7-22, 7-30
I/O Micro-Order Summary, 7-32
I/O Microprogrammed Example, 14-4
I/O Operation, 5-11
I/O Output, 7-26
I/O Relation to Memory Protection, 7-28
I/O Select Logic, Table 2-1
I/O Signal Generation, 7-24

I/O Signals, 7-25
 I/O Special Techniques, 7-31
 I/O Synchronizing, 7-23
 I/O Timing Operations, 5-11
 IOFF (Modifier/Special Field), Table 4-1
 IOFF Considerations, 7-20
 IOG (Modifier/Special Field), Table 4-1
 IOG Considerations, 7-18, 13-1
 IOI, Table 4-1, 13-1
 ION, Table 4-1
 IOO, Table 4-1, 13-1
 IOR, Table 4-1
 IR11, Table 4-1
 IR8, Table 4-1
 IRCM Considerations, 2-15, Table 4-1, 7-15
 IRCM (FPP), 13-32

J

J74, Table 4-1, Appendix C
 JMP (Branch Field), Table 4-1
 JMP, Table 4-1, 4-6
 JSB (Branch Field), Table 4-1
 JTAB, Table 4-1, Appendix C
 Jump Tables, Table 2-1, 2-14, Appendix G

L

L, Table 4-1
 L-Register, Table 2-1
 L1, Table 4-1
 L4, Table 4-1
 L15, Table 4-1
 Label Field, 8-15
 LDR, Table 4-1
 LGS, Table 4-1
 Library, Contributed Catalog, 1-2
 Linkage, Control Memory/Main Memory, 6-10
 LO, Table 4-1
 LOAD Command (MDE), 10-6
 Loaders, Table 2-1, 7-34
 Loading Microprogramming Support Software, 3-9
 LOCATE Command (MDE), 10-13
 LOW, Table 4-1
 LU Command (MDE), 10-7
 LWF, Table 4-1

M

M (Store or S-bus Field), Table 4-1
 M- to E-/F-Series Comparison Summary, Appendix F
 M-bus, Table 2-1
 M-Register, Table 2-1
 M-Series Micro-Orders, Appendix F
 MAC0, Appendix C
 MAC1, Appendix C
 MACTABL0, Appendix C
 MACTABL1, Appendix C
 Magnitude Tests, 7-12
 Main Memory Address 00 and 01, 7-2

Main Memory Operations, 5-12
 Main Memory Procedures, 6-11
 Main Memory Section, Table 4-1, 2-4
 Main Memory/Control Memory Linkage, 6-7
 Manual/Software Reference, 3-10
 Mapping Control Memory- Section 6
 Mapping Details, 6-2, Appendix C
 Mapping, See Dynamic Mapping System and DMS
 MBIO and MPP Considerations, 13-7
 MDE Calling, 10-19
 MDE Commands, 10-2
 MDE Messages, 10-17
 MDE Operator Command Syntax, 10-1
 MDE Restrictions, 10-18
 MDE Scheduling, 10-1, 10-13
 MDE Sequence of Operations, 10-19
 MDE, See Microdebug Editor
 MDEP, See Microdebug Editor
 MDES, See Microdebug Editor
 MEM Signals, 7-33
 MEM, See Memory Expansion Module
 Memory Address Register, Table 2-1
 Memory Expansion Module (MEM), 2-8
 Memory Protect Considerations, 7-13
 Memory Protect, 2-7
 Memory Protection Relation to I/O, 7-28
 Memory Timing Operations, 5-12
 MESP, Table 4-1
 Messages (MDE), 10-17
 Messages, Error and Informative
 (MICRO and MXREF), 9-8
 MEU Micro-Order (Store or S-bus Field), Table 4-1,
 Table 7-3, 7-26
 MIC Pseudo-Instruction, 6-12
 MIC Use Example, 6-14
 MICMXE, 8-9
 MICRO, 9-1, Also See Microassembler
 Micro-Order, Binary Field Summary, Appendix C
 Micro-Order Binary Formats, 4-1
 Micro-Order Comparison Summary, Appendix F
 Micro-Order Definitions, 4-7
 Micro-Order Summary, Appendix C
 Micro-Orders, 2-10, Table 4-1, 8-16
 Micro-Orders, Special Use, Appendix C
 Microassembler \$CODE Command, 8-10
 Microassembler \$LIST and \$NOLIST Commands, 8-12
 Microassembler \$PAGE Command, 8-11
 Microassembler \$PUNCH and \$NOPUNCH
 Commands, 8-13
 Microassembler ?? Command (MDE), 10-3
 Microassembler Assembly Command MIC, 8-9
 Microassembler Binary Object Code, 9-4
 Microassembler Control Commands, 8-8
 Microassembler Cross-Reference Generator, 3-5, 9-7
 Microassembler Description, 8-6
 Microassembler Error Messages, 9-10
 Microassembler Execution Command, 9-2
 Microassembler Fields, 4-2, 8-14
 Microassembler Formats, 4-2, Appendix B
 Microassembler Informative Messages, 9-9
 Microassembler Listing Output, 9-5
 Microassembler Messages, 9-8

Index

Microassembler Output, 9-3
Microassembler Planning and Preparation, 8-1
Microassembler Preliminary Information, 8-3
Microassembler Rules, 8-7
Microassembler Symbol Table Output, 9-6
Microassembler, 3-4, 8-6, 9-1
Microassembler, Using, 9-1
Microbranches, Conditional, 4-5
Microbranches, Unconditional, 4-6
Microcycle Estimating Flowchart, 5-7
Microcycle, 2-11
Microcycle, Long, 5-2
Microcycle, Short, 5-2
Microcycle, Variable, 5-6
Microdebug Editor, 1-3, Also See MDE
Microinstruction Binary Structures, 4-1
Microinstruction Formats, Section 4, Appendix B
Microinstruction Register (MIR), Table 2-1
Microinstruction, 2-10, 4-1, 8-14
Microinstruction, Pseudo, 8-19
Microjump Logic (MJL), Table 2-1
Microprogram Entry, 8-5
Microprogram Planning, 8-2
Microprogram Preparation, Section 8
Microprogram, Base Set Listing, Appendix F
Microprogram, Shell Sort Example, 14-3
Microprogrammable Processor Port (MPP), 13-5
Microprogrammed I/O Operation, 14-4
Microprogrammed I/O, 7-22
Microprogramming Accessories, 2-18
Microprogramming Concept, 1-1
Microprogramming Considerations, Section 7
Microprogramming Environment, 3-1
Microprogramming Examples, Section 14
Microprogramming Execution, 1-4
Microprogramming Form, Appendix D
Microprogramming Hardware, 3-2, 12-6
Microprogramming Overview, 1-1
Microprogramming Preparation, Section 3
Microprogramming Process, 1-3
Microprogramming Related Products, 1-5
Microprogramming Support Software, 3-3
Microprogramming Techniques, 7-6
Microprogramming the Floating Point Processor, 13-36
Microprogramming, User 2-10
Microsubroutine, 2-11
Modified Privileged Driver, 14-4
Modifier Field Word Type II, Table 4-1
Module Selection, Appendix C
MPBEN Signal, Table 13-1
MPBST Signal, Table 13-1
MPCK Use, 7-4
MPCK, Table 4-1, 7-17
MPP and MBIO Considerations, 13-8
MPP Hardware Interface, 13-6
MPP Microprogram, 13-7
MPP Signal Summary, 13-7
MPP Signals, Table 13-1
MPP, Table 4-1, 13-5
MPP1, Table 4-1, 13-27
MPP2, Table 4-1, 13-24
MPPA and MPPB (Store or S-bus Field), Table 4-1

MPPIO 0 thru 15 Signals, Table 13-1
MPY, Table 4-1
MRG, Table 4-1
MXREF, 9-7, Also See Microassembler

N

NAND, Table 4-1
NDEC, Table 4-1
NINC, Table 4-1
NINT, Table 4-1
NLDR, Table 4-1
NLT, Table 4-1
NMDE, Table 4-1
NMDE, Table 4-1
NMLS, Table 4-1
NOP (OP, Special, Store, or S-bus Field), Table 4-1
NOR, Table 4-1
NRM, Table 4-1
NRT, Table 4-1
NSAL, Table 4-1
NSFP, Table 4-1
NSNG, Table 4-1
NSOL, Table 4-1
NSTB, Table 4-1
NSTR, Table 4-1

O

Object Microcode, 9-4, Appendix B
Object Tape Formats, Appendix E
ONE, Table 4-1
ONES and ZERO Pseudo-Microinstruction, 8-25
ONES, Table 4-1
OP Fields, Word Type, Table 4-1, 4-7
OP1 through OP11 and OP13, Table 4-1
Operand Field Word Type II, Table 4-1
Operand to FPP, 13-25
Operational Overview, 2-15
Operator Panel, Table 2-1, 2-6
ORG Pseudo-Microinstruction, 8-20
Overall Block Diagram, 2-3, Appendix H
Overall Timing, 5-7
Overflow Detection, FPP, 13-29
Overflow Register, Table 2-1
Overflow, 7-9
OVFL, Table 4-1

P

P (Store or S-bus Field), Table 4-1
P-Interval, 5-2, 5-5
Parameter Accessing, 6-9
Parameter Assignment Example, 6-13
Parameter Passing, 6-9
PARAMETERS Command (MDE), 10-14
PASL, Table 4-1
PASS, Table 4-1
Pause, 5-5

PIRST Signal, Table 13-1
 Planning, 8-2
 PLRO Signal, Table 13-1
 PNM, Table 4-1, 7-10
 PP1SP Signal, Table 13-1
 PP2SP Signal, Table 13-1
 PP5 Signal, Table 13-1
 Preparatory Steps, 3-1, 3-10
 Programmable Read Only Memory (pROM), 1-3
 pROM Generation, Section 12
 pROM Hardware, 12-6
 pROM Tape Generator, 3-9, 12-1
 PRST, Table 4-1, 7-2
 Pseudo-Microinstructions, 8-19
 PTGEN Error Messages, 12-5
 PTGEN Initialize Phase, 12-2
 PTGEN Punch Phase, 12-3
 PTGEN Verify Phase, 12-4
 PTGEN, 12-1
 Punch Phase, 12-3

Q

Q0-Q7 Control Signals, 7-33

R

R1, Table 4-1
 RAM Registers, Table 2-1
 Read and Write Considerations, 7-1
 Read Operation Examples, 5-13, 7-2
 READ, Table 4-1
 READ/WRITE Conditional and Invalid Operations, 7-5
 Reading from Memory, 5-13
 Real Time Executive, 3-1
 Registers, Table 2-1
 REPLACE Command (MDE), 10-9
 Reserved UIG Codes, 6-4
 Restrictions, Microdebug Editor, 10-18
 RJ30 (Modifier/Special Field), Table 4-1
 RJS, Table 4-1
 Rotate Shifter (R/S), Table 2-1
 RPT, Table 4-1
 RTE Microassembler, 3-4, Sections 8, 9
 RTE Microdebug Editor, 3-6, Section 10
 RTE, 3-1
 RTN (OP or Branch Field), Table 4-1
 RUN Command (MDE), 10-15
 RUN, Table 4-1
 RUNE, Table 4-1

S

S (Store or S-bus Field), Table 4-1
 S-bus Field Word Type, Table 4-1

S-bus, Table 2-1
 S1 through S11 (Store or S-bus Field), Table 4-1
 Sample Privileged Driver, 14-4
 SANL, Table 4-1
 Save Stack, Table 2-1
 Scientific Instruction Set Listing, Appendix G
 Select Code (SC) `bus, Table 2-1
 SET Command (MDE), 10-16
 Shell Sort Assembler Program, 14-3
 Shell Sort Example, 14-3
 SHLT, Table 4-1
 Short/Long Microcycles, 5-4
 SHOW Command (MDE), 10-10
 Signals, BIOI, BIOO, BIOS, 13-1
 Signals, MPP, 13-1
 SKPF, Table 4-1
 Software Entry Point Assignments, 6-2
 SONL, Table 4-1
 SOV, Table 4-1
 SP (Store or S-bus Field), Table 4-1
 Special Facilities, Computer, Section 13
 Special Field Word Type, Table 4-1
 Special Use Micro-Orders, Appendix C
 Specialized Microprogramming, Appendix C
 SRG1 and SRG2, Table 4-1, 7-7
 SRUN, Table 4-1
 STFL, Table 4-1
 Store Field Word Type I and II, Table 4-1
 STOV Signal, Table 13-1
 SUB, Table 4-1
 Summary, Controllable Functions, 2-19
 Summary, I/O Micro-Orders, 7-32
 Summary, Mapping to User's Area, 6-16
 Summary, Microinstruction Formats, 4-8
 Summary, Microprogramming Concepts, 1-6
 Summary, Microprogramming Considerations, 7-35
 Summary, MPP, 13-34
 Summary, Preparation With The Microassembler, 8-26
 Summary, Special Facilities, 13-10
 Summary, Timing Considerations, 5-15
 Symbol Table, 9-6
 Synchronizing with I/O, 7-23

T

T-bus, Table 2-1
 T-period, 2-11
 T-Register, Table 2-1
 TAB (Store or S-bus Field), Table 4-1, 7-2
 Test Programs, 14-3
 Timing Calculations, 5-8
 Timing Control Memory Branches, 5-10
 Timing Definitions, 5-2
 Timing I/O Operations, 5-11
 Timing Main Memory Operations, 5-12
 Timing Variables, 5-3
 Timing, 5-1
 Timing, Special Facilities Transfer Rate Summary, 13-33
 Timing Considerations, Section 5

Index

U

UCS, See User Control Store, 1-3
UIG Codes, 6-3, 6-5
UIG Decoding, Appendix C
UIG Mapping, Appendix C
Unconditional Microbranches, 4-6
User Area UIG Codes, 6-5
User Control Store (UCS), 1-3
User Instruction Group (UIG), 6-5
User Microprogramming, 2-10
User's Area Mapping Example, 6-6

V

Variable Microcycle with Pause Conditions, 5-8
Vendor Default Formats, 12-1
Verify Phase, 12-4

W

WCS Hardware, 11-1
WCS Initialization, 11-2, 14-1

WCS Software, 11-2
WCS Support Software, 11-1
WLOAD, 3-8, Section 11
Word Type I, 4-3
Word Type II, 4-4
Word Type III, 4-5
Word Type IV, 4-6
Word Type/Binary Format, 4-1
Writable Control Store (WCS), 3-2
Write Operation Example, 5-14, 7-3
Write, Table 4-1
WRITE/READ Conditional and Invalid
Operations, 7-5
Writing from Memory, 5-14

X Y Z

X (Store or S-bus Field), Table 4-1
XNOR, Table 4-1
XOR, Table 4-1
Y (Store or S-bus Field), Table 4-1
ZERO and ONES Pseudo-Microinstructions, 8-25
ZERO, Table 4-1