HEWLETT hp PACKARD

2100 computer

microprogramming software

HEWLETT hp PACKARD

MICROPROGRAMMING SOFTWARE

for

Hewlett-Packard Model 2100 Computer

HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA U.S.A.

02100-90133 Printed: AUG 1972

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

ERRATA LIST

2100 COMPUTER
MICROPROGRAMMING SOFTWARE

Part No. 02100-90133, Printed Aug. 1972

The italics show which part of the text has been changed.

Page	Instructions
2-1	Replace the last two paragraphs with the following: "The HP Microassembler is designed to operate in a Basic Control System (BCS) environment and requires a minimum of 8K of memory. In addition, it requires a tape punch device (the teleprinter's tape punch may be used for this purpose). "The operating instructions for loading and executing the HP Microassembler are as described on pages BCS-11 through BCS-14 of the BASIC CONTROL SYSTEM module (5951-1391) of the Software Operating Procedures manual. Even though it is described as being optional (step 6, page BCS-12), the BCS Relocatable Subroutine Library must be loaded at step 7. "Note: In an 8K BCS environment, neither the magnetic tape driver nor the buffered version of IOC should be used. In addition, the user must select the absolute output option (bit-14 of Switch Register ON) during step 3 of the loading procedure."
2-2	3rd paragraph. Change the final sentence to: "*During* Pass 2, the assembly listing is printed *and the* object microprogram is punched."

Page	Instructions

4th paragraph. Delete the 3rd sentence and change the 4th sentence to: "Fatal messages, on the other hand, *draw the microprogrammer's attention to illegal microprogramming usage which must be corrected.*"

2-3 The following note belongs between the 2nd and 3rd paragraphs:

"Note: While the microprogrammer may use symbolic addresses as jump addresses in JMP, JSB, and CJMP microinstructions, he may not use a symbolic address ± a constant as a jump address."

5-1, 5-2 The format of the $INPUT, $PASS2, $LIST, and $OUTPUT statements should be:

$INPUT=*x*
$PASS2=*x*
$LIST=*x*
$OUTPUT=*x*

The format of the $EXTERNALS statement should be:

$EXTERNALS=name 1 ∧ octal address 1, . . . ,name n ∧ octal address n

where ∧ is a space.

5-3 Replace the description of $DEBUG with the following:

"Specifies that the debug option is to be used. *Note that the debug option requires that the microprogram be smaller than* 272_8 *(*186_{10}*) locations long.* See Section 9, 'Micro Debug Editor,' of this manual for further details."

Page	Instructions

8-3 Message #14 should be flagged by a pair of asterisks and the "Meaning" and "Corrective Action" entries should be:

"Warning! JMP, JSB, or CJMP in the Function field and a non-NOP in the S-bus field.	JMP, JSB, and CJMP use the low-order bit of the S-bus field as part of the jump address. Make certain that the S-bus micro-order does not set this bit incorrectly."

9-8 Add the following to the description of VERIFY ahead of the sentence which begins "If no errors are detected . . .":

"In response to a VERIFY command, the editor asks the microprogrammer to identify which of the six tapes is to be verified. The microprogrammer responds by entering one of the following tape I.D. numbers:

I.D. Number	Tape
2320	Identifies the mask tape which contains bits 23 through 20 of all WCS words.
1916	Identifies the mask tape which contains bits 19 through 16 of all WCS words.
1512	Identifies the mask tape which contains bits 15 through 12 of all WCS words.
1108	Identifies the mask tape which contains bits 11 through 8 of all WCS words.
0704	Identifies the mask tape which contains bits 7 through 4 of all WCS words.
0300	Identifies the mask tape which contains bits 3 through 0 of all WCS words.

Page	Instructions
9-10	2nd paragraph. Delete the 2nd sentence, beginning "The microprogram should then . . .".

Add the following after the 3rd paragraph:

"The editor's dump routine uses core memory location 0 for temporary storage. If the microprogram being debugged uses core location 0, the microprogrammer should remember that the contents of that location are altered every time a breakpoint is encountered. Also, since the editor's dump routine occupies control store locations 272_8 through 377_8, the microprogrammer should not set a breakpoint above control store location 271_8.

"The editor's dump routine executes an EOP. Among other things, the EOP clears the JSB flip-flop. Consequently, if the breakpoint occurred within a subroutine, execution must not be restarted within the subroutine because the RSB at the end of the subroutine will not work as expected. After such a breakpoint, the microprogrammer should restart execution either from the beginning (EXECUTE,0) or from some location (EXECUTE,xxxx) which would not allow the subroutine's RSB to be executed."

9-13	Delete the last sentence.
9-14	Step 4. Delete the 1st sentence and change the 2nd sentence to: "If the editor is to be run in the normal mode, the user must force program *loading* at this point even though there *are two* undefined external symbols (TEST *and MACRO*)."
9-16	Change the final sentence of step 1 to: "Each segment must be able to be entered by using the same 105xxx macro instruction *and operate independently of the other segments.*"

Page	Instructions
10-1	Add the following topic between "Requirements" and "Initial Parameters":

LOADING INSTRUCTIONS

To load the HP Programmable ROM Writer program, do as follows:

1. Load BCS using the Basic Binary Loader.
2. Load the HP Programmable ROM Writer program using the BCS Relocating Loader.
3. When BCS prints the message "RUN" on the system console device, enter the select code of the HP 12909A Programmable ROM Writer into the Switch Register and then press RUN.

Note: If the user forgets to enter a value into the Switch Register and merely presses RUN, the program reacts in either of the following ways:

- If the Switch Register contains all zeros, the program halts with 102022 in the Memory Data register. The user responds by entering the select code into the Switch Register and then pressing RUN.
- If the Switch Register contains a non-zero value, the program accepts the specified value as the select code and proceeds with execution.

10-3 Add the following between the 2nd paragraph and "General Operation":

TIMING CONSTANT?

The user enters one of the following timing constants to identify which model computer is being used:

Computer	Timing Constant
2100	169
2114	130
2115	130
2116	148

New Add the following pages as a new section following Section 10.

PREFACE

This manual is a complete software reference source for microprogramming the Hewlett-Packard 2100 Computer. With the information given here, the microprogrammer can expand the already powerful capability of the 2100 by adding custom-tailored instructions to the existing set of microprogrammed operations. This ability to expand the Central Processor Unit, in addition to the extraordinary expansion features of the memory and I/O sections, contributes to the total flexibility and adaptability of the 2100.

It is assumed that the microprogrammer has read the *2100 Computer Microprogramming Guide* (5951-3028) and that he has a copy of the handbook available as a comprehensive reference source. The overview presented in section 1 of this manual is merely meant to supplement the handbook by providing additional emphasis and actual symbolic microinstruction examples.

This manual is divided into ten sections. Section 1 is an overview of HP 2100 microprogramming, sections 2 through 8 describe the HP Microassembler, section 9 describes the HP Micro Debug Editor, and section 10 describes the HP Programmable ROM Writer.

While Hewlett-Packard cannot assume responsibility for the effectiveness of microprograms written by the user, further information and assistance may be obtained by contacting a Hewlett-Packard field office. Sales and Service offices throughout the world are listed in the back of this manual.

CONTENTS

ILLUSTRATIONS

TABLES

TABLES (Continued)

INDEX OF HP 2100 MICRO-ORDERS

Note: In each case, the first page reference is that of the description of the micro-order.

INDEX OF HP 2100 MICRO-ORDERS (Continued)

An HP 2100 computer may include one to four control store modules containing microprograms. These modules are referred to as modules #0, #1, #2, and #3.

Control store module #0 is always present and is used exclusively for the HP 2100 basic instruction set. The other three control store modules are optional. An HP 2100 may include any of the following control store module combinations:

0
0 and 1
0 and 2
0 and 3
0, 1, and 2
0, 1, and 3
0, 2, and 3
0, 1, 2, and 3

The HP floating-point instruction set, if included in the HP 2100, pre-empts module #1. Modules #2 and #3 are available for user-supplied microprograms (as is module #1 if the floating-point instruction set is not expected to be used).

Each control store module contains 400_8 (256_{10}) locations; each location accommodates one microinstruction containing six micro-orders. The locations in module #0 have the octal addresses 000-377; those in module #1 have the octal addresses 400-777; those in module #2 have the octal addresses 1000-1377; and those in module #3 have the octal addresses 1400-1777.

This section has seven parts. The first part summarizes the entities that the microprogrammer may work with; the second describes the format of a microinstruction; the third discusses how to pass control from a

program to a microprogram; the fourth comments on jump table conventions; the fifth describes microprogramming input/output; the sixth describes how to pass data between core memory and control store modules; and the last part describes microprogramming shift operations.

MICROPROGRAMMING FACILITIES

The microprogrammer has the following entities to work with:

Thirteen registers

- A-register (16 bits)
- B-register (16 bits)
- M-register (15 bits)
- T-register (16 bits)
- Q-register (16 bits)
- F-register (16 bits)
- P-register (16 bits)
- Four Scratch Pad Registers (16 bits each)
- Central Interrupt Register (6 bits)
- CPU Instruction Register (16 bits)

The shaded registers are available to the microprogrammer only for a few strictly defined uses. The M- and T-registers are used for accessing core memory locations. The Central Interrupt Register is a read-only register that lets the microprogrammer know which I/O device has caused an interrupt. The CPU Instruction Register is used for performing input/output operations, for performing a special shift operation (shift data item left four bit positions), for calling a microprogram by way of a secondary jump table, and for passing a 4-bit parameter from the calling program to a microprogram.

The other nine registers may be considered as general purpose registers.

A five-bit hardware counter

A function generator

A shifter

Five 16-bit data paths between the registers, the counter, the function generator, the shifter, and the I/O hardware

- R-bus
- S-bus
- ALU-bus
- T-bus
- I/O-bus

Four flip-flops

- Flag (not to be confused with the I/O Flag)
- Overflow
- Extend
- Carry

By examining Figure 1-1, most of the available microprogramming tasks (referred to as micro-orders) are apparent. For example, a micro-order can:

- Read the contents of a register onto a bus.
- Read the contents of a bus into a register.
- Read the contents of the R-bus onto the S-bus.
- Read the contents of the S-bus onto the I/O-bus.

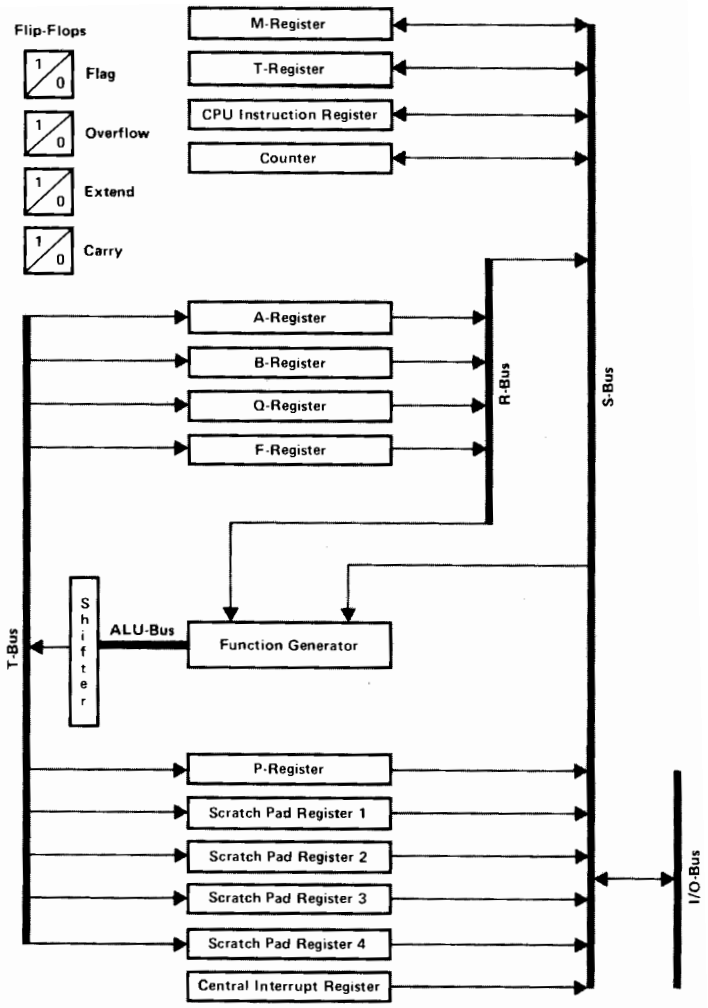


Figure 1-1. Microprogramming Facilities

- Read the contents of the I/O-bus onto the S-bus.
- Cause the function generator and the shifter to perform a function (e.g., add, subtract, logical "inclusive OR", shift left one bit position) using the contents of the R- and S-buses as input. The result is automatically read onto the T-bus.

All the available micro-orders are described in section 4 of this manual.

MICROINSTRUCTION FORMAT

An HP 2100 microinstruction comprises 24 bits and is divided into six fields as shown in figure 1-2.

Bits:	23-21	20-17	16-12	11-8	7-4	3-0
Field:	R-Bus	S-Bus	Function	Store	Special	Skip

Figure 1-2. Microinstruction Format

All micro-orders in a given microinstruction are executed simultaneously. Whenever a data item is read onto a bus, the data item is available on the bus only during execution of that particular microinstruction.

The R-bus field reads the contents of the specified register onto the R-bus.

The S-bus field reads the contents of the specified register onto the S-bus. It is also used for reading a constant or the contents of the R-bus or I/O-bus onto the S-bus.

The Function field causes the function generator and the shifter to perform the specified function using the contents of the R- and S-buses as input. The result is automatically read onto the T-bus. The Function field is also used for jumping and for manipulating the Overflow, Extend, and Flag flip-flops.

The Store field reads the contents of the T-bus into the specified register. It is also used for reading the contents of the S-bus into the M-register, the T-register, the CPU Instruction Register, or onto the I/O-bus.

The Special field is used for diverse purposes. It is used for initiating input/output operations, for accessing core memory locations, for loading the counter from the S-bus, for manipulating the Carry flip-flop, and it is used in shifting operations for specifying which direction the data is to be shifted.

The Skip field is used for skipping a microinstruction. If the condition specified in the Skip field is true, the next sequential microinstruction is skipped. In HP 2100 microprogramming, the term "skip" is used in an unconventional way: if the skip condition is true, the next sequential microinstruction is not actually "jumped over", but is forced to be a NOP. The micro-order EOP (End of Phase) is used in the Skip field to exit from a microprogram. When an EOP is sensed, the exit occurs after the next microinstruction is executed. It should be noted that if an EOP is in a microinstruction that is to be skipped, the EOP *is executed* and the exit occurs after execution of the next sequential microinstruction.

There are two cases where the usual function of the Special and Skip fields is inhibited:

1) If the Function field specifies that a jump be performed, the jump address is supplied in place of the Special and Skip fields. The jump address may be in the form of an asterisk expression or a symbolic address.

```
Examples:  --    --    JMP   --    *+20
           --    --    JMP   --    XYZ
```

2) If the S-bus field specifies that a constant be read onto the S-bus, the constant is supplied in place of the Special and Skip fields.

```
Examples:  --    CL    IOR   --    10
           --    CR    IOR   --    377
```

The constant is always coded in the symbolic microinstruction as an octal number.

CL specifies that the constant be read onto the leftmost eight bits (8-15) of the S-bus; CR specifies that the constant be read onto the rightmost eight bits (0-7) of the S-bus.

The Function field cannot contain a NOP. By convention, if a pseudo-NOP is desired in the Function field (as in the above example), an IOR is used.

ACCESSING A MICROPROGRAM

JUMP TABLES

One control store module must be designated as the entry module. This decision is communicated to the hardware by a hardwired connection on the control store board (A2).

To transfer control from a program to a microprogram, the program executes a macro instruction whose format is 105xxx (octal), where xxx is 000-377. This passes control to one of the first sixteen locations of the entry module. See Table 1-1.

Table 1-1. Effect of the Various 105xxx Macro Instructions

Value of *xxx*	Control Store Location Jumped To		
	Entry Module		
	#1	#2	#3
000-017	400	1000	1400
020-037	401	1001	1401
040-057	402	1002	1402
060-077	403	1003	1403
100-117	404	1004	1404
120-137	405	1005	1405
140-157	406	1006	1406
160-177	407	1007	1407
200-217	410	1010	1410
220-237	411	1011	1411
240-257	412	1012	1412
260-277	413	1013	1413
300-317	414	1014	1414
320-337	415	1015	1415
340-357	416	1016	1416
360-377	417	1017	1417

The first sixteen locations of the entry module are referred to collectively as the primary jump table. Each location in the primary jump table normally contains a jump microinstruction which passes control either to the desired microprogram or to a secondary jump table.

If secondary jump tables are not used, a maximum of 16 microprograms are callable. In this case, the calling program must use one of the macro instructions 105000, 105020, 105040, 105060, 105100, 105120, 105140, 105160, 105200, 105220, 105240, 105260, 105300, 105320, 105340, or 105360.

Note: When secondary jump tables are not used, the only reason for using any of the other 240 macro instructions would be to pass a four-bit parameter to the microprogram. The passing of parameters is discussed later in this section.

However, each microinstruction in the primary jump table may pass control to another jump table (referred to as a secondary jump table). Each secondary jump table may be up to 16 locations long. If every microinstruction in the primary jump table points to a 16-location secondary jump table, the maximum number of callable microprograms increases to 256. The following paragraphs discuss the use of secondary jump tables.

When a 105xxx macro instruction is executed, the instruction itself is in the CPU Instruction Register. Whenever a jump microinstruction is executed, the rightmost four bits (bits 0-3) of the S-bus are automatically "OR"ed with the specified jump address. Usually the S-bus contains all zeros and the specified jump address is not altered. However, microinstructions can read the contents of the CPU Instruction Register onto the S-bus.

The following example demonstrates the use of a secondary jump table. Assume that:

a) module #1 is the entry module

b) the second microinstruction in the primary jump table (control store location 401) passes control to a secondary jump table

c) the secondary jump table resides in control store locations 500 through 517

The micro-coding is as shown in table 1-2.

When the macro instruction 105025 is executed by the calling program, control passes to control store location 401 which, in turn, passes control to control store location 776. The microinstructions at control store locations 776 and 777 cause the rightmost four bits of the

Table 1-2. Secondary Jump Table Usage

Control Store Location	Contents
401	-- -- JMP -- 776
.	
.	
.	
500 through 517	JMP microinstructions
.	
.	
776	-- ADR IOR S1
777	-- S1 JMP -- 500

105025 macro instruction (05, octal) to be "OR"ed with the jump address (500), thus causing a jump to control store location 505. The microinstruction in control store location 505 then passes control to the desired microprogram.

Specifically, the microinstructions shown in control store locations 776 and 777 of the above example work as shown in table 1-3.

The microcoding in the above example may be used for jumping to secondary jump tables that reside in modules #1 or #3 (the only permissible variation being that Scratch Pad Register 3 may be used instead of Scratch Pad Register 1).

If the secondary jump table resides in control modules #0 or #2, the pair of microinstructions shown in control store locations 776 and 777 are combined into one microinstruction, as follows:

Control Store Location	Contents
776	-- ADR JMP -- 500

Table 1-3. Microinstruction Commentary

First microinstruction: • The ADR reads bits 0-9 of the CPU Instruction Register onto the S-bus. • The IOR reads the contents of the S-bus onto the T-bus. • The S1 reads the contents of the T-bus into Scratch Pad Register 1.
Second microinstruction: • The S1 reads the contents of Scratch Pad Register 1 onto the S-bus. • The JMP passes control to the effective jump address. The effective jump address is formed automatically by "OR"ing bits 0-3 of the S-bus with the specified jump address (500).

This difference results from the way the jump address is stored in the microinstruction. Bits 0-7 of the microinstruction specify an address 000-377 while the least significant bit of the S-bus and Function fields together specify what control store module is being jumped to: 00 = module #0, 01 = module #1, 10 = module #2, and 11 = module #3. Figure 1-3 shows how the binary module addresses are stored in the microinstruction.

As long as nothing is coded in the S-bus field, the microassembler automatically sets these two bits to the proper values. However, when the microprogrammer codes something in the S-bus field, he forces the least significant bit of the S-bus field to be set either to a zero or a one. An ADR micro-order sets the bit off (thus specifying control store module #0 or #2) whereas an S1 micro-order sets the bit on (thus specifying control store module #1 or #3).

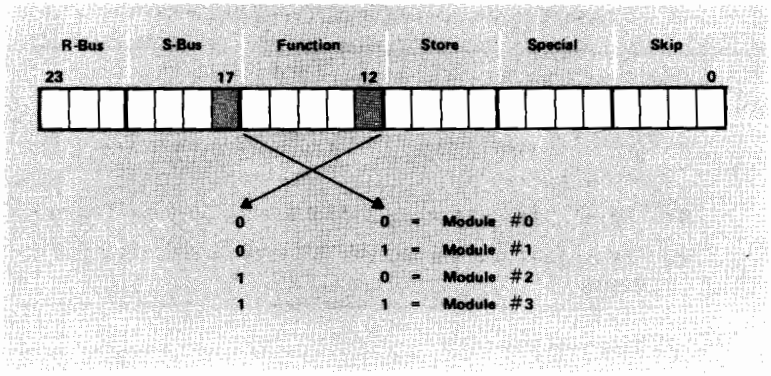


Figure 1-3. Control Store Module Number as Stored in HP 2100 Jump Microinstruction.

PASSING CONTROL FROM AN ASSEMBLY LANGUAGE PROGRAM

There are two ways to pass control from an assembly language program to a microprogram. The first applies only if the RAM (Random Access Memory) psuedo-instruction is available; the second applies in any case. The two methods are as illustrated in table 1-4.

Table 1-4. Passing Control From an Assembly Language Program to a Microprogram

Method 1:		RAM	SWB
		.	
		.	
		.	
	SWB	EQU	xxxB
Method 2:		OCT	105xxx

The RAM pseudo-instruction automatically forms the 105xxx macro instruction using the constant supplied in the EQU statement (the 105xxx macro instruction replaces the RAM pseudo-instruction). In both cases, xxx is 000-377.

PASSING CONTROL FROM A FORTRAN PROGRAM

A FORTRAN program passes control to a microprogram indirectly by way of an assembly language program.

For example, the FORTRAN statement

```
      CALL   XYZ (A,B)
```

generates the following calling sequence:

```
      JSB    XYZ
      DEF    *+3
      DEF    address of the first parameter
      DEF    address of the second parameter
```

When the above calling sequence is executed, control passes to the assembly language program named XYZ. XYZ replaces the JSB XYZ instruction in the above calling sequence with the 105xxx macro instruction and then passes control to the 105xxx macro instruction. The program XYZ is as follows:

```
      ENT    XYZ
XYZ   NOP
      LDA    XYZ
      ADA    =D-1
      LDB    105xxxB
      STB    0,I
      JMP    0,I
```

Notes: When the above calling sequence is executed, the memory address of DEF *+3 is automatically stored in the entry point location (XYZ NOP).

The A-register is referenced as memory location 0.

Specifically, the program XYZ works as follows:

- The LDA instruction loads the memory address of the DEF *+3 instruction into the A-register.
- The ADA instruction subtracts one from the contents of the A-register. The A-register now contains the memory address of the JSB XYZ instruction.
- The LDB instruction loads the 105xxx octal constant into the B-register.
- The STB instruction stores the contents of the B-register in the memory location pointed to by the A-register.
- The JMP instruction passes control to the memory location pointed to by the A-register.

Note that the microprogrammer must be aware of the assembly language calling sequence produced by the FORTRAN or ALGOL compiler in order to properly access the parameters passed by the calling program and to return control to the proper location in the calling sequence (see "Passing Parameters" later in this section).

PASSING CONTROL FROM AN ALGOL PROGRAM

An ALGOL program passes control to a microprogram indirectly by way of an assembly language program. The method is the same as described for FORTRAN programs, above.

PASSING PARAMETERS

ASSEMBLY LANGUAGE PROGRAMS

There are three methods of passing parameters from an assembly language program to a microprogram:

- Use the A- and/or B-registers.
- Use DEF or OCT statements immediately following either the RAM psuedo-instruction or the octal 105xxx macro instruction.
- Use the rightmost four bits of the octal 105xxx macro instruction.

With the first method, the calling program loads the parameters into the A- and/or B-registers (using LDA and/or LDB instructions) just prior to executing the 105xxx macro instruction. The microprogram could then access the parameters directly from the registers. The microcode for accessing the parameters in this manner would be as follows:

```
A   --  IOR  *  --  --
B   --  IOR  *  --  --
```

* = any register

With the second method, the calling program supplies a series of OCT and/or DEF statements immediately following the 105xxx macro instruction. The OCT and DEF statements may either contain the parameters or point to them. When control passes to the microprogram, the P-register contains the address of the first instruction following the 105xxx macro instruction.

If the DEF or OCT statement actually contains the parameter, the microprogram does the following:

- Reads the contents of the P-register into the M-register.
- Executes a read memory (RW) operation.

- Retrieves the parameter from the T-register and reads it into a general purpose register.
- Increments the P-register.

The microcode would be as follows:

```
--   P    IOR   M   RW   --
--   T    IOR   *   --   --
--   P    INC   P   --   --
```

* = any register

If more than one parameter is being passed, the microprogram executes the above microcode once for each parameter, as needed.

If the DEF or OCT statement contains the address of the parameter, the microprogram does the following:

- Reads the contents of the P-register into the M-register.
- Executes a read memory (RW) operation.
- Retrieves the parameter address from the T-register and reads it into the M-register.
- Executes another read memory (RW) operation.
- Retrieves the parameter from the T-register and reads it into a general purpose register.
- Increments the P-register.

The microcode would be as follows:

```
--  P   IOR  M   RW  --
--  T   IOR  S1  --  --
--  S1  IOR  M   RW  --
--  T   IOR  *   --  --
--  P   INC  P   --  --
```

* = any register

Again, if more than one parameter is being passed, the microprogram executes the above microcode once for each parameter.

With the third method, the microprogram uses the ADR micro-order to read bits 0-9 of the CPU Instruction Register onto the S-bus and then reads the bits into a general purpose register. The microcode would be as follows:

```
--  ADR  IOR  *  --  --
```

* = any register

The three methods described above may be used in any combination.

FORTRAN PROGRAMS

A FORTRAN program passes parameters to a microprogram by supplying them in parentheses in the CALL to the assembly language linkage program, as follows:

```
CALL  XYZ (15,100,500,7)
```

where XYZ is the entry point of the assembly language linkage program; and

15, 100, 500, and 7 are the actual parameters being passed.

After the assembly language linkage program has performed its function, the following calling sequence is executed:

```
OCT   105xxx
DEF   *+5 (this is the return address)
DEF   address of the first parameter
DEF   address of the second parameter
DEF   address of the third parameter
DEF   address of the fourth parameter
```

When the microprogram receives control, the P-register is pointing to the instruction immediately following the octal 105xxx macro instruction (i.e., it is pointing to the return address). To access the parameters, the microprogram does the following:

- Increments the P-register.
- Reads the contents of the P-register into the M-register.
- Executes a read memory (RW) operation.
- Retrieves the parameter address from the T-register and reads it into the M-register.
- Executes another read memory (RW) operation.
- Retrieves the parameter from the T-register and reads it into a general purpose register.

The microcode would be as follows:

```
        --  P   INC   P   --   --
        --  --  JSB   --  GETAD
        --  S1  IOR   *   --   --
                  .
                  .
                  .
GETAD   --  P   IOR   M   RW   --
GETAX   --  T   IOR   S1  --   NEG
        --  --  RSB   --  --   --
        --  S1  IOR   M   RW   --
        --  --  JMP   --  GETAX
            * = any register
```

If more than one parameter is being passed, the microprogram executes the above microcode once for each parameter. The GETAD routine handles multiple levels of indirect addressing. After accessing the final parameter, however, the microprogram must increment the P-register one more time so it is pointing to the first instruction following the calling sequence.

ALGOL PROGRAMS

The passing of parameters from an ALGOL program to a microprogram involves the same technique described for FORTRAN programs, above.

JUMP TABLE CONVENTIONS

The jump table conventions are described on pages 3-6 through 3-8 of the *2100 Computer Microprogramming Guide* (5951-3028).

Note that these conventions in effect divide the primary jump table among three control store modules (i.e., the first six locations reside in module #1, the next five locations effectively reside in module #2, and the last five locations effectively reside in module #3).

It is recommended, though not required, that the microprogrammer adhere to these conventions.

In actual fact, the following generalizations apply:

- Any module (#1, #2, or #3) may be the entry module.
- Primary jump table entries may point backward or forward, and may point to any location in modules #1, #2, or #3.
- Any primary jump table location may point to a secondary jump table.

The following restrictions apply if the HP floating-point instruction set is present:

- The HP floating-point instruction set must reside in module #1.
- The microprogrammer is restricted to the use of macro instructions 105140 through 105377. These map into modules #2 and #3 as shown on page 3-7 of the *2100 Computer Microprogramming Guide.*

If HP options other than the floating-point instruction set are present, similar restrictions apply.

INPUT/OUTPUT

This section discusses how to pass data during an input/output operation. Microprogrammed I/O operations that use the interrupt system also require that certain control instructions such as STC xx,C and CLC xx be executed. This is done by loading the octal representation of the particular instruction into the CPU Instruction Register and then executing an IOG1 micro-order. See the teleprinter output driver example in section 6 ("Sample Microprograms") of this manual.

INPUT

An input operation transfers one character between an input device and a register. The micro-orders for performing an input operation are as shown in table 1-5.

Table 1-5. Input Micro-Orders

--	--	--	--	IOG1	--
--	--	--	--	--	--
--	IOI	IOR	--	--	--
--	IOI	IOR	*	--	--

* = M, T, A, B, Q, F, P, S1, S2, S3, or S4

Before executing the above micro-orders, however, the micro-programmer must place the octal representation of an input instruction (LIA, LIB, MIA, or MIB) in the CPU Instruction Register. 1025xx is an LIA instruction, 1065xx is an LIB instruction, 1024xx is an MIA instruction, and 1064xx is an MIB instruction (xx is the select code of the desired input device).

The IOG1 causes the hardware to decode and execute the input instruction. This results in one character being transmitted from the input device to the I/O-bus. The IOI reads the character from the I/O-bus onto the S-bus (this is done twice in order to compensate for the possibility of noise occurring during the "I/O-bus to S-bus" data transfer). The IOR in the last microinstruction reads the character from the S-bus onto the T-bus; the Store field reads the character into the specified register.

The micro-orders must be coded into four consecutive micro-instructions and must be in the relative positions shown above. The fields containing "--" are available for other tasks.

OUTPUT

An output operation transfers one character between a register and an output device. The micro-orders for performing an output operation are as shown in table 1-6.

Table 1-6. Output Micro-Orders

--	--	--	--	IOG1	--
*	**	IOR	--	--	--
*	**	IOR	IOO	--	--
*	**	IOR	IOO	--	--

* = NOP or an R-bus register mnemonic

** = RRS or an S-bus register mnemonic (RRS reads the contents of the R-bus onto the S-bus)

Before executing the above micro-orders, however, the micro-programmer must load the octal representation of an output instruction (OTA or OTB) into the CPU Instruction Register. 1026xx is an OTA instruction and 1066xx is an OTB instruction (xx is the select code of the desired output device).

The IOG1 causes the hardware to decode and execute the output instruction. This results in one character being transmitted from the I/O-bus to the output device. The S-bus field of the last three micro-instructions reads the character onto the S-bus. The IOO in the last two microinstructions reads the character from the S-bus onto the I/O-bus. The repetition of the R-bus, S-bus, and Store field mnemonics (*, **, and IOO) is necessary in order to compensate for the possibility of noise occurring during the "S-bus to I/O-bus" data transfer. The IOR in the last three microinstructions is a "pseudo-NOP".

The micro-orders must be coded into four consecutive microinstructions and must be in the relative positions shown above. The fields containing "--" are available for other tasks.

ACCESSING CORE MEMORY LOCATIONS

By placing a core memory address in the M-register and then executing an RW or CW micro-order, the microprogrammer can read data from a core memory location or write data into a core memory location. The T-register is always used for passing data between core memory and a control store module.

READ FROM MEMORY

To read data from a core memory location, the microprogrammer first loads the core memory address into the M-register by using either of the following microinstructions:

```
*    RRS    IOR    M    RW    -
     (or)
-    **     IOR    M    RW    -
```

where * is any R-bus register.
** is any S-bus register.

The specified register must contain the core memory address. The RW micro-order initiates the "read from memory" operation.

The microprogrammer then retrieves the data from the T-register by using the following microinstruction:

```
-    T    IOR    *    -    -
```

where * is the register into which the data item is to be stored.

For example, to read the contents of core memory location 300_8 into the B-register, the microprogrammer could use the following microinstructions:

```
-    CR    IOR    S1    300
-    S1    IOR    M     RW    -
-    T     IOR    B     -     -
```

WRITE INTO MEMORY

To write data into a core memory location, the microprogrammer first loads the core memory address into the M-register by using either of the following microinstructions:

```
*    RRS   IOR    M     CW    ***
     (or)
-    **    IOR    M     CW    ***
```

where * is any R-bus register.
** is any S-bus register.
*** is a "skip" micro-order (usually UNC or NMPV).

The specified register must contain the core memory address. The CW micro-order initiates the "write into memory" operation. In order for the operation to be performed, *the next sequential microinstruction must be skipped.*

The microprogrammer then loads the data into the T-register by using either of following microinstructions:

```
*    RRS   IOR    T     -     -
     (or)
-    **    IOR    T     -     -
```

where * is any R-bus register.
** is any S-bus register.

For example, to write a data word from the B-register into core memory location 100_8, the microprogrammer could use the following microinstructions:

```
-   CR    IOR   S1   100
-   S1    IOR   M    CW    UNC
-   -     IOR   -    -     -
B   RRS   IOR   T    -     -
```

An NMPV micro-order is used for testing whether or not the specified core memory address points to a location in the protected area of core memory. The above example is again shown, only this time using NMPV.

```
-   CR    IOR   S1   100
F   S1    DEC   M    CW    NMPV
-   -     JMP   -    ERROR
B   RRS   IOR   T    -     -
```

If a memory protect violation is detected, the "write into memory" operation is *not* performed and control passes to ERROR. If no memory protect violation is detected, the JMP microinstruction is skipped and the "write into memory" operation is performed.

MICROPROGRAMMING SHIFT OPERATIONS

The microprogrammer can perform a variety of shift operations. In the following paragraphs, the shift operations are categorized according to the size of the data item being shifted.

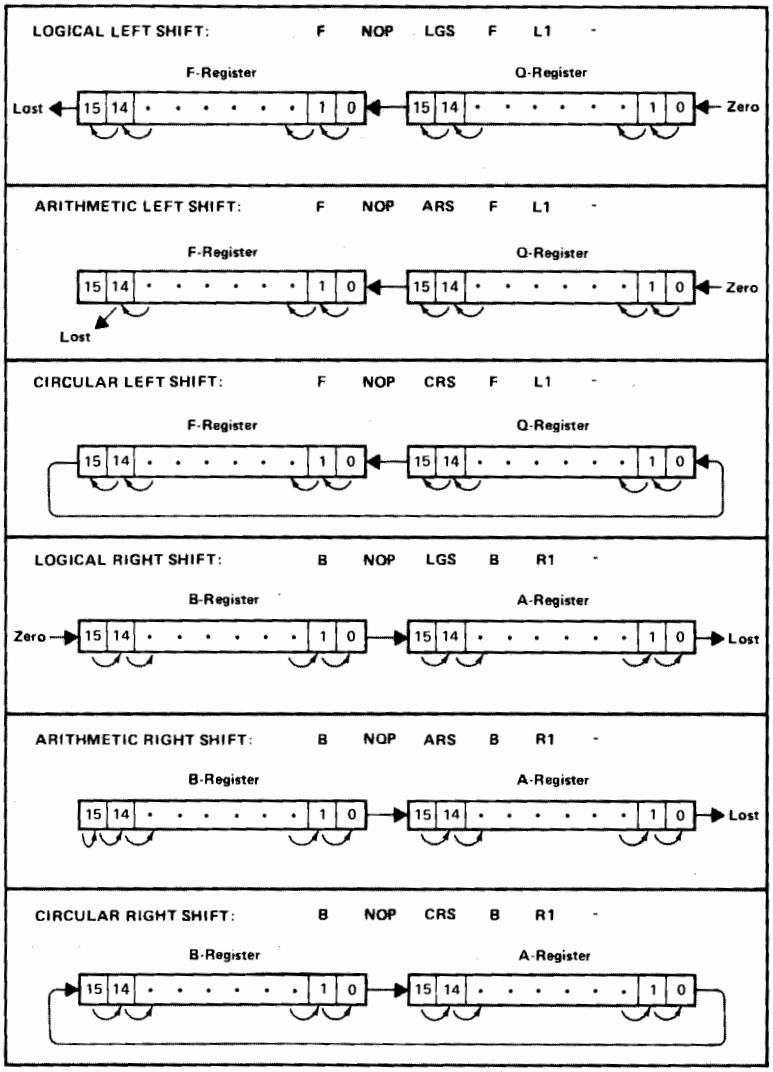


Figure 1-4. 32-bit Data Item Shifts

32-BIT DATA ITEMS

The data item must be in the B- and A-registers (for right shifts) or the F- and Q-registers (for left shifts).

For right shifts, the B-register contains the high-order 16 bits and the A-register contains the low-order 16 bits. For left shifts, the F-register contains the high-order 16 bits and the Q-register contains the low-order 16 bits.

The various 32-bit data item shift operations are shown in figure 1-4.

17-BIT DATA ITEMS

The LWF micro-order allows the microprogrammer to shift the Flag flip-flop in conjunction with the contents of any register.

The 17-bit data item shift operations are shown in figure 1-5.

16-BIT DATA ITEMS

The data item may be in any register. There are two types of 16-bit data item shift operations: a logical shift and a circular shift. The logical shifts are shown in figure 1-6.

The circular shift operation results in the data item being rotated four bit positions to the left. This is accomplished by using the Shift-Rotate Group (SRG) instruction decoders.

The microprogrammer must first load the constant 000027_8 into the CPU Instruction Register. This is done using the following microinstruction:

```
NOP    CR     IOR    IR     27
```

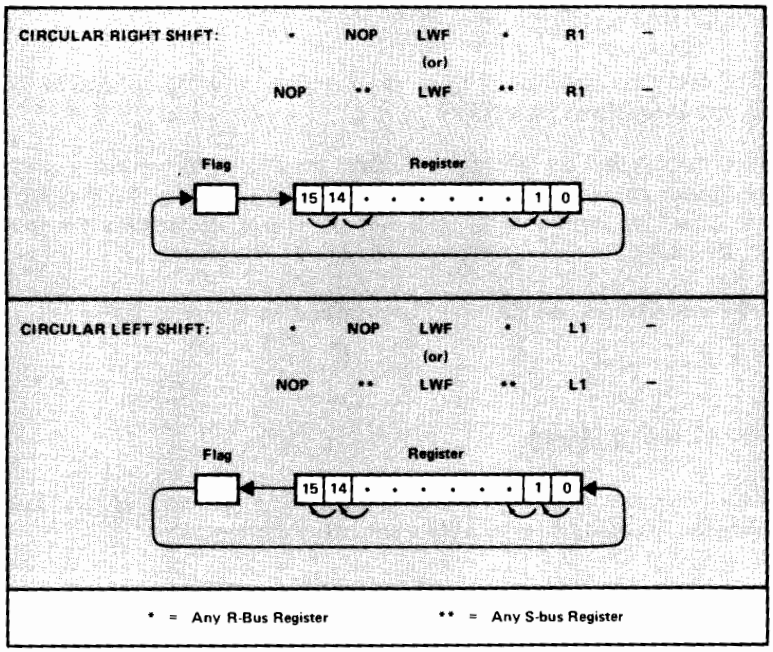


Figure 1-5. 17-bit Data Item Shifts

Each time the data item is to be rotated four bit positions to the left, the microprogrammer executes either of the following:

```
*       NOP     IOR     *       SRG2    -
      (or)
NOP     **      IOR     **      SRG2    -
```

where * is any R-bus register and ** is any S-bus register. The only restriction is that the same Scratch Pad Register cannot be specified in both the S-bus and Store fields of the same microinstruction.

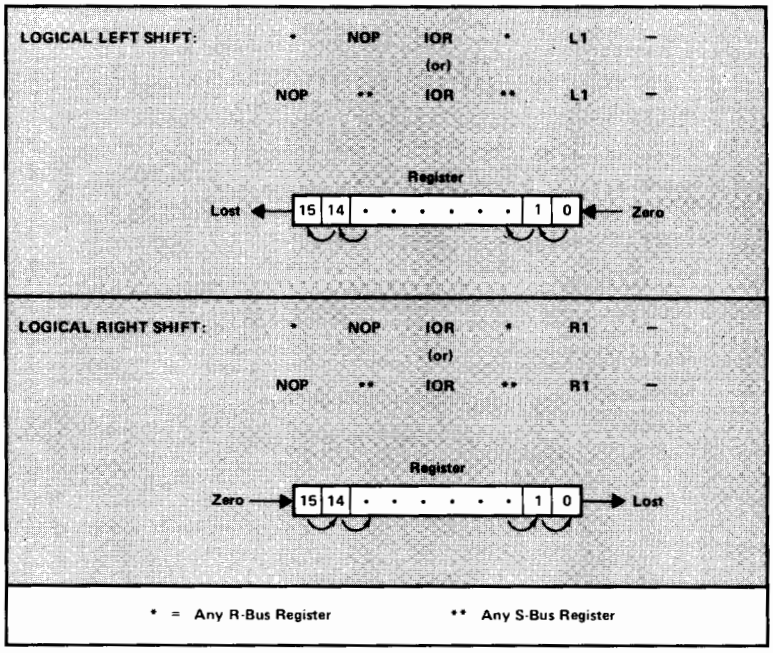


Figure 1-6. 16-bit Data Item Logical Shifts

For example, to rotate the contents of the B-register eight positions to the left, the microprogrammer would use the following micro-instructions:

```
NOP     CR      IOR     IR      27
B       NOP     IOR     B       SRG2    –
B       NOP     IOR     B       SRG2    –
```

SECTION 2
GENERAL DESCRIPTION OF THE HP MICROASSEMBLER

The HP Microassembler translates symbolic source language micro-instructions into a machine language object microprogram. Source input is read from either punched cards or paper tape; the object program is punched on paper tape in a format acceptable to the HP Micro Debug Editor. The source language provides:

- Alphanumeric mnemonics for each micro-order.
- Symbolic addressing capability.
- A set of assembler control statements for controlling the assembly process.

The HP Microassembler is designed to run under the Basic Control System (BCS) and requires a minimum of 8K of memory. In addition, it requires a system console device (teleprinter or CRT terminal), a line printer, a paper tape photoreader, a paper tape punch, and a card reader.

In an 8K BCS environment, neither the magnetic tape driver nor the buffered version of IOC should be used. In addition, instead of actually loading the assembler and relocatable library into memory, the user must produce an absolute punched tape and then load the absolute tape into memory using the Basic Binary Loader. The operating instructions for producing the absolute tape are presented in the *Software Operating Procedures* manual (5951-1369). In this regard, it should be noted that the assembler is loaded at step 5 and the BCS relocatable library is loaded at step 7. The operating instructions for loading the absolute tape into memory using the Basic Binary Loader are presented on pages 2100-10 and 2100-11 of the *Software Operating Procedures* manual.

THE ASSEMBLY PROCESS

The assembling of a source microprogram into an object microprogram is a two-pass operation. A *pass* is defined as one processing cycle of the source input.

In the first pass, the microassembler reads the entire source microprogram and creates a symbol table (discussed later in this section) based upon the statement labels that are used. In addition, it checks for duplicate labels and, if necessary, generates appropriate error messages.

In the second pass, the microassembler again reads the entire source microprogram and, using the symbol table, resolves all references to symbolic addresses. In addition, it checks for more errors and, if necessary, generates appropriate error messages. It is during Pass 2 that the object program is created. At the end of Pass 2, the assembly listing is printed and (if no fatal errors were detected) the object microprogram is punched.

There are two types of error messages: *warning* and *fatal.* Warning messages are merely informational, drawing the microprogrammer's attention to questionable, but not always illegal, microprogramming usage. Warning messages do not curtail the assembly process. Fatal errors, on the other hand, prevent the object program from being punched. All warning and fatal error messages are presented in section 8, "Error Messages", of this manual.

The assembly listing contains a copy of the symbol table, a copy of the source language microprogram, plus any error messages. To facilitate debugging, each error message immediately preceeds the offending source statement. The assembly listing is discussed in greater detail later in this section.

Usually the microassembler halts at the end of Pass 1 to allow the operator to reload the source input in the input device. However, if a magnetic tape drive is available, the microprogrammer may use an assembler control statement ($PASS2) to cause the input to Pass 1 to be copied to magnetic tape for use as input to Pass 2.

PROGRAM LOCATION COUNTER

The microassembler maintains a counter, called the *program location counter*, that is used for assigning absolute control store addresses to successive microinstructions. By using an assembler control statement ($ORIGIN), the microprogrammer may reset this counter to any desired value. $ORIGIN statements may appear anywhere within a source language microprogram. If no $ORIGIN statements are used, the program location counter is originally set to 400_8 and is incremented by one for each successive microinstruction.

SYMBOLIC ADDRESSING

Each source language microinstruction may include an alphanumeric *statement label.* The statement label, if present, is the microinstruction's *symbolic address.* Symbolic addresses may be used as jump addresses in JMP, JSB, and CJMP microinstructions.

During Pass 1 the microassembler compiles a table, called the *symbol table,* containing all statement labels used in the microprogram. With each symbol, the microassembler also records the absolute control store address assigned to the associated microinstruction. In addition, the symbol table contains all external symbols that are declared in an $EXTERNALS assembler control statement.

Whenever it encounters a symbol as the jump address in a jump microinstruction, the microassembler consults the symbol table and replaces the symbolic jump address with the appropriate absolute control store address.

There are three rules pertaining to the use of symbolic addresses (violation of any constitutes a fatal error):

1) Two microinstructions may not have the same statement label.

2) A microinstruction may not have a statement label identical to a declared external symbol.

3) Symbols used as jump addresses must be defined somewhere in the microprogram.

A symbol is defined if it is used as a statement label or if it appears in an $EXTERNALS assembler control statement.

ASTERISK (*) AS AN ADDRESS

The microprogrammer may use an asterisk expression as a jump address in JMP, JSB, or CJMP microinstructions. When used in this manner, the asterisk means "the address of the present microinstruction". Thus, the microinstruction:

```
-   -   JMP   -   *+10
```

causes control to pass to the tenth microinstruction following the JMP *+10 microinstruction. Similarly, the microinstruction:

```
-   -   JMP   -   *-6
```

causes control to pass to the sixth microinstruction preceding the JMP *-6 microinstruction.

ASSEMBLY OPTIONS

Through the use of assembler control statements, the microprogrammer can do the following (the statement mnemonic is shown in parentheses):

- Specify what device is to be used for reading the source input ($INPUT).
- Specify what device is to be used for punching the object program ($OUTPUT).
- Specify what device is to be used for printing the assembly listing ($LIST).
- Cause the input to Pass 1 to be copied to magnetic tape for use as input to Pass 2 ($PASS2).
- Suppress all warning messages ($SUPPRESS).
- Reset the program location counter ($ORIGIN).
- Define external symbolic addresses ($EXTERNALS).
- Specify that the debug option is to be used ($DEBUG). The debug option affects the mode of operation of the HP Micro Debug Editor. See section 9 of this manual.

The assembler control statements are described in section 5 of this manual.

ASSEMBLER OUTPUT

The microassembler produces a printed listing and a punched paper tape. The punched tape contains the object microprogram in a format acceptable to the HP Micro Debug Editor. The format is illustrated in Figure 2-2.

The assembly listing is in two parts: a symbol table listing and a source microprogram listing (error messages, if present, are interspersed among

the source statements). Figure 2-3 shows a symbol table listing while figures 2-4 and 2-5 show the first and last pages, respectively, of a source microprogram listing. All three figures are extracted from the same assembly listing.

SYMBOL TABLE LISTING

External symbols are listed first. They are in the order in which they were defined in the $EXTERNALS statements. In the symbol table listing, an external symbol is easily identifiable by the "X" immediately following the associated absolute control store address.

The symbols that appear as statement labels within the source microprogram are listed next. Note that they are listed in ascending order by absolute control store address.

Specifically, the format of a symbol table listing is as shown in table 2-1.

Table 2-1. Symbol Table Listing Format

Print Positions	Contents
1–5	Symbol
9–14	Absolute Control Store Address
15	X (if external symbol) blank (if internal statement label)

SOURCE MICROPROGRAM LISTING

Every source statement in the microprogram is assigned a decimal line number. These line numbers appear in print positions 1 through 3 of each line in the listing.

Assembler control statements and comments statements are printed, starting in print position 4, exactly as they appear in the source input.

For microinstruction statements, however, two additional fields are displayed:

- the absolute control store address assigned to the microinstruction
- the machine language object code for the microinstruction

The control store address appears in print positions 6 through 9. The octal representation of the machine language object code appears in print positions 11 through 20.

The object code is interpreted as follows:

- the leftmost three octal digits represent bits 16 through 23 of the machine language microinstruction
- the rightmost six octal digits represent bits 0 through 15 of the machine language microinstruction

This is best illustrated by example. The object code 375 017533 represents the bit pattern shown in figure 2-1.

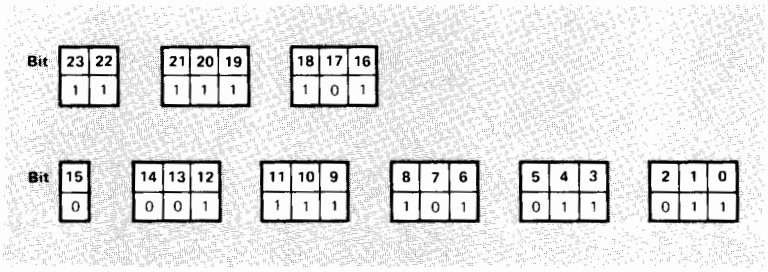


Figure 2-1. Object Code Illustration

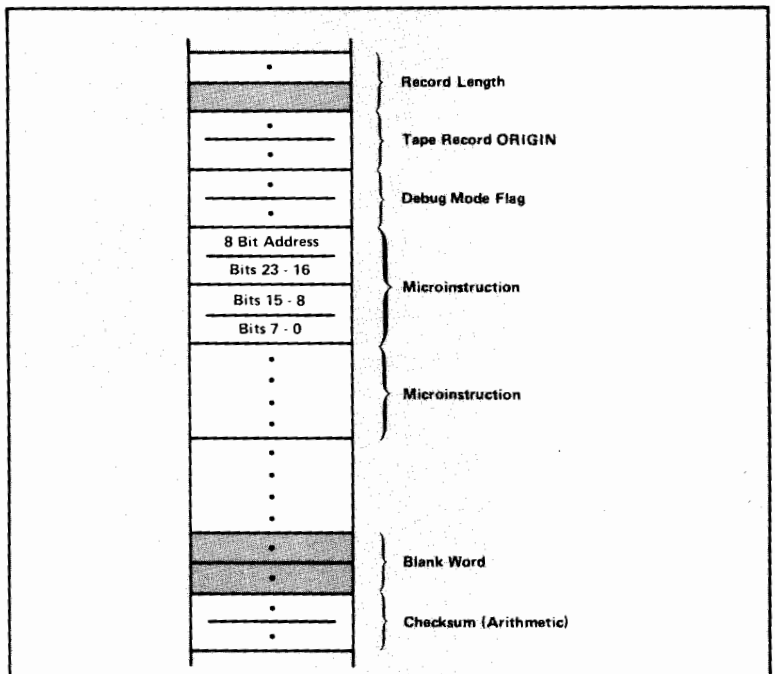
The source language microinstruction is then printed, starting in print position 24, exactly as it appears in the source input. Note that if a teleprinter or an 80-column line printer is used for printing the assembly listing, the source statements are truncated after columns 48 and 56, respectively.

The final line of the source microprogram listing tells the program length and the total number of messages in the listing. Note that the length is specified in octal and it refers to the number of control store locations that the object program requires (maximum allowable = 400_8 per module).

Warning and fatal error messages immediately precede the offending source statement. The messages are in the following form:

WARNING xx IN LINE yy
ERROR xx IN LINE yy

where xx is the message number (see section 8 of this manual) and yy is the line number of the offending statement.



Notes: The record length, tape record origin, debug mode flag, blank word, and checksum each consist of one computer word (two tape characters).

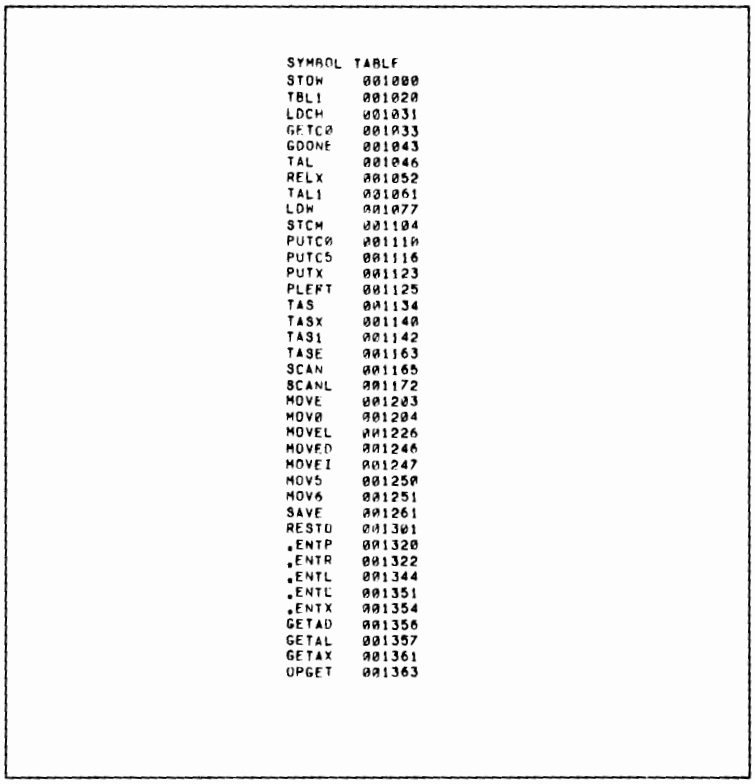
Each microinstruction consists of two computer words (four tape characters).

A tape record may contain a maximum of 27 microinstructions.

Whenever a new origin is declared (via an $ORIGIN assembler control statement), a new tape record is begun.

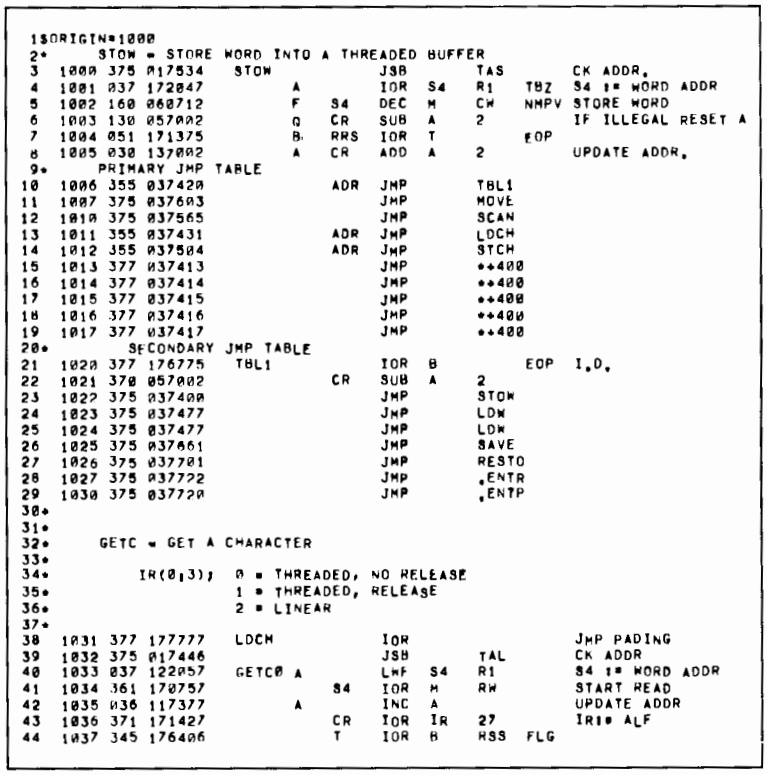
The tape record length can be from 7 to 59 computer words.

Figure 2-2. Object Microprogram Tape Format



```
SYMBOL TABLE
STOH    001000
TBL1    001020
LDCH    001031
GETC0   001033
GDONE   001043
TAL     001046
RELX    001052
TAL1    001061
LDH     001077
STCH    001104
PUTC0   001110
PUTC5   001116
PUTX    001123
PLEFT   001125
TAS     001134
TASX    001140
TAS1    001142
TASE    001163
SCAN    001165
SCANL   001172
MOVE    001203
MOV0    001204
MOVEL   001226
MOVED   001246
MOVEI   001247
MOV5    001250
MOV6    001251
SAVE    001261
RESTO   001301
.ENTP   001320
.ENTR   001322
.ENTL   001344
.ENTC   001351
.ENTX   001354
GETAD   001356
GETAL   001357
GETAX   001361
OPGET   001363
```

Figure 2-3. Symbol Table Listing



```
1$ORIGIN=1000
2*      STOW = STORE WORD INTO A THREADED BUFFER
3   1000 375 017534   STOW            JSB         TAS         CK ADDR,
4   1001 037 172047          A        IOR   S4    R1    TBZ   S4 := WORD ADDR
5   1002 160 060712          F   S4   DEC   M     CW    NMPV  STORE WORD
6   1003 130 057002          Q   CR   SUB   A     2           IF ILLEGAL RESET A
7   1004 051 171375          B   RRS  IOR   T           EOP
8   1005 030 137002          A   CR   ADD   A     2           UPDATE ADDR,
9*      PRIMARY JMP TABLE
10  1006 355 037420              ADR  JMP         TBL1
11  1007 375 037603                   JMP         MOVE
12  1010 375 037565                   JMP         SCAN
13  1011 355 037431              ADR  JMP         LDCH
14  1012 355 037504              ADR  JMP         STCH
15  1013 377 037413                   JMP         **400
16  1014 377 037414                   JMP         **400
17  1015 377 037415                   JMP         **400
18  1016 377 037416                   JMP         **400
19  1017 377 037417                   JMP         **400
20*         SECONDARY JMP TABLE
21  1020 377 176775   TBL1            IOR   B           EOP   I,D,
22  1021 370 057002              CR   SUB   A     2
23  1022 375 037400                   JMP         STOW
24  1023 375 037477                   JMP         LDW
25  1024 375 037477                   JMP         LDW
26  1025 375 037661                   JMP         SAVE
27  1026 375 037701                   JMP         RESTO
28  1027 375 037722                   JMP         .ENTR
29  1030 375 037720                   JMP         .ENTP
30*
31*
32*     GETC = GET A CHARACTER
33*
34*         IR(0,3):  0 = THREADED, NO RELEASE
35*                   1 = THREADED, RELEASE
36*                   2 = LINEAR
37*
38  1031 377 177777   LDCH            IOR                     JMP PADING
39  1032 375 017446                   JSB         TAL         CK ADDR
40  1033 037 122057   GETC0 A         LMF   S4    R1          S4 := WORD ADDR
41  1034 361 170757              S4   IOR   M     RW          START READ
42  1035 036 117377          A        INC   A                 UPDATE ADDR
43  1036 371 171427              CR   IOR   IR    27          IR!= ALF
44  1037 345 176406              T    IOR   B     RSS   FLG
```

Figure 2-4. Source Microprogram Listing (First Page)

```
319  1336 026 055377          A   S1  SUB  Q                 Q:= -LENG ASKED -1
320  1337 122 137403          Q   S3  ADD      RSS  NEG     ASKED IS SMALLER
321  1340 362 155377              S3  NOR  Q                    OR ALLOWED IS!
322  1341 361 176777              S4  IOR  B                CK FOR ZERO
323  1342 367 172777              S1  IOR  S3
324  1343 035 034351          A       JMP  P    ,ENTC
325*
326  1344 375 017756  ,ENTL           JSB       GETAD       S1:=NEXT PARM
327  1345 160 060712          F   S4  DEC  M    CW   NMPV   STORE NEXT PARM
328  1346 375 037754                  JMP       ,ENTX           (MEM VIOLATION)
329  1347 367 171377              S1  IOR  T
330  1350 360 116777              S4  INC  B                INC OUT PTR,STO TEM
331  1351 374 114377  ,ENTC       P   INC  P                INC IN PTR
332  1352 136 115367          Q       INC  Q         TBZ    DONE?
333  1353 075 032344          B       JMP  S4   ,ENTL       NO,RESET OUTPTR, GO
334  1354 365 174375  ,ENTX       S2  IOR  P         EOP    EXIT, RESTORE P
335  1355 363 177377              S3  IOR  A
336*
337*      GETAD  AND OPGET
338  1356 375 170757  GETAD       P   IOR  M    RW          GET ADDR
339  1357 345 173763  GETAL       T   IOR  S1        NEG    INDIRECT?
340  1360 377 057777              RSB                       NO, EXIT
341  1361 367 170757  GETAX       S1  IOR  M    RW          YES, READ AGAIN
342  1362 375 037757                  JMP       GETAL
343*
344  1363 367 170757  OPGET       S1  IOR  M    RW          GET  PARAM
345  1364 345 053377              T   RSB  S2
346*
347*       END
348$END
**NO ERRORS**
```

Figure 2-5. Source Microprogram Listing (Last Page)

Source microprograms must be coded using the symbolic statement format described in this chapter.

Each symbolic statement is 80 character positions long and contains the fields shown in table 3-1.

Table 3-1. Symbolic Microinstruction Format

Field	Character Positions
Label	1–5
R-bus	7–9
S-bus	11–14
Function	16–19
Store	21–24
Special	26–29
Skip	31–34
Comments	36–80

Figure 3-1 illustrates the symbolic microinstruction format on an 80-column punched card.

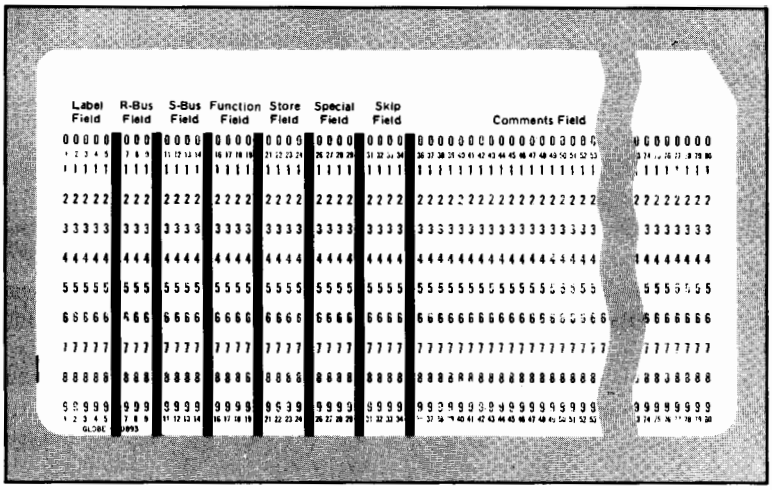


Figure 3-1. Symbolic Microinstruction Format

SYMBOLIC STATEMENT FIELDS

LABEL FIELD

This field is used for assigning a symbolic address to a microinstruction. Statement labels are used as jump addresses in JMP, JSB, and CJMP microinstructions. A valid statement label consists of 1 to 5 alphanumeric characters, of which the first character is not a dollar sign ($) or an asterisk (*). Statement labels must begin in character position 1 and may not contain embedded blanks. The Label field may, of course, be entirely blank (i.e., a microinstruction may be unlabeled).

An asterisk in character position 1 specifies that the remaining 79 character positions contain an alphanumeric comment. Such statements

appear in the source microprogram listing but are otherwise ignored by the microassembler.

A dollar sign in character position 1 specifies that the source statement is an assembler control statement. See section 5 of this manual.

R-BUS FIELD

This field corresponds to the R-bus field of an HP 2100 microinstruction. The purpose of the R-bus field is summarized in section 1 of this manual.

The R-bus field may be entirely blank or may contain any of micro-order mnemonics shown in column 1 of table 3-2. The effect of each mnemonic is described in section 4 of this manual. If a mnemonic is used, it must begin in character position 7. If the R-bus field is entirely blank, the microassembler automatically supplies a NOP.

S-BUS FIELD

This field corresponds to the S-bus field of an HP 2100 microinstruction. The purpose of the S-bus field is summarized in section 1 of this manual.

The S-bus field may be entirely blank or may contain any of the micro-order mnemonics shown in column 2 of table 3-2. The effect of each mnemonic is described in section 4 of this manual. If a mnemonic is used, it must begin in character position 11. If the S-bus field is entirely blank, the microassembler automatically supplies a NOP.

FUNCTION FIELD

This field corresponds to the Function field of an HP 2100 microinstruction. The purpose of the Function field is summarized in section 1 of this manual.

The Function field may be entirely blank or may contain any of the micro-order mnemonics shown in column 3 of table 3-2. The effect of each mnemonic is described in section 4 of this manual. If a mnemonic is used, it must begin in character position 16. If the Function field is entirely blank, the microassembler automatically supplies an IOR.

STORE FIELD

This field corresponds to the Store field of an HP 2100 microinstruction. The purpose of the Store field is summarized in section 1 of this manual.

The Store field may be entirely blank or may contain any of the micro-order mnemonics shown in column 4 of table 3-2. The effect of each mnemonic is described in section 4 of this manual. If a mnemonic is used, it must begin in character position 21. If the Store field is entirely blank, the microassembler automatically supplies a NOP.

SPECIAL FIELD

This field corresponds to the Special field of an HP 2100 microinstruction. The purpose of the Special field is summarized in section 1 of this manual.

The Special field may be entirely blank or may contain any of the micro-order mnemonics shown in column 5 of table 3-2. The effect of each mnemonic is described in section 4 of this manual. If a mnemonic is used, it must begin in character position 26. If the Special field is entirely blank, the microassembler automatically supplies a NOP.

SKIP FIELD

This field corresponds to the Skip field of an HP 2100 microinstruction. The purpose of the Skip field is summarized in section 1 of this manual.

The Skip field may be entirely blank or may contain any of the micro-order mnemonics shown in column 6 of table 3-2. The effect of each mnemonic is described in section 4 of this manual. If a mnemonic is used, it must begin in character position 31. If the Skip field is entirely blank, the microassembler automatically supplies a NOP.

Table 3-2. Valid Mnemonics

R-bus	S-bus	Function	Store	Special	Skip
NOP	NOP	IOR	NOP	NOP	NOP
A	P	XOR	M	CNTR	EOP
B	S1	AND	T	CW	COUT
Q	S2	NOR	IR	ECYN	CTR
F	S3	LWF	A	ECYZ	CTRI
AAB	S4	ARS	B	IOG1	FLG
CAB	M	LGS	Q	L1	ICTR
CQ	T	CRS	F	R1	NEG
	ADR	JMP	P	RSS	NMPV
	CNTR	CJMP	S1	RW	ODD
	CL	JSB	S2	AAB	OVF
	CR	RSB	S3	ASG1	RPT
	CIR	ADD	S4	ASG2	TBZ
	IOI	ADDO	IOO	LEP	UNC
	RRS	SUB	AAB	SRG1	AAB
	COND	MPY	CAB	SRG2	NAAB
		DIV			
		DEC			
		INC			
		INCO			
		SOV			
		CLO			
		SFLG			
		CFLG			
		RFE			
		RFI			
		P1A			

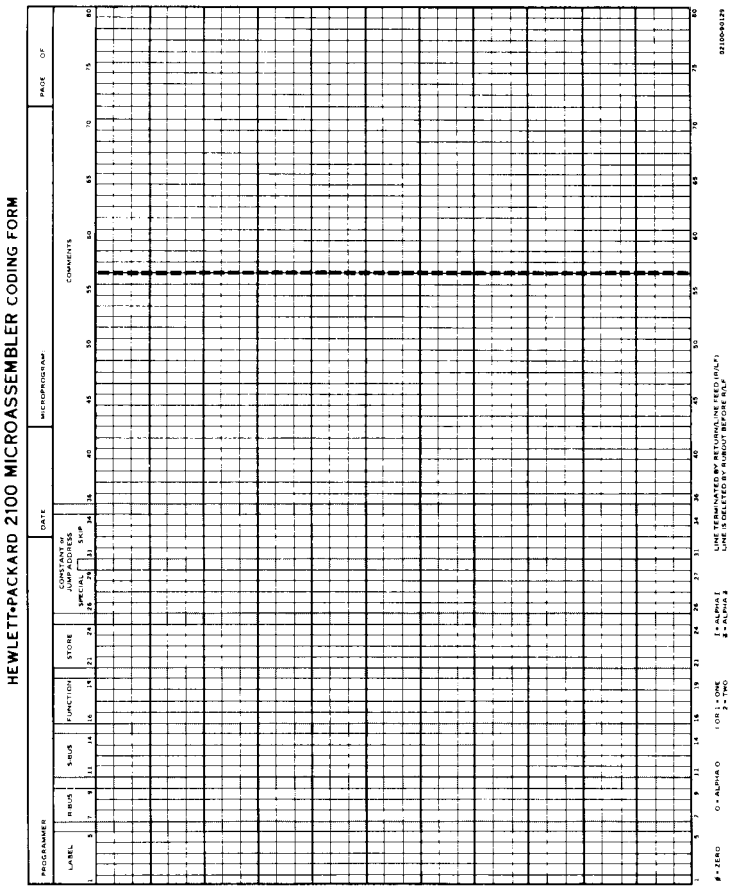


Figure 3-2. Standard Coding Form

COMMENTS FIELD

This field may be freely used by the microprogrammer to introduce alphanumeric comments into the assembly listing. Other than this, the Comments field is ignored by the microassembler.

STANDARD CODING FORM

Hewlett-Packard provides a standard form to facilitate the coding of source microprograms. This form is illustrated in figure 3-2.

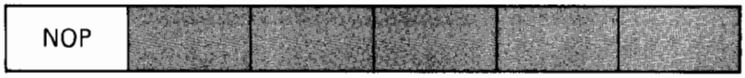
This section describes what each micro-order does. It is assumed that the microprogrammer has read both the *2100 Computer Microprogramming Guide* and the overview presented in section 1 of this manual. A few of the descriptions (e.g., MPY, DIV, etc.) refer the reader to the *2100 Computer Microprogramming Guide.*

To facilitate learning, the more esoteric information is shaded. The reader should concentrate first on the unshaded material. The shaded information pertains mainly to, but is not limited to, module #0 programming. In conjunction with such descriptions, the reader should study section 7, "Miscellany", of this manual.

R-BUS FIELD

The following micro-order mnemonics are valid in the R-bus field of an HP 2100 microinstruction:

NOP A B Q F AAB CAB CQ



Reads all zeros onto the R-bus.

Reads the contents of the A-register onto the R-bus. The information in the register is not altered.

Reads the contents of the B-register onto the R-bus. The information in the register is not altered.

Reads the contents of the Q-register onto the R-bus. The information in the register is not altered.

Reads the contents of the F-register onto the R-bus. The information in the register is not altered.

Reads the contents of either the A- or B-register onto the R-bus, depending upon whether the A-Addressable or B-Addressable flip-flop is set (both cannot be set at the same time). If neither flip-flop is set,

the contents of the A-register are read onto the R-bus. The information in the particular register is not altered.

If COND is coded in the S-bus field, the effect of AAB is slightly different. Refer to the description of COND in this section.

Reads the contents of either the A- or B-register onto the R-bus, depending upon whether bit 11 of the CPU Instruction Register is a zero (A-register) or a one (B-register). The information in the particular register is not altered.

Reads the contents of the Q-register onto the R-bus if bit 9 of the CPU Instruction Register is a one *and* the Index flip-flop is set. The information in the register is not altered.

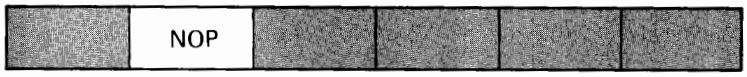
CAUTION

The CQ micro-order is not intended for use in special microprogramming. The use of CQ will effect the operation of module #0 and consequently will cause incorrect operation of HP software. To allow continued use of existing software, it will be necessary to rewrite those instruction routines in module #0 which use the Q-register. As noted elsewhere in this manual, *such changes will void Hewlett-Packard warranties and support quarantees.*

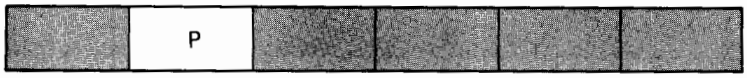
S-BUS FIELD

The following micro-order mnemonics are valid in the S-bus field of an HP 2100 microinstruction:

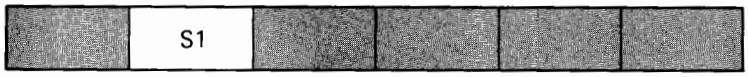
NOP	P	S1	S2	S3	S4	M	T	ADR
CNTR	CL	CR	CIR	IOI	RRS	COND		



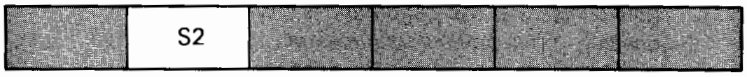
Reads all zeros onto the S-bus.



Reads the contents of the P-register onto the S-bus. The information in the register is not altered.



Reads the contents of Scratch Pad Register 1 onto the S-bus. The information in the register is not altered.



Reads the contents of Scratch Pad Register 2 onto the S-bus. The information in the register is not altered.

Reads the contents of Scratch Pad Register 3 onto the S-bus. The information in the register is not altered.

Reads the contents of Scratch Pad Register 4 onto the S-bus. The information in the register is not altered.

Reads the contents of the M-register onto bits 0-14 of the S-bus (bit 15 of the S-bus is set to a zero). The information in the register is not altered.

Reads the contents of the T-register onto the S-bus. The information in the register is not altered.

Reads bits 0-9 of the CPU Instruction Register onto bits 0-9 of the S-bus. The information in the register is not altered.

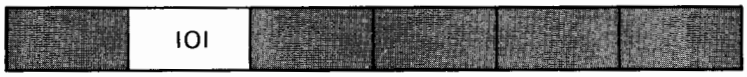
If bit 10 of the CPU Instruction Register is set (1), then bits 10-15 of the P-register are read onto bits 10-15 of the S-bus. The information in the P-register is not altered. If bit 10 of the CPU Instruction Register is clear (0), then bits 10-15 of the S-bus are set to zeros.

Reads the contents of the counter onto bits 0-4 of the S-bus (bits 5-15 of the S-bus are set to zeros). The information in the counter is not altered.

Reads an eight-bit constant onto bits 8-15 of the S-bus (bits 0-7 of the S-bus are set to zeros). The constant is extracted from bits 0-7 of the microinstruction. Note that when CL is coded in the S-bus field, normal execution of the Special and Skip fields is inhibited.

Reads an eight-bit constant onto bits 0-7 of the S-bus (bits 8-15 of the S-bus are set to zeros). The constant is extracted from bits 0-7 of the microinstruction. Note that when CR is coded in the S-bus field, normal execution of the Special and Skip fields is inhibited.

Reads the contents of the Central Interrupt Register onto bits 0-5 of the S-bus (bits 6-15 of the S-bus are set to zeros). The information in the register is not altered.



Reads the contents of the I/O-bus onto the S-bus.

Reads the contents of the R-bus onto the S-bus.

Normally used with AAB coded in the R-bus field.

If AAB is coded in the R-bus field, COND has the following effect:

- If the A-Addressable flip-flop is set, the contents of the A-register are read onto both the R- and S-buses.
- If the B-Addressable flip-flop is set, the contents of the B-register are read onto both the R- and S-buses.
- If neither flip-flop is set, the contents of the T-register are read onto the S-bus.

If something other than AAB is coded in the R-bus field, COND has the following effect:

- If either the A-Addressable or B-Addressable flip-flop is set, the contents of the register specified in the R-bus field are read onto both the R- and S-buses.
- If neither flip-flop is set, the contents of the T-register are read onto the S-bus.

FUNCTION FIELD

The following micro-order mnemonics are valid in the Function field of an HP 2100 microinstruction:

Logical operators:	IOR	XOR	AND	NOR	
Shift operators:	LWF	ARS	LGS	CRS	
Jump operators:	JMP	CJMP	JSB	RSB	
Arithmetic operators:	ADD DEC	ADDO INC	SUB INCO	MPY	DIV
Flip-flop operators:	SOV	CLO	SFLG	CFLG	RFE
	RFI				
Phase operators:	P1A				

The Function field cannot contain a NOP. By convention, an IOR is used whenever a Function field pseudo-NOP is desired. When an IOR is used in this manner, a logical "inclusive OR" is still performed by the function generator.

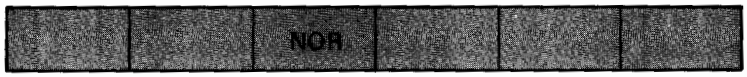
Refer to figure 1-1. The function generator and the shifter use a pair of inputs: the contents of the R-bus and the contents of the S-bus. If a non-shifting operation is specified (e.g., ADD, IOR, AND, etc), the result of the operation passes from the function generator onto the ALU-bus, into the shifter, and then onto the T-bus *without being altered*. If a shift operation is specified, the result is available as described under the individual shift mnemonics (LWF, ARS, LGS, and CRS) later in this section.

LOGICAL OPERATORS

Causes the function generator to perform a logical "inclusive OR".

Causes the function generator to perform a logical "exclusive OR".

Causes the function generator to perform a logical "AND".



Causes the function generator to perform a logical "NOR". If a NOP is specified in either the R-bus or S-bus field, the complement of the other is obtained. If both the R-bus and S-bus fields contain a NOP, the function generator passes all ones onto the ALU-bus.

SHIFT OPERATORS

The LWF micro-order allows the programmer to shift the contents of the Flag flip-flop in conjunction with the contents of a register.

If L1 is coded in the Special field, the contents of the Flag flip-flop and the contents of the register together are rotated (circular shift) one bit position to the left. The flip-flop bit is shifted into bit 0 of the register, bit 15 of the register is shifted into the flip-flop, and bits 0-14 of the register are shifted one position to the left.

If R1 is coded in the Special field, the contents of the Flag flip-flop and the contents of the register together are rotated (circular shift) one bit position to the right. The flip-flop bit is shifted into bit 15 of the register, bit 0 of the register is shifted into the flip-flop, and bits 1-15 of the register are shifted one position to the right.

LWF also causes the function generator to perform an IOR.

Causes an arithmetic shift to be performed on a 32-bit data item.

The mnemonic in the Special field determines the direction of the shift (R1 = right; L1 = left).

For right shifts, the B- and A-registers are used: the B-register contains the sign bit plus the high-order fifteen data bits and the A-register contains the low-order sixteen data bits. All 32 bits are shifted one bit position to the right (the sign bit is unchanged, bit 0 of the A-register is lost).

The required microcoding is

```
B   -  ARS  B  R1  *
```

The Skip field (*) is available for any valid use.

For left shifts, the F- and Q-registers are used: the F-register contains the sign bit plus the high-order fifteen data bits and the Q-register contains the low-order sixteen data bits. The sign bit is unchanged and the 31 data bits are shifted one bit position to the left (bit 14 of the F-register is lost, bit 0 of the Q-register is set to a zero).

The required microcoding is

```
F   -  ARS  F  L1  *
```


The Skip field (*) is available for any valid use.

Causes a logical shift to be performed on a 32-bit data item.

The mnemonic in the Special field determines the direction of the shift (R1 = right; L1 = left).

For right shifts, the B- and A-registers are used: the B-register contains the high-order sixteen bits and the A-register contains the low-order sixteen bits. All 32 bits are shifted one bit position to the right (bit 15 of the B-register is set to zero, bit 0 of the A-register is lost).

The required microcoding is

```
B  -  LGS  B  R1  *
```

The Skip field (*) is available for any valid use.

For left shifts, the F- and Q-registers are used: the F-register contains the high-order sixteen bits and the Q-register contains the low-order sixteen bits. All 32 bits are shifted one bit position to the left (bit 15 of the F-register is lost, bit 0 of the Q register is set to a zero).

The required microcoding is

```
F  -  LGS  F  L1  *
```

The Skip field (*) is available for any valid use.

Causes a circular shift to be performed on a 32-bit data item.

The mnemonic in the Special field determines the direction of the shift (R1 = right; L1 = left).

For right shifts, the B- and A-registers are used: the B-register contains the high-order sixteen bits and the A-register contains the low-order sixteen bits. All 32 bits are rotated one bit position to the right (bit 0 of the A-register is shifted into bit 15 of the B-register).

The required microcoding is

```
B  -  CRS  B  R1  *
```

The Skip field (*) is available for any valid use.

For left shifts, the F- and Q-registers are used: the F-register contains the high-order sixteen bits and the Q-register contains the low-order sixteen bits. All 32 bits are rotated one bit position to the left (bit 15 of the B-register is shifted into bit 0 of the A-register).

The required microcoding is

```
F  -  CRS  F  L1  *
```

The Skip field is available for any valid use.

JUMP OPERATORS

Causes control to pass to the specified jump address.

Causes control to pass to the specified jump address only if an I/O interrupt or a front panel halt has occurred. In single cycle operation, the computer halts unconditionally upon execution of a CJMP micro-order (this is useful in diagnostics) and the jump is not executed.

Causes control to pass to the specified jump address. The address of the next sequential microinstruction is saved as a return address. This micro-order is used for passing control to a subroutine.

Causes control to pass to the return address. This micro-order is used for exiting from a subroutine.

ARITHMETIC OPERATORS

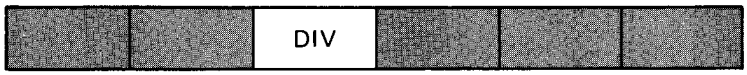
Adds the contents of the S-bus to the contents of the R-bus (the overflow logic is disabled). The overflow logic is discussed under ADDO, below.

Adds the contents of the S-bus to the contents of the R-bus (the overflow logic is enabled). If the sign (bit 15) of the R- and S-buses are

the same (both positive or both negative) and the sign of the ALU-bus is different, the Overflow flip-flop is set. Note that if the Overflow flip-flop is set prior to execution of an ADDO micro-order and the ADDO operation does *not* result in an overflow condition, the Overflow flip-flop is *not* cleared.

Subtracts the contents of the S-bus from the contents of the R-bus in two's complement form.

Multiply. This micro-order is normally used in a repeat loop as part of a multiply algorithm. See the description of MPY on page 4-10 of the *2100 Computer Microprogramming Guide.*



Divide. This micro-order is normally used in a repeat loop as part of a divide algorithm. See the description of DIV on page 4-9 of the *2100 Computer Microprogramming Guide.*

Subtracts the contents of the S-bus from the contents of the R-bus in

one's complement form. If the S-bus contains all zeros, the contents of the R-bus are decremented by one.

Adds the contents of the S-bus to the contents of the R-bus and increments the sum by one (the overflow logic is disabled). The overflow logic is discussed under ADDO, above.

Adds the contents of the S-bus to the contents of the R-bus and increments the sum by one (the overflow logic is enabled). The overflow logic is discussed under ADDO, above.

FLIP-FLOP OPERATORS

Sets the Overflow flip-flop on (also causes the function generator to perform an IOR).

Sets the Overflow flip-flop off (also causes the function generator to perform an IOR).

Sets the Flag flip-flop on (also causes the function generator to perform an IOR).

Sets the Flag flip-flop off (also causes the function generator to perform an IOR).

Exchanges the contents of the Flag and Extend flip-flops (also causes the contents of the R-bus to be read onto the T-bus).

Exchanges the contents of the Flag and Index Mode flip-flops.

CAUTION

The RFI micro-order is not intended for use in special microprogramming. The use of RFI will affect the operation of module #0 and consequently will cause incorrect operation of HP software. To allow continued use of existing software, it would be necessary to rewrite those instruction routines in module #0 which use the Q-register. As noted elsewhere in this manual, *such changes will void Hewlett-Packard warranties and support guarantees.*

PHASE OPERATORS

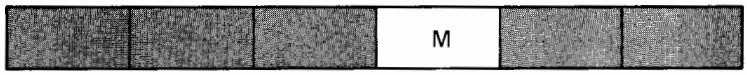
Sets phase 1A (the "fetch" phase) and clears the current phase (normally the "execute" phase). This micro-order is used mainly in diagnostics.

STORE FIELD

The following micro-order mnemonics are valid in the Store field of an HP 2100 microinstruction:

NOP	M	T	IR	A	B	Q	F	P	S1
S2	S3	S4	IOO	AAB	CAB				

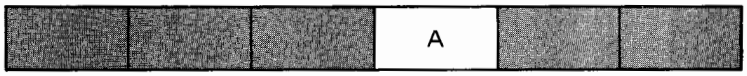
No store.



Stores the contents of bits 0-14 of the S-bus in the M-register.

Stores the contents of the S-bus in the T-register.

Stores the contents of the S-bus in the CPU Instruction Register.



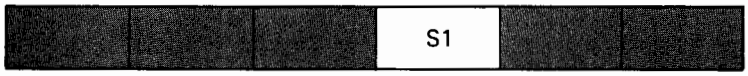
Stores the contents of the T-bus in the A-register.

Stores the contents of the T-bus in the B-register.

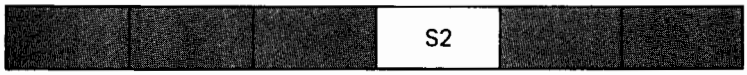
Stores the contents of the T-bus in the Q-register.

Stores the contents of the T-bus in the F-register.

Stores the contents of the T-bus in the P-register.



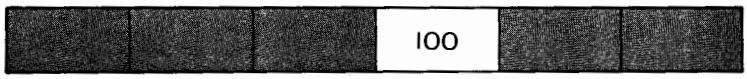
Stores the contents of the T-bus in Scratch Pad Register 1.



Stores the contents of the T-bus in Scratch Pad Register 2.

Stores the contents of the T-bus in Scratch Pad Register 3.

Stores the contents of the T-bus in Scratch Pad Register 4.



Reads the contents of the S-bus onto the I/O-bus.

Stores the contents of the T-bus in either the A- or B-register, depending upon whether the A-Addressable or B-Addressable flip-flop is set. If neither flip-flop is set, no store occurs.

Stores the contents of the T-bus in either the A- or B-register, depending upon whether bit 11 of the CPU Instruction Register is a zero (A-register) or a one (B-register).

SPECIAL FIELD

The following micro-order mnemonics are valid in the Special field of an HP 2100 microinstruction:

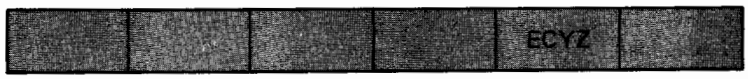
NOP	CNTR	CW	ECYN	ECYZ	IOG1
L1	R1	RSS	RW	AAB	ASG1
ASG2	LEP	SRG1	SRG2		

No operation.

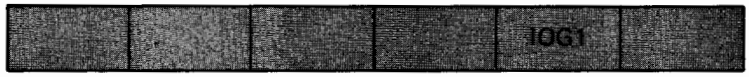
Reads bits 0-3 of the S-bus into bits 0-3 of the counter (bit 4 of the counter is set to a zero).

Initiates a "write-into-memory" operation. See the discussion "Accessing Core Memory Locations" in section 1 of this manual.

Sets the Carry flip-flop if the T-bus does *not* contain all zeros. When the Carry flip-flop is set, the P-register is automatically incremented by one upon exiting from the microprogram.



Sets the Carry flip-flop if the T-bus contains all zeros. When the Carry flip-flop is set, the P-register is automatically incremented by one upon exiting from the microprogram.



Initiates an input/output operation. See the discussion "Input/Output" in section 1 of this manual.

Specifies a left shift. See the description of the shift mnemonics (LWF, ARS, LGS, and CRS) earlier in this section.

Specifies a right shift. See the description of the shift mnemonics (LWF, ARS, LGS, and CRS) earlier in this section.

Reverses the condition specified in the Skip field. For example, if the Skip field contains TBZ (skip if the T-bus contains all zeros), an RSS in

the Special field changes the skip condition to "skip if the T-bus does *not* contain all zeros".

Initiates a "read-from-memory" operation. See the discussion "Accessing Core Memory Locations" in section 1 of this manual.

RW also enables the setting of the A-Addressable and B-Addressable flip-flops. See the discussion of AAB, below.

Enables the setting of the A-Addressable and B-Addressable flip-flops. If bit 0 of the ALU-bus is a zero and bits 1-14 of the T-bus are all zeros, the A-Addressable flip-flop is set. If bit 0 of the ALU-bus is a one and bits 1-14 of the T-bus are all zeros, the B-Addressable flip-flop is set.

Enables the skip and extend logic specified by bits 0 and 3-7 of the CPU Instruction Register.

Enables the skip and increment logic specified by bits 0-2 of the CPU Instruction Register.

Legal entry point. This micro-order is only used in module #0. See the discussion of LEP on page 4-14 of the *2100 Computer Micro-programming Guide.*

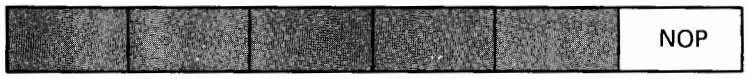
Enables the shift-rotate group functions specified by bits 6-9 of the CPU Instruction Register.

Enables the shift-rotate group functions specified by bits 0-2 and 4 of the CPU Instruction Register.

SKIP FIELD

The following micro-order mnemonics are valid in the Skip field of an HP 2100 microinstruction:

NOP	EOP	COUT	CTR	CTRI	FLG	ICTR
NEG	NMPV	ODD	OVF	RPT	TBZ	UNC
AAB	NAAB					

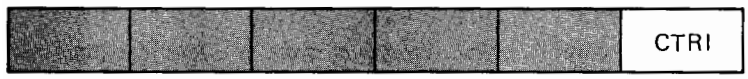


No skip.

End-of-phase. This micro-order is used for exiting from a micro-program. The exit occurs *after* the next sequential microinstruction is executed.

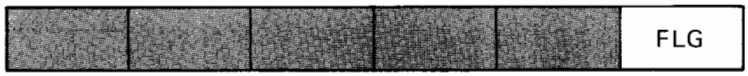
Skips the next sequential microinstruction if a carry-out from bit 15 of the ALU-bus occurs during execution of the current microinstruction. A carry-out can result from an ADD, ADDO, SUB, INC, INCO, MPY or DIV function.

Skips the next sequential microinstruction if bits 0-3 of the counter are all ones (octal 17). Bit 4 of the counter is ignored. The contents of the counter are not altered.

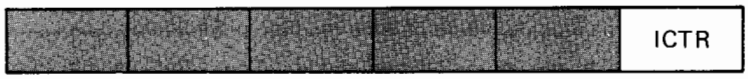


Skips the next sequential microinstruction if bits 0-3 of the counter are

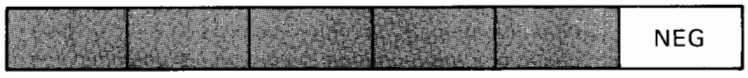
all ones (octal 17). Bit 4 of the counter is ignored. The counter is always incremented by one after the test.



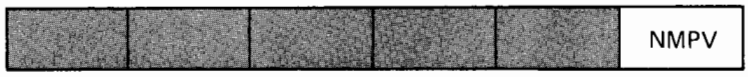
Skips the next sequential microinstruction if the Flag flip-flop is set. The setting of the flip-flop is not altered. FLG tests the state of the flip-flop as it was just prior to execution of the current microinstruction.



Increments the counter by one.



Skips the next sequential microinstruction if the output of the function generator is negative (if bit 15 of the ALU-bus is set).



Skips the next sequential microinstruction if either of the following is true:

a) Memory protect is disabled and both the A-Addressable and B-Addressable flip-flops are clear.

b) Memory protect is enabled, no memory protect violation is detected, and both the A-Addressable and B-Addressable flip-flops are clear.

Skips the next sequential microinstruction if the output of the function generator is odd (if bit 0 of the ALU-bus is set).

Skips the next sequential microinstruction if the Overflow flip-flop is set. OVF does not alter the setting of the flip-flop. OVF tests the state of the Overflow flip-flop as it was just prior to execution of the current microinstruction.

Causes the next sequential microinstruction to be executed repeatedly until the condition specified in its Skip field is true.

Some restrictions:

- The next sequential microinstruction must *not* contain TBZ in its Skip field.
- The next sequential microinstruction must *not* have an add-type mnemonic (ADD, INC, etc.) in its Function field if its Skip field contains NEG or ODD.

Both of the above restrictions apply even if the skip condition is reversed (RSS in the Special field).

Skips the next sequential microinstruction if the T-bus contains all zeros.

Skips the next sequential microinstruction unconditionally.

Skips the next sequential microinstruction if either the A-Addressable or B-Addressable flip-flop is set. The setting of the flip-flops is not altered.

Skips the next sequential microinstruction if bits 1-14 of the ALU-bus are *not* all zeros.

The nine statements described in this section control the assembly process. Each assembler control statement must begin in character position 1 and may not contain embedded blanks.

With the exception of $ORIGIN and $END, all assembler control statements should appear ahead of the first executable microinstruction.

There may be more than one $ORIGIN statement. They may be placed anywhere in the source microprogram.

The $END statement must be the final statement in the source microprogram.

If an erroneous assembler control statement is detected, the Microassembler prints BAD CONTROL STATEMENT and the erroneous statement on the system console device and then the computer halts. The user is expected to repunch the erroneous card, place it at the front of the deck in the input hopper, and press the RUN switch on the computer front panel.

$INPUT = L.U.x

x = logical unit number of a card reader or paper tape reader.

Causes all subsequent input to be read through the specified device.

When the assembly process is first begun, the Microassembler expects the first source statement to be entered through the system console device. The user may enter the whole source program through the system console device. Most often, however, he merely enters an

$INPUT command specifying through what card reader or paper tape photoreader the rest of the source program is to be read.

$PASS2=L.U.x

x = logical unit number of a magnetic tape drive.

Causes all subsequent input to be recorded on magnetic tape for use as input to Pass 2. If omitted, the computer halts at the end of Pass 1 to allow the operator to reload the source microprogram in the $INPUT device.

$LIST=L.U.x

x = logical unit number of a printing device.

Causes the assembly listing to be printed by the specified device. If omitted, L.U.6 is assumed.

$OUTPUT=L.U.x

x = logical unit number of a paper tape punch.

Causes the object code produced by the assembler to be routed to the specified device. If omitted, L.U.4 is assumed.

$EXTERNALS=<name 1><octal address 1> , . . .
,<name n><octal address n>

name 1 through name n are symbols; octal address 1 through octal address n are control store addresses.

Assigns symbolic addresses to control store addresses that are external to the program being assembled. External symbols must conform to the rules for forming statement labels. Each symbol in the list, along with the associated octal address, is entered into the symbol table. Once defined in this manner, external symbols may be used as jump addresses in JMP, JSB, and CJMP microinstructions.

$SUPPRESS

Suppresses all warning error messages. Fatal error messages are not affected.

$DEBUG

Specifies that the debug option is to be used. See section 9, "HP Micro Debug Editor", of this manual for the implications of $DEBUG.

$ORIGIN=xxxx

xxxx = octal control store address

Sets the program location counter in the microassembler to the specified value. If more than one $ORIGIN statement is present, the specified control store addresses must occur in ascending order.

$END

Signals the end of the source microprogram.

This section presents four sample microprograms:

1) A "save registers" microprogram that stores the contents of all the registers in core memory locations.

2) A "block move" microprogram that moves a group of computer words from one place in core memory to another.

3) A "table search" microprogram that scans a group of computer words in core memory until a specified character is found.

4) A "teleprinter output driver" microprogram that sends characters from a user-specified output buffer in core memory to the teleprinter.

Each example is presented in the following manner. First, there is a prose description telling what the sample microprogram does, how it is called, and what information it passes to the calling program. Then there is a listing of the actual microinstructions. Finally, there is a microinstruction by microinstruction commentary describing how the microprogram works. To aid the reader, both the microprogram listing and the microinstruction commentary are divided into functional segments. If several microinstructions together perform a specific task, the particular group of microinstructions is easily discerned.

The examples are meant to be documented thoroughly enough so that no additional help is required to understand them.

REGISTER SAVE MICROPROGRAM

This sample microprogram copies the contents of all the registers into core memory locations as shown in Table 6-1.

```
DUMP            IOR   M     CW    UNC   ADDRESS LOC 0
                IOR
          F     RRS   IOR   T           STORE F IN LOC 0

                CW    IOR   F     67    PUT 67 IN F
          F     RRS   IOR   M     CW    UNC   ADDRESS LOC 67
                IOR
                S1    IOR   T           STORE S1 IN LOC 67

          F           INC   F           INCREMENT F
          F     RRS   IOR   M     CW    UNC   ADDRESS LOC 70
                IOR
                S2    IOR   T           STORE S2 IN LOC 70

          F           INC   F           INCREMENT F
          F     RRS   IOR   M     CW    UNC   ADDRESS LOC 71
                IOR
                S3    IOR   T           STORE S3 IN LOC 71

          F     RRS   IOR   M     CW    UNC   ADDRESS LOC 72
                IOR
                S4    IOR   T           STORE S4 IN LOC 72

          F           INC   F           INCREMENT F
          F     RRS   IOR   M     CW    UNC   ADDRESS LOC 73
                IOR
          A     RRS   IOR   T           STORE A IN LOC 73

          F           INC   F           INCREMENT F
          F     RRS   IOR   M     CW    UNC   ADDRESS LOC 74
                IOR
          B     RRS   IOR   T           STORE B IN LOC 74

          F           INC   F           INCREMENT F
          F     RRS   IOR   M     CW    UNC   ADDRESS LOC 75
                IOR
                P     IOR   T           STORE P IN LOC 75

          F           INC   F           INCREMENT F
          F     RRS   IOR   M     CW    UNC   ADDRESS LOC 76
                IOR
          Q     RRS   IOR   T           STORE Q IN LOC 76

          F           INC   S1          INCREMENT F AND STORE IN S1

                IOR   M     RW          ADDRESS LOC F
                T     IOR   F           RESTORE F FROM LOC 0

                S1    IOR   M     CW    UNC   ADDRESS LOC 77
                IOR
          F     RRS   IOR   T     EOP   STORE F IN LOC 77
                IOR                     EXIT
```

Figure 6-1. Register Save Microprogram

Table 6-1. Register Save Locations

Register	Core Memory Locations (octal)
S1	67
S2	70
S3	71
S4	72
A	73
B	74
P	75
Q	76
F	77

The register save microprogram is called by other microprograms through use of the JSB micro-order. No parameters are passed.

MICROINSTRUCTION COMMENTARY

```
DUMP  - -   IOR  M   CW    UNC   Address core memory location 0.
      - -   IOR  -   -     -
      F RRS IOR  T   -     -     Store the contents of the
                                 F-register in core memory
                                 location 0.

      - CR  IOR  F   67          Store the value 67 (octal) in the
      F RRS IOR  M   CW    UNC   F-register. Address core memory
      - -   IOR  -   -     -     location 67.
      - S1  IOR  T   -     -     Store the contents of Scratch Pad
                                 Register 1 in core memory
                                 location 67.
```

F	-	INC	F	-	-	Increment the F-register.
F	RRS	IOR	M	CW	UNC	Address core memory location
-	-	IOR	-	-	-	70.
-	S2	IOR	T	-	-	Store the contents of Scratch Pad Register 2 in core memory location 70.

And so forth . . .

F	-	INC	F	-	-	Increment the F-register.
F	RRS	IOR	M	CW	UNC	Address core memory location
-	-	IOR	-	-	-	76.
Q	RRS	IOR	T	-	-	Store the contents of the Q-register in core m emory location 76.
F	-	INC	S1	-	-	Increment the F-register and store the result in Scratch Pad Register 1.
-	-	IOR	M	RW	-	Address core memory location 0.
-	T	IOR	F	-	-	Restore the F-register from core memory location 0.
-	S1	IOR	M	CW	UNC	Address core memory location
-	-	IOR	-	-	-	77.
F	RRS	IOR	T	-	EOP	Store the contents of the
-	-	IOR	-	-	-	F-register in core memory location 77 and exit.

BLOCK MOVE MICROPROGRAM

This sample microprogram moves a group of computer words from one place in core memory to another. When the microprogram receives control, it is assumed that:

- The number of words to be moved is in the A-register (in two's complement form).
- The FROM address is in the B-register.
- The TO address is in the core memory location pointed to by the P-register.

The HP assembly language calling sequence is as follows:

```
LDA   -(number-of-words)
LDB    from-address
105xxx
DEF    to-address (cannot be indirect)
```

```
MOV       A    IOR        RSS  TBZ  CHARACTER COUNT = ZERO ?
          JMP        OUT           YES, EXIT (ELSE PROCEED)
          P    IOR  M    RW         GET 'TO' ADDRESS
          T    IOR  Q               PUT IT IN Q
-------------------------------------------------------------
LOOP  B   RWS  IOR  M    RW         READ A DATA WORD
          T    IOR  S1              PUT IT IN S1
-------------------------------------------------------------
          Q    IOR  S2              PUT 'TO' ADDRESS IN S2
          F S2 DEC  M    CW   NMPV ADDRESS THE 'TO' LOCATION
               JMP        OUT      (MEMORY PROTECT VIOLATION)
          S1   IOR  T               WRITE A DATA WORD TO MEMORY
-------------------------------------------------------------
          B    INC  B               INCREMENT THE 'FROM' ADDRESS
          Q    INC  Q               INCREMENT THE 'TO' ADDRESS
          A    INC  A          TBZ  DECREMENT AND TEST THE COUNTER
               JMP        LOOP     REPEAT THE MOVE LOOP
-------------------------------------------------------------
OUT       P    INC  P          EOP  INCREMENT THE P REGISTER
               IOR                  EXIT
```

Figure 6-2. Block Move Microprogram

MICROINSTRUCTION COMMENTARY

MOV	-	P	IOR	M	RW	-	Get the TO address and store it
	-	T	IOR	Q	-	-	in the Q-register. The TO address cannot be indirect.

LOOP	B	RRS	IOR	M	RW	-	Read a data word from the core memory location pointed to by the FROM address and store the data word in Scratch Pad Register 1.
	-	T	IOR	S1	-	-	
	Q	-	IOR	S2	-	-	Put the TO address in Scratch Pad Register 2. Address the TO core memory location. Write the data word into the core memory location pointed to by the TO address. The F, DEC, and NMPV micro-orders in the "write into memory" microinstruction test the TO address to make sure it does not refer to a location in the protected portion of core memory. If a memory protect violation is detected, control passes to OUT (otherwise the "write into memory" operation is performed).
	F	S2	DEC	M	CW	NMPV	
	-	-	JMP	-	OUT		
	-	S1	IOR	T	-	-	
	B	-	INC	B	-	-	Increment the FROM address.
	Q	-	INC	Q	-	-	Increment the TO address.
	A	-	INC	A	-	TBZ	Increment and test the number of words (remember that the number of words is in two's complement form; consequently, the number is effectively decremented). If the number = 0, control passes to OUT. Otherwise, the move loop is repeated.
	-	-	JMP	-	LOOP		
OUT	-	P	INC	P	-	EOP	Increment the P-register and exit.
	-	-	IOR	-	-	-	

TABLE SEARCH MICROPROGRAM

This sample microprogram searches a table for a specific character. Each word in the table contains two characters: one in the high byte position (bits 8-15) and one in the low byte position (bits 0-7).

The calling program passes the following parameters:

- The address of the first byte to be examined. Bits 1-15 specify the starting core memory location while bit 0 specifies whether the table search is to begin with the high or low byte (0 = high; 1 = low).
- The number of bytes to be examined.
- The character being searched for.
- A terminator character.

The table is searched until the specified character is found, until the terminator character is found, or until the specified number of bytes have been examined. If the starting byte address is even, the search is performed as shown in Table 6-2.

If the starting byte address is odd, the search is performed as shown in Table 6-3.

Table 6-2. Even Starting Byte Address

	High Byte	Low Byte
Starting core memory location	1	2
Next higher core memory location	3	4
Next higher core memory location	5	6
And so forth . . .		

```
SERCH   A       IOR             RSS  TBZ   BYTE COUNT = ZERO ?
                JMP             OUT        YES, EXIT (ELSE PROCEED)
                RFE   M         CM   UNC   ADDRESS LOC B AND CLEAR E
                IOR
        P       IOR   T                    STORE P IN LOC B

        B       IOR   Q                    STORE BYTE ADDRESS IN Q

        A  CL   AND   S1   377             FORM HIGH BYTE 'TEST' CONSTANT
        A  CM   AND   S3   377             FORM LOW BYTE 'TERM' CONSTANT

           CR   IOR   S2   10              STORE 10 (OCTAL) IN S2
           S2   IOR         CNTR RPT      INITIALIZE THE COUNTER
        B       CRS   B    R1   CTRI      ROTATE B&A RIGHT B BIT POSITIONS

        A  CR   AND   S2   377             FORM LOW BYTE 'TEST' CONSTANT
        B  CL   AND   S4   377             FORM HIGH BYTE 'TERM' CONSTANT

           P    IOR   M    RW              GET THE BYTE COUNT
        Q       IOR   B                    RESTORE THE BYTE ADDRESS
           T    IOR   Q                    STORE BYTE COUNT IN Q

        B       IOR   P    R1   ODD       BYTE ADDRESS ODD OR EVEN?
                JMP        RPEAT         (EVEN)

           P    SFLG  M    RW              READ TWO BYTES
           T    IOR   A                    STORE IN A
                JMP        LOW           SKIP FIRST HIGH BYTE TEST

RPEAT      P    SFLG  M    RW              READ TWO BYTES
           T    IOR   A                    STORE IN A

        A  CL   AND   B    377             ISOLATE HIGH BYTE

        B  S1   XOR        RSS  TBZ       BYTE = 'TEST' ?
                JMP        TESTH          (YES)

        B  S4   XOR        RSS  TBZ       BYTE = 'TERM' ?
                JMP        TERMH          (YES)

        Q       INC   Q    RSS  TBZ       DECREMENT AND TEST BYTE COUNT
                JMP        EXIT           (BYTE COUNT = 0)

LOW     A  CR   AND   B    377             ISOLATE LOW BYTE

        B  S2   XOR        RSS  TBZ       BYTE = 'TEST' ?
                JMP        TESTL          (YES)

        B  S3   XOR        RSS  TBZ       BYTE = 'TERM' ?
                JMP        TERML          (YES)

           P    INC   P                    INCREMENT BYTE ADDRESS

        Q       INC   Q         TBZ       DECREMENT AND TEST BYTE COUNT
                JMP        RPEAT          (BYTE COUNT = 0)

EXIT    Q       IOR   B                    SET B TO ALL ZEROS
                IOR   P                    SET P TO ALL ZEROS

TESTH           RFE                        EXCHANGE FLAG & E
                SFLG              UNC      SET FLAG = 1

TERML           CFLG              UNC      SET FLAG = 0

TESTL           RFE                        EXCHANGE FLAG & E

TERMH      P    IOR   A    L1   FLG       SET A TO ZEROS AND SHIFT LEFT

        A       INC   A                    SET ON 'HIGH BYTE INDICATOR'
                IOR   M    RW   EOP       ADDRESS LOC 0
           T    INC   P                    FIX THE P REGISTER & EXIT
```

Figure 6-3. Table Search Microprogram

Table 6-3. Odd Starting Byte Address

	High Byte	Low Byte
Starting core memory location		1
Next higher core memory location	2	3
Next higher core memory location	4	5
And so forth . . .		

The assembly language calling sequence is as follows:

```
LDA   < test > < term >
LDB   starting-byte-address
105xxx
DEC   number-of-bytes
```

where test is the character being searched for.

term is the terminator character.

starting-byte-address is the address of the first byte to be examined.

number-of-bytes is the number of bytes to be examined.

After the table search is complete, the microprogram passes the following information to the calling program:

A-register: The address of the last byte examined (bits 1-15 specify the core memory location and bit 0 is the high/low byte indicator).

All zeros if neither the specified character nor the terminator character was found.

B-register: The last byte examined.

All zeros if neither the specified character nor the terminator character was found.

Extend flip-flop: 1 = specified character found (or neither the specified character nor the terminator character was found).

0 = terminator character found.

MICROINSTRUCTION COMMENTARY

Note: The Flag flip-flop is always clear (0) when the microprogram receives control.

-	-	RFE	M	CW	UNC	Address core memory location 0
-	-	IOR	-	-	-	and clear the Extend flip-flop.
-	P	IOR	T	-	-	Store the contents of the P-register in core memory location 0.
B	-	IOR	Q	-	-	Store the starting byte address in the Q-register.
A	CL	AND	S1	377		Save the test character as a "high
A	CR	AND	S3	377		byte compare constant" and save the terminator character as a "low byte compare constant".
-	CR	IOR	S2	10		Set the counter to 10 (octal) and
-	S2	IOR	-	CNTR	RPT	rotate the B- and A-registers eight
B	-	CRS	B	R1	CTRI	bit positions to the right.
A	CR	AND	S2	377		Save the test character as a "low
B	CL	AND	S4	377		byte compare constant" and save the terminator character as a "high byte compare constant".

```
        - P   IOR   M   RW    -     Read the number of bytes into
        Q -   IOR   B   -     -     the Q-register (first restoring the
        - T   IOR   Q   -     -     starting byte address to the
                                    B-register).

        B -   IOR   P   R1    ODD   Test the starting byte address to
        - -   JMP   -   RPEAT       determine whether the search
                                    should begin with the high or low
                                    byte (odd=low; even=high), shift
                                    the byte address one bit position
                                    to the right, and store it in the
                                    P-register.

                                    If the address is even, control
                                    passes to RPEAT. Otherwise,
                                    continue.

        - P   SFLG  M   RW    -     Set the Flag flip-flop and read
        - T   IOR   A   -     -     the first word in the table. Store
        - -   JMP   -   LOW         the word in the A-register.
                                    Control passes to LOW.

RPEAT   - P   SFLG  M   RW    -     Set the Flag flip-flop and read a
        - T   IOR   A   -     -     table word. Store the word in the
                                    A-register.

        A CL  AND   B   377         Isolate the high byte in the
                                    B-register.

        B S1  XOR   -   RSS   TBZ   Byte = test character?
        - -   JMP   -   TESTH       Yes, control passes to TESTH.
                                    Otherwise, continue.

        B S4  XOR   -   RSS   TBZ   Byte = terminator character?
        - -   JMP   -   TERMH       Yes, control passes to TERMH.
                                    Otherwise, continue.

        Q -   INC   Q   RSS   TBZ   Increment and test the byte
        - -   JMP   -   EXIT  -     count (remember that the count
                                    is in two's complement form;
                                    consequently, the count is
                                    effectively decremented). If the
```

```
                                                  count = 0, control passes to
                                                  EXIT. Otherwise, continue.

LOW     A CR    AND  B  377                       Isolate the low byte in the
                                                  B-register.

        B S2    XOR  -  RSS      TBZ              Byte = test character?
        - -     JMP  -  TESTL                     Yes, control passes to TESTL.
                                                  Otherwise, continue.

        B S3    XOR  -  RSS      TBZ              Byte = terminator character?
        - -     JMP  -  TERML                     Yes, control passes to TERML.
                                                  Otherwise, continue.

        - P     INC  P  -        -                Increment the byte address.

        Q -     INC  Q  -        TBZ              Increment and test the byte
        - -     JMP  -  RPEAT                     count (remember that the count
                                                  is in two's complement form;
                                                  consequently, the count is
                                                  effectively decremented). If the
                                                  count = 0, continue. Otherwise,
                                                  repeat the byte search loop.

EXIT    Q -     IOR  B  -        -                Set the B- and P-registers to all
        - -     IOR  P  -        -                zeros.

TESTH   - -     RFE  -  -        -                Exchange the Flag and Extend
        - -     SFLG -  -        UNC              flip-flops. Then set the Flag
                                                  flip-flop.
                                                  Control passes to TESTL.

TERML   - -     CFLG -  -        -                Clear the Flag flip-flop.

TESTL   - -     RFE  -  -        -                Exchange the Flag and Extend
                                                  flip-flops.

TERMH   - P     IOR  A  L1       FLG              Store the byte address in the
                                                  A-register (shifting the address
                                                  one bit position to the left).
                                                  Then test the Flag flip-flop. If
```

						the Flag flip-flop is set, skip the next microinstruction.
A	-	INC	A	-	-	Set bit 0 of the A-register.
-	-	IOR	M	RW	EOP	Address core memory location 0.
-	T	IOR	P	-	-	Read the contents of core memory location 0 into the P-register and then exit.

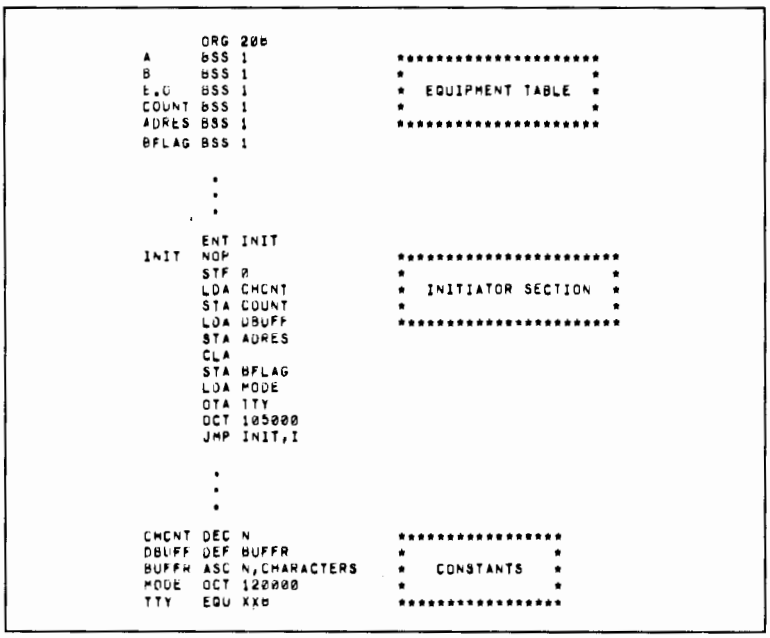
TELEPRINTER OUTPUT DRIVER

This example presents a teleprinter output driver program. The driver is in two parts: an Initiator Section and a Continuator Section. The Initiator Section resides in core memory while the Continuator Section is a microprogram residing in Module #1 (starting at control store location 400_8).

A 105000 macro instruction resides in the teleprinter interrupt location in low core memory.

During any given teleprinter output operation, the Initiator Section is executed only once while the Continuator Section is executed once for each character that is sent to the teleprinter.

To begin a teleprinter output operation, the user prepares an output buffer (BUFFR) and a character-count constant (CHCNT) in core memory and transfers control to the Initiator Section. The Initiator Section passes parameters to a low core equipment table and then transfers control to the Continuator Section. Thereafter, the Continuator Section is entered on an interrupt basis as many times as are necessary to satisfy the specified character count. The Continuator Section essentially sends one character to the teleprinter and then returns control to the interrupted program.



```
        ORG 20B
A       BSS 1               **********************
B       BSS 1               *                    *
E,O     BSS 1               *  EQUIPMENT TABLE   *
COUNT   BSS 1               *                    *
ADRES   BSS 1               **********************
BFLAG   BSS 1

         .
         .
         .

        ENT INIT
INIT    NOP                 ************************
        STF 0               *                      *
        LDA CMCNT           *  INITIATOR SECTION   *
        STA COUNT           *                      *
        LDA DBUFF           ************************
        STA ADRES
        CLA
        STA BFLAG
        LDA MODE
        OTA TTY
        OCT 105000
        JMP INIT,I

         .
         .
         .

CMCNT   DEC N               *****************
DBUFF   DEF BUFFR           *               *
BUFFR   ASC N,CHARACTERS    *   CONSTANTS   *
MODE    OCT 120000          *               *
TTY     EQU XXB             *****************
```

Figure 6-4. Initiator Section

When the microprogram receives control, it is assumed that the following parameters are in the low core equipment table:

Core Memory Location	Contents
23	Number of characters yet to be printed.
24	Address of the core memory location that contains the next character to be printed.
25	A "high/low byte" flag (zero = high byte; non-zero = low byte).

Each location in the output buffer contains two characters: one in the high byte position (bits 8-15) and one in the low byte position (bits 0-7). At any given time, the microprogram must know whether it is to send the high or low byte character to the teleprinter. Core memory location 25 in the equipment table is used for this purpose.

Upon entry, the microprogram reads the contents of core memory location 25 into Scratch Pad Register 1 and uses Scratch Pad Register 1 as the "high/low byte" indicator, as follows:

zero = Print the high byte character.

non-zero = Print the low byte character.

When the microprogram receives control from the Initiator Section, core memory location 25 contains zero. Whenever it prints a high byte character, the microprogram sets Scratch Pad Register 1 to non-zero. Similarly, whenever it prints a low byte character, the microprogram resets Scratch Pad Register 1 to zero. Just before exiting, the microprogram reads the contents of Scratch Pad Register 1 into core memory location 25.

Every time it is executed, the microprogram decrements the character-count (core memory location 23). The character-address (core memory location 24) is incremented only after a low byte character is printed.

The character output operation is performed as follows:

1) The microprogram tests the character count.

   If the count is *not* zero, the microprogram proceeds with the character output operation.

   If the count is zero, the microprogram forms a CLC instruction, puts it in the CPU Instruction Register, and causes the I/O decoders to decode and execute the instruction. This clears the teleprinter output interrupt. Control then returns to the interrupted program.

```
            CM    IOR   S1    20          SAVE A IN LOC 20
            S1    IOR   M     CW    UNC
                  IOR
       A    RRS   IOR   T

            S1    INC   S2                SAVE B IN LOC 21
            S2    IOR   M     CW    UNC
                  IOR
       B    RRS   IOR   T

            S2    INC   S1                SAVE OVF=FF & E IN LOC 22
                  RFE   Q     RSS   OVF
       Q          INC   Q
       Q          LWF   Q     L1
            S1    IOR   M     CW    UNC
                  IOR
       Q    RRS   IOR   T

            S1    INC   S2                READ CHAR-COUNT INTO Q
            S2    IOR   M     RW
            S2    INC   S1
            T     IOR   Q     RSS   TBZ
                  JMP         CLC

            S1    IOR   M     RW          READ CHAR-ADDRESS INTO A
            S1    INC   S2
            T     IOR   A

            S2    IOR   M     RW          READ BYTE-FLAG INTO S1
            T     IOR   S1

       A    RRS   IOR   M     RW          READ A WORD INTO B AND FORM
            CM    IOR   B     211         AN OTA INSTRUCTION
       B    CL    IOR   S3    205
            T     IOR   B

            S1    IOR               TBZ   TEST BYTE-FLAG
                  JMP   S1    LOW         (LOW BYTE)
                  NOR   S1                SET BYTE-FLAG 'LOW'

            CM    IOR   IR    27          SHIFT B RIGHT EIGHT POSITIONS
       B          IOR   B     SRG2
       B          IOR   B     SRG2

  LOW   A          INC   A                 INCREMENT CHAR-ADDRESS
        B    CM    AND   B     377         CLEAR THE HIGH BYTE FROM B
```

(continued)

Figure 6-5. Continuator Section

2) The microprogram reads the contents of the specified core memory location into the B-register and isolates the character to be printed in the low byte position of the B-register:

   - If a high byte character is to be printed, the microprogram shifts the character into the low byte position and then clears the high byte from the B-register.

```
OTA         S3    IOR   IR    IOG1        EXECUTE THE OTA INSTRUCTION
         B  RRS   IOR
         B  RRS   IOR   IOO
         B  RRS   IOR   IOO
            CR    IOR   B     311         FORM AN STC,C INSTRUCTION
         B  CL    IOR   S3    207
STC,C       S3    IOR   IR    IOG1        EXECUTE THE STC,C INSTRUCTION
         Q        INC   Q                 INCREMENT CHAR=COUNT
STORE       CR    IOR   B     23          STORE CHAR=COUNT IN LOC 23
         B  RRS   IOR   M     CW    UNC
                  IOR
         Q  RRS   IOR   T
         B        INC   B                 STORE CHAR=ADDRESS IN LOC 24
         B  RRS   IOR   M     CW    UNC
                  IOR
         A  RRS   IOR   T
         B        INC   B                 STORE BYTE=FLAG IN LOC 25
         B  RRS   IOR   M     CW    UNC
                  IOR
            S1    IOR   T
            CR    IOR   S1    20          RESTORE A FROM LOC 20
            S1    IOR   M     RW
            S1    INC   S2
            T     IOR   A
            S2    IOR   M     RW          RESTORE B FROM LOC 21
            S2    INC   S1
            T     IOR   B
            S1    IOR   M     RW          RESTORE OVF=FF & E FROM LOC 22
                  CLO
            T     IOR   Q
         Q        LWF   Q     M1
         Q        RFE         RSS   ODD
                  SOV               EOP
                  IOR                     EXIT
CLC         CR    IOR   B     311         FORM A CLC INSTRUCTION
         B  CL    IOR   S3    215         AND EXECUTE IT
            S3    IOR   IR    IOG1
                  JMP         STORE       JUMP TO STORE
```

Figure 6-5. Continuator Section (continued)

- If a low byte character is to be printed, the microprogram merely clears the high byte from the B-register.

3) The microprogram forms an OTA instruction, puts it in the CPU Instruction Register, and causes the I/O decoders to decode and execute the instruction. This results in the character in the B-register being sent to the teleprinter.

4) The microprogram forms an STC,C instruction, puts it in the CPU Instruction Register, and causes the I/O decoders to decode and execute the instruction.

5) Control returns to the interrupted program.

INITIATOR SECTION COMMENTARY

INIT	NOP	Initiator section entry point.
	STF 0	Turn on the interrupt system.
	LDA CHCNT STA COUNT	Pass the character-count constant to the low core equipment table.
	LDA DBUFF STA ADRES	Pass the character-address to the low core equipment table.
	CLA STA BFLAG	Set BFLAG in the low core equipment table to zero.
	LDA MODE OTA TTY	Specify that the teleprinter is to be used as an output device.
	OCT 105000	Transfer control to the Continuator Section.
	JMP INIT,I	Initiator section exit.

CONTINUATOR SECTION COMMENTARY

-	CR	IOR	S1	20		Save the A-register in core memory location 20 of the low core equipment table.
-	S1	IOR	M	CW	UNC	
-	-	IOR	-	-	-	
A	RRS	IOR	T	-	-	

```
- S1   INC   S2   -     -
- S2   IOR   M    CW    UNC
- -    IOR   -    -     -
B RRS IOR   T    -     -
```

Save the B-register in core memory location 21 of the low core equipment table.

```
- S2   INC   S1   -     -
- -    RFE   Q    RSS   OVF
Q -    INC   Q    -     -
Q -    LWF   Q    L1    -
- S1   IOR   M    CW    UNC
- -    IOR   -    -     -
Q RRS IOR   T    -     -
```

Read the contents of the Overflow and Extend flip-flops (0,0 or 0,1) into bit positions 1 and 0, respectively, of the Q-register. Save the Q-register in core memory location 22 of the low core equipment table.

```
- S1   INC   S2   -     -
- S2   IOR   M    RW    -
- S2   INC   S1   -     -
- T    IOR   Q    RSS   TBZ
- -    JMP   -    CLC
```

Read the character-count from core memory location 23 in the low core equipment table into the Q-register. Test the count. If zero, pass control to CLC. Otherwise, continue.

```
- S1   IOR   M    RW    -
- S1   INC   S2   -     -
- T    IOR   A    -     -
```

Read the character-address from core memory location 24 in the low core equipment table into the A-register.

```
- S2   IOR   M    RW    -
- T    IOR   S1   -     -
```

Read the "high/low byte" flag from core memory location 25 in the low core equipment table into Scratch Pad Register 1.

```
A RRS IOR   M    RW    -
- CR   IOR   B    211
B CL   IOR   S3   205
- T    IOR   B    -     -
```

Get the first word to be printed and store it in the B-register. Form an OTA instruction in Scratch Pad Register 3.

```
- S1   IOR   -    -     TBZ
- -    JMP   S1   LOW
- -    NOR   S1   -     -
```

Test the "high/low byte" flag.
If it is non-zero, pass control to LOW.
If it is zero, set it to non-zero (all ones) and continue.

```
        - CR  IOR  IR  27          Shift the high byte character in
        B -   IOR  B   SRG2  -     the B-register into the low byte
        B -   IOR  B   SRG2  -     position.

LOW     A -   INC  A   -     -     Increment the character-address
        B CR  AND  B   377         (core memory location 24) and
                                   clear the high byte from the
                                   B-register.

OTA     - S3  IOR  IR  IOG1  -     Load the OTA instruction from
        B RRS IOR  -   -     -     Scratch Pad Register 3 into the
        B RRS IOR  IOO -     -     CPU Instruction Register. Cause
        B RRS IOR  IOO -     -     the instruction to be decoded
                                   and executed.

        - CR  IOR  B   311         Form the STC,C instruction in
        B CL  IOR  S3  207         Scratch Pad Register 3.

STC,C   - S3  IOR  IR  IOG1  -     Load the STC,C instruction from
                                   Scratch Pad Register 3 into the
                                   CPU Instruction Register. Cause
                                   the instruction to be decoded
                                   and executed.

        Q -   INC  Q   -     -     Increment the character-count
                                   (remember that the count is in
                                   two's complement form; conse-
                                   quently, it is effectively
                                   decremented).

STORE   - CR  IOR  B   23          Store the character-count in core
        B RRS IOR  M   CW    UNC   memory location 23.
        - -   IOR  -   -     -
        Q RRS IOR  T   -     -

        B -   INC  B   -     -     Store the character-address in
        B RRS IOR  M   CW    UNC   core memory location 24.
        - -   IOR  -   -     -
        A RRS IOR  T   -     -
```

```
          B  -    INC  B   -      -      Store the "high/low byte" flag in
          B  RRS  IOR  M   CW     UNC    core memory location 25.
          -  -    IOR  -   -      -
          -  S1   IOR  T   -      -

          -  CR   IOR  S1  20            Restore the A-register from core
          -  S1   IOR  M   RW     -      memory location 20.
          -  S1   INC  S2  -      -
          -  T    IOR  A   -      -

          -  S2   IOR  M   RW     -      Restore the B-register from core
          -  S2   INC  S1  -      -      memory location 21.
          -  T    IOR  B   -      -

          -  S1   IOR  M   RW     -      Restore the Overflow and Extend
          -  -    CLO  -   -      -      flip-flops from core memory
          -  T    IOR  Q   -      -      location 22.
          Q  -    LWF  Q   R1     -
          Q  -    RFE  -   RSS    ODD
          -  -    SOV  -   -      EOP
          -  -    IOR  -   -      -

CLC       -  CR   IOR  B   311           Form a CLC instruction and load
          B  CL   IOR  S3  215           it into the CPU Instruction
          -  S3   IOR  IR  IOG1   -      Register. Cause the instruction to
          -  -    JMP  -   STORE         be decoded and executed. Then
                                         pass control to STORE.
```

This section is included as a "catch-all" for discussions that may be added in future editions. The discussions in this first edition were prepared by HP instructors for use in the HP 2100 Microprogramming course.

- Interrupting a Microprogram
- A/B Addressable Flip-flops
- RPT Micro-order
- JSB/RSB Micro-orders
- Counter

INTERRUPTING A MICROPROGRAM

Microprogram execution cannot be interrupted by *hardware*; however, the execution time of most microprograms is generally short enough so as not to be detrimental to the interrupt system.

Of course there will be exceptions. For example, a "block move" microprogram could, if the number of bytes or words to be moved is large enough, take several milliseconds to execute.

The CJMP micro-order makes it possible for a microprogram to *detect* an interrupt. If an interrupt request is present, the CJMP executes as a JMP; otherwise it executes as a NOP.

Usually, the section of micro-code that is jumped to upon detection of an interrupt performs the following functions:

1) Saves all address pointers, counters, and flags used by the microprogram.

2) Loads the core memory address of the 105xxx macro instruction into the P-register.

3) Executes an EOP micro-order (this allows the interrupt to occur).

Performance of the above functions assumes that the interrupted microprogram, when re-entered after the interrupt, will continue execution at the logical point where it was interrupted.

An alternate method is to merely load the core memory address of the first location in the calling sequence into the P-register and execute an EOP. In this case, the entire microprogram will be re-executed from the beginning after the interrupt. This is the method used by the Extended Arithmetic Group (EAG) instructions, since their execution times are relatively short.

Refer to Figure 7-1. The CJMP microinstruction passes control to INRUP if an interrupt request is detected. At INRUP, the P-register contains the core memory address of COUNT and is used for saving the current word count in the calling sequence. The P-register is then decremented (so it now points to the 105xxx instruction) and an EOP is executed (setting the PH1B interrupt phase). It is the responsibility of the interrupting program to save the contents of the A- and B-registers.

After the interrupt is serviced, control returns to the microprogram by way of the 105xxx macro instruction. The A- and B-registers were restored by the interrupting program and the word count is restored by the first two microinstructions. The microprogram then continues at the logical point where it was interrupted.

```
*                LDA =(# OF WORDS)
*                STA COUNT
*                LDA 'FROM' ADDRESS
*                LDB 'TO' ADDRESS
*                105XXX
*          COUNT RSS 1
*
*
*
*
MOVE       P     IOR   M    RW              GET THE #-OF-WORDS
           T     IOR   Q
LOOP             CJMP        INWUP          INTERRUPT WAITING ?
       A   RRS   IOR   M    RW              MOVE A WORD FROM CORE TO S1
           T     IOR   S1
       B         IOR   S2                    PUT 'TO' ADDRESS IN S2
*
       F   S2    DEC   M    CW    NMPV       WRITE THE WORD INTO CORE
                 JMP         OUT             (MEMORY PROTECT VIOLATION)
           S1    IOR   T
       A         INC   A                     BUMP THE 'FROM' ADDRESS
       B         INC   B                     BUMP THE 'TO' ADDRESS
       Q         INC   Q          TBZ        BUMP THE COUNT & SKIP IF 0
                 JMP         LOOP            NOT DONE YET
OUT        P     INC   P          EOP        SET P AND THEN EXIT
                 IOR
*
*          UPON INTERRUPT JUMP HERE AND SAVE THE COUNT
*
INWUP      P     IOR   M    CW    UNC        PUT THE COUNT BACK
                 IOR                           IN THE CALLING SEQUENCE
       Q   RRS   IOR   T
           P     SUB   P          EOP        DECREMENT P AND THEN EXIT
           P     NOR   P
```

Figure 7-1. Interrupt Example

Had there been more variables to be saved, additional core locations would be required either in the calling sequence or at some other predesignated area of core memory.

If the microprogram logic requires that a particular group of microinstructions be skipped upon re-entry after an interrupt, an interrupt flag may be used. Normally, the interrupt flag would be off. The interrupt flag would be set on upon interrupt exit. Whenever the microprogram is entered, it would test the state of the interrupt flag. If the flag is on, the particular microinstructions would be skipped; if the flag is off, the microinstructions would be executed.

Another method would be to use two 105xxx macroinstructions in the calling sequence: one that would be used for originally entering the microprogram and another that would be used for re-entering the microprogram after an interrupt.

It is also possible to write FORTRAN-callable microprograms that detect interrupts; however, the FORTRAN-produced assembly language calling sequence requires greater microprogram manipulation.

A/B ADDRESSABLE FLIP-FLOPS

Refer to the following pages in the *2100 Computer Microprogramming Guide:* 1-9, 2-4, 2-16, 2-19, 4-4, 4-6, 4-13 through 4-16, 5-5, and 5-6. These two flip-flops are used primarily to implement Memory Reference Group (MRG) instructions that contain an operand address of 0 or 1 (A-register = 0; B-register = 1).

A microcode (RW or CW) reference to core memory address 0 or 1 will access the *actual* core locations

MEMORY READ

An RW micro-order reads the contents of a core memory location into the T-register and, for core memory addresses 0 or 1, sets the A-Addressable or B-Addressable flip-flop. Both flip-flops are cleared if the core memory address is other than 0 or 1.

The microprogram may test the A-Addressable and B-Addressable flip-flops and, if either is set, use the contents of the appropriate register instead of the contents of the actual core location. The micro-code would be as follows:

```
-    P      IOR M  RW -
AAB COND IOR S1 -   -
```

If the P-register contains 0 or 1, the A-Addressable or B-Addressable flip-flop is set and the AAB reads the contents of the A- or B-register (instead of core locations 0 or 1) onto the R-bus. The COND micro-order reads the contents of the R-bus onto the S-bus.

If the P-register instead contains 300, the A-Addressable and B-Addressable flip-flops are both cleared and the AAB reads zeros onto the R-bus. The COND micro-order reads the contents of the T-register (i.e., the contents of core memory location 300) onto the S-bus.

In the event that the contents of core location 0 or 1 is wanted, the AAB and COND micro-orders should be replaced by a T in the S-bus field (in this case, the A-Addressable and B-Addressable flip-flops are ignored).

```
- P IOR M  RW -
- T IOR S1 -   -
```

MEMORY WRITE

A CW micro-order, if executed, with 0 or 1 in the M-register writes data from the T-register into core locations 0 or 1. CW does *not* enable the setting of the A-Addressable or B-Addressable flip-flops.

```
- S1 IOR M CW UNC (Assume S1 contains 1)
- -  IOR - -   -
- S4 IOR T -   -
```

The above coding would cause the contents of Scratch Pad Register 4 to be written into core memory location 1.

To avoid this, an NMPV micro-order should be used in the Skip field of the CW microinstruction and the B-Addressable flip-flop must be set prior to execution of the CW microinstruction.

```
- S1 IOR -     AAB -      (Assume S1 contains 1)
- S1 IOR M    CW  NMPV
- S4 IOR AAB -     -
- S4 IOR T    -     -
```

The above coding would cause the contents of Scratch Pad Register 4 to be copied into the B-register. The same principle applies to the A-register if S1 contains 0. The "write into memory" operation would *not* be performed. Note that the last microinstruction is always executed; however, when writing to the A- or B-registers, the last microinstruction has no net effect since the "write into memory" operation was not performed.

The above example assumes that the memory protect feature is not enabled.

With memory protected enabled, the NMPV micro-order also traps memory protect violations. The microprogram must decide which event caused the skip. If memory protect is enabled, the coding could be as follows:

```
Line 1  -  S2  IOR -     AAB  -
Line 2  F  S2  DEC M     CW   NMPV
Line 3  Q  -   IOR AAB   RSS  AAB
Line 4  Q  RRS IOR T     -    UNC
Line 5  -  -   JMP -     VILAT
```

(Assume that the data to be written is in the Q-register)

Line 1 sets either the A-Addressable or B-Addressable flip-flop, or clears them both. The clearing is necessary since a previous reference to core address 0 or 1 could have occurred.

Line 2 skips to Line 4 if no memory protect violation occurs and if both the A-Addressable and B-Addressable flip-flops are clear. In Line 4 the contents of the Q-register are loaded into the T-register, the "write into memory" operation is performed, and the JMP VILAT microinstruction is skipped.

If either the A-Addressable or B-Addressable flip-flop is set, control passes from Line 2 to Line 3. Line 3 copies the contents of the Q-register into the A- or B-register and passes control to Line 4. Line 4 has no net effect (other than to cause Line 5 to be skipped) because the "write into memory" operation was not performed.

If a memory protect violation occurs, control passes from Line 2 to Line 3. Since both the A- and B-Addressable flip-flops are clear, Line 3 skips to Line 5 which passes control to VILAT. Since the A- and B-Addressable flip-flops are clear, the AAB micro-order in the Store field of Line 3 is not enabled.

Note that the above sequence assumes that memory-protect violations are to be handled by microcode. If memory protect violations are to be handled by software (the usual case), the RSS and AAB micro-orders in Line 3, plus all of Line 5, are unnecessary. In such a case, when an EOP is detected the hardware automatically enters phase 1B because the memory protect violation flip-flop is set. This causes execution of the contents of core memory location 5.

The bypassing of core locations 0 and 1 is necessary for implementing the Memory Reference Group (MRG) instructions. In most user microprograms, however, memory references to locations 0 and 1 will not be made and the above discussions may be disregarded.

RPT MICRO-ORDER

Refer to page 4-16 of the *2100 Computer Microprogramming Guide*.

The RPT micro-order causes the next sequential microinstruction to be executed repeatedly until its skip condition is met. RPT is used primarily in conjunction with a CTRI micro-order to shift a single or double word data item up to 16_{10} times.

Example: Rotate the combined B- and A-registers right 8 positions.

-	CR	IOR	S1	10	Load 10 (octal) into S1.
-	S1	IOR	-	CNTR RPT	Load the counter from S1 and set the repeat mode.
B	-	CRS	B	R1 CTRI	Rotate the B- and A-registers and increment the counter. Repeat until counter = 17_8.
-	S3	IOR	S4	- -	This microinstruction is *not* skipped. The CTRI in the previous microinstruction merely breaks the repeat loop.

RPT may be used in conjunction with skip micro-orders other than CTRI (e.g., NEG, ODD, etc.).

If the skip condition is never met, the repeat loop will be executed continually until the power is turned off (i.e., infinite loop).

JSB/RSB MICRO-ORDERS

Refer to the following pages of the *2100 Computer Microprogramming Guide*: 2-2, 2-3, 2-18, and 4-10.

During microprogram execution, the Save Register copies the ROM Address Register (RAR) until a JSB micro-order is executed. A JSB micro-order sets the JSB flip-flop, isolating the Save Register from the RAR and thereby preserving the return address.

The RSB micro-order is used for returning control from a subroutine to the calling program. Execution of an RSB micro-order resets the JSB flip-flop, causing the contents of the Save Register to be loaded into the RAR.

Execution of an RSB micro-order without prior execution of a JSB micro-order is interpreted as a two-cycle NOP. This allows a subroutine to also be executed as a main line program.

The single Save Register limits the use of subroutines to one level (i.e., a subroutine cannot, in turn, call another subroutine).

COUNTER

Refer to the following pages of the *2100 Computer Microprogramming Guide*: 2-9, 4-15, and 4-16.

The counter was designed primarily for implementation of the Extended Arithmetic Group (EAG) instructions that require a maximum shift count of 16_{10}.

However, the microprogrammer may also use the counter's full capacity of 32_{10}.

Example:

```
      - -      IOR - CNTR -        Set the counter to zero.
LOOP             :
                 :
                 :                  (Repeated microinstructions)
                 :
                 :
      - -      IOR - -     ICTR    Increment the counter.
      - CNTR   IOR - -     TBZ     Break out of the loop when
      - -      JMP - LOOP            counter = 0.
           (continue)
```

Incrementing the counter when it contains 37_8 (maximum capacity) causes it to "roll over" to zero.

The CNTR micro-order in the S-bus field reads all five bits of the counter onto the S-bus.

All error messages are presented in table 8-1.

Note: Warning messages are flagged by ** in the left margin.

Table 8-1. Error Messages

#	Meaning	Corrective Action
1	Duplicate label.	The statement label of the specified microinstruction is the same as another statement label in the microprogram or the same as a declared external symbol. Assign a new statement label and reassemble.
2	Bad control statement.	The specified assembler control statement is illegal. Correct it and reassemble.
3	Illegal RBUS micro-order.	The micro-order in the R-bus field of the specified microinstruction is illegal. Correct it and reassemble.
4	Illegal SBUS micro-order.	The micro-order in the S-bus field of the specified microinstruction is illegal. Correct it and reassemble.
5	Illegal FUNCTION micro-order.	The micro-order in the Function field of the specified microinstruction is illegal. Correct it and reassemble.

Table 8-1. Error Messages (continued)

#	Meaning	Corrective Action
6	Illegal STORE micro-order.	The micro-order in the Store field of the specified microinstruction is illegal. Correct it and reassemble.
7	Illegal SPECIAL micro-order.	The micro-order in the Special field of the specified microinstruction is illegal. Correct it and reassemble.
8	Illegal SKIP micro-order.	The micro-order in the Skip field of the specified microinstruction is illegal. Correct it and reassemble.
9	Illegal jump address.	The asterisk jump address (* ± x) in the specified microinstruction lies outside the bounds of the current control store module or the symbolic jump address in the specified microinstruction is undefined. Correct it and reassemble.
10	CW in Special field and no skip condition in the Skip field.	This combination will *not* write into core memory. Correct it and and reassemble.
11	Program too large.	The program will occupy more than 256_{10} (400_8) control store locations. The microprogram must either be rewritten or assembled in smaller parts.
**12	Warning! CQ detected in the R-bus field or RFI detected in the Function field.	These micro-orders affect the operation of control store module #0. BE CAREFUL!

Table 8-1. Error Messages (continued)

#	Meaning	Corrective Action
**13	Warning! NOP in the R-bus field with DEC in the Function field.	This combination results in the complement of the contents of the S-bus. The use of a NOR micro-order in the Function field is suggested.
14	SBUS is incompatible with the jump address.	JMP, CJMP, and JSB use the low-order bit of the S-bus field as part of the jump address. The micro-order in the S-bus field of the specified microinstruction cannot be used in conjunction with that particular jump address. Correct the S-bus field micro-order and reassemble.
15	SBUS is incompatible with STORE micro-order.	The same Scratch Pad Register cannot be referenced in both the S-bus and Store fields of a microinstruction. Correct the specified microinstruction and reassemble.
**16	Warning! LEP detected in the Special field.	This micro-order cannot be used for anything other than enabling entry points to the 2100 Extended Arithmetic Group.
**17	Warning! Potential memory access problem detected.	An M was detected in the Store field of the specified microinstruction and something other than RW or CW was detected in the Special field. Memory access should be started in the specified microinstruction since the M-register could be modified by a DMA transfer.

Table 8-1. Error Messages (continued)

#	Meaning	Corrective Action
**18	Warning! Potential phasing problem	An EOP was detected in the previous microinstruction and the Skip field of the specified microinstruction contains something other than a NOP. Be aware that the specified microinstruction will be executed *before* the EOP is executed.
19	Repeat on non-skip condition.	An RPT was detected in the Skip field of the previous microinstruction but the Skip field of the specified microinstruction does not contain a "skip" micro-order. Correct the microinstruction and reassemble.
20	Repeat until NEG or ODD with an add-type micro-order in the Function field or repeat until TBZ or RSS, TBZ.	An RPT was detected in the Skip field of the previous microinstruction but the specified microinstruction contains NEG or ODD in the Skip field with an add-type micro-order (ADD, ADDO, INC, or INCO) in the Function field or the specified microinstruction contains TBZ in the Skip field. These combinations are illegal. Correct the microinstruction and reassemble.
**21	Warning! RFE in the Function field with a non-NOP in the S-bus field.	In addition to exchanging the Extend and Flag flip-flops, RFE causes the contents of the R-bus to be read onto the T-bus (the S-bus is ignored).

Table 8-1. Error Messages (continued)

#	Meaning	Corrective Action
**22	Warning! P in the S-bus field acts as a NOP when a JMP is in the Function field.	Leave S-bus field blank instead.

The HP Micro Debug Editor is a program that makes it possible for the user to load object microprograms from an HP Microassembler output tape into a Writable Control Store (WCS) module, debug microcode, or produce a set of six mask paper tapes that can be used for "burning" a set of programmable ROM chips.

REQUIREMENTS

The editor is designed to operate in an 8K Basic Control System (BCS) environment and requires a system console device (either a teleprinter or a CRT terminal). In addition, a paper tape photoreader is required if the input is to be in the form of an object tape and a paper tape punch is required if the output is to be in the form of mask tapes. For most purposes, a WCS module is also required.

MODES OF OPERATION

The editor operates in two modes: the *normal mode* and the *debug mode*. The mode of operation is determined by the presence (debug mode) or absence (normal mode) of the $DEBUG control statement during the assembling of the microprogram. Though their capabilities overlap considerably, the two modes are treated separately in the following discussion.

NORMAL MODE

The normal mode was designed for two purposes: to transfer object microcode from an HP Microassembler output tape to a WCS module and to punch mask tapes from an HP Microassembler output tape.

DEBUG MODE

The debug mode was designed to allow the user to employ breakpoints to debug microprograms. In this mode the user can insert a breakpoint in the buffer, load the microprogram from the buffer into a WCS module, and then execute the microprogram. When the breakpoint is encountered, execution halts and the editor displays the contents of the machine registers and flip-flops on the system console device. The user may then alter the microprogram, alter the contents of the registers, and/or set another breakpoint.

Specifically, in the debug mode the user can:

- Read a microprogram from an object tape into a core memory buffer.
- Set a breakpoint in the buffer.
- Write a microprogram from the buffer into a WCS module.
- Execute a WCS-resident microprogram.
- Display the contents of any buffer location on the system console device.
- Alter the contents of any buffer location.
- Alter the contents of any or all of the machine registers.

The user can also read a microprogram from a WCS module into the core memory buffer or punch an updated object tape from the contents

of the buffer. However, these are considered to be secondary capabilities and are of marginal practical value to most users.

To run the editor in the debug mode, the user must previously have loaded an initialization program, named TEST (see "The Initialization Program" later in this section). Briefly, however, it is used at the start of debug execution to pass parameters and control to the microprogram.

HP MICRO DEBUG EDITOR COMMANDS

When the editor is executed, it prints COMMAND? on the system console device. The user responds to entering one of the input, edit, output, or debug commands described later in this manual. After the editor has performed the specified operation, it again prints COMMAND? on the system console device, etc. To terminate a Micro Debug Editor run, the user enters FINISH in response to the COMMAND? message.

There are twelve Micro Debug Editor commands. They are shown in Table 9-1. In all cases, the first character of the mnemonic is sufficient to identify the command to the editor (for example, to terminate a Micro Debug Editor run, the user may enter F, FI, FIN, FINI, FINIS, or FINISH).

INPUT COMMANDS

The input commands are:

LOAD[,x]
READ,x

Table 9-1. Micro Debug Editor Commands

Input		
Commands:	LOAD[,x]	The brackets indicate that the parameter may be omitted.
	READ,x	
Edit		
Commands:	SHOW,xxxx[,yyyy]	
	MODIFY,xxxx[,yyyy]	
Output		
Commands:	DUMP[,x]	
	WRITE,x	
	PREPARE[,x]	These commands are available only in the normal mode.
	VERIFY[,x]	
Termination		
Command:	FINISH	
Debug		
Commands:	BREAK,yyyy	These commands are available only in the debug mode.
	CHANGE[,mnemonic]	
	EXECUTE[,0 or yyyy]	

LOAD[,x]

x is the logical unit number of a paper tape photoreader. If omitted, x is assumed to be 5.

The LOAD command reads the contents of an HP Microassembler output tape into core memory through the specified device.

READ,x

x is the logical unit number of a WCS module.

The READ command reads the contents of the specified WCS module into core memory.

EDIT COMMANDS

The edit commands are:

```
SHOW,xxxx[,yyyy]
MODIFY,xxxx[,yyyy]
```

SHOW,xxxx[,yyyy]

xxxx and yyyy are control store addresses (0-1777, octal). xxxx is the address of the first location to be displayed and yyyy is the address of the final location to be displayed. If omitted, yyyy is assumed to be the same as xxxx. If the user enters fewer than four digits for xxxx or yyyy, the value entered is right-justified with zeros automatically filled to the left. Note that the editor uses only the rightmost eight bits of xxxx and yyyy (0-377, octal).

The SHOW command displays the specified core memory buffer location(s) on the system console device. The display format is as follows:

aaaa mmm nnnnnn

where aaaa is the control store address (0-1777, octal) of the location being displayed, mmm is the octal representation of bits 24-16 of the location, and nnnnnn is the octal representation of bits 15-0 of the location.

MODIFY,xxxx[,yyyy]

xxxx and yyyy are control store addresses (0-1777, octal). xxxx is the address of the first location to be modified and yyyy is the address of the final location to be modified. If omitted, yyyy is assumed to be the same as xxxx. If the user enters fewer than four digits for xxxx or yyyy, the value entered is right-justified with zeros automatically filled to the left. Note that the editor uses only the rightmost eight bits of xxxx and yyyy (0-377, octal).

The MOFIFY command allows the user to change the contents of the specified core memory buffer location(s).

In response to the MODIFY command, the Micro Debug Editor prints the following on the system console device:

aaaa mmm nnnnnn

where aaaa is the control store address (0-1777, octal) of the location being altered, mmm is the octal representation of the current state of bits 24-16 of the location, and nnnnnn is the octal representation of the current state of bits 15-0 of the location.

The user then enters:

mmm,nnnnnn

where mmm is the octal representation of the desired state of bits 24-16 of the location and nnnnnn is the desired state of bits 0-15 of the location. If the user enters fewer than three digits for mmm or fewer than six digits for nnnnnn, the number entered is right-justified with zeros automatically filled to the left. If it is desired to leave mmm or nnnnnn unchanged, the user may enter an asterisk instead of an octal number.

Examples:

6,123 is equivalent to entering 006,000123.

*,123456 means that bits 24-16 of the location are not to be modified and bits 0-15 are to be set to the value 123456_8.

123,* means that bits 24-16 of the location are to be set to the value 123_8 and bits 0-15 are not to be modified.

If the user specifies that a series of locations are to be altered, the Micro Debug Editor responds by printing aaaa mmm nnnnnn on the system console device, etc. If the user does not wish to alter the contents of a particular location in the series, he enters *,* instead of mmm,nnnnnn.

After the last specified location has been altered, the Micro Debug Editor prints COMMAND? on the system console device.

Note that the MODIFY command and the associated entries alter the specified core memory locations (not the actual WCS locations). To update the WCS module to the revised state, the user must write the contents of the core memory buffer into the WCS module (using the WRITE command).

OUTPUT COMMANDS

The output commands are:

```
DUMP[,x]
WRITE,x
PREPARE[,x]
VERIFY[,x]
```

DUMP[,x]

x is the logical unit number of a paper tape punch device. If omitted, x is assumed to be 4.

The DUMP command punches the contents of the core memory buffer on the specified device. The tape thus produced is in the same format as the output tape produced by the HP Micro-assembler.

WRITE,x

x is the logical unit number of a WCS module.

The WRITE command copies the contents of the core memory buffer into the specified WCS module.

PREPARE[,x]

x is the logical unit number of a paper tape punch device. If omitted, x is assumed to be 4.

The PREPARE command punches a set of six mask tapes on the specified device from the contents of the core memory buffer. Before punching each tape, the editor asks the user to enter the tape's I.D. header information. The user may then enter up to three lines of information (any characters). For tapes two through six, the user has the option of duplicating the I.D. lines used on the previous tape.

VERIFY[,x]

x is the logical unit number of a paper tape photoreader. If omitted, x is assumed to be 5.

The VERIFY command reads a mask tape through the specified device and compares the contents of the tape against the contents of the core memory buffer. If no errors are detected, the editor asks for the next command. If errors are detected, the editor prints

BAD MASK TAPE
DO YOU WANT TO REPUNCH THIS TAPE?

The user responds by entering Y or N. If the user enters N, the editor asks for the next command. If the user enters Y, the editor prints

ENTER PUNCH LOGICAL UNIT #

and the user enters the logical unit number of the paper tape punch device. The editor then asks the user to enter three lines of tape I.D. information, repunches the tape, and asks for the next command.

The mask tapes may be verified in any order. To verify an entire set of mask tapes, the user must enter the VERIFY command a total of six times (assuming that none of the tapes has to be repunched and reverified).

TERMINATION COMMAND

The termination command is:

FINISH

FINISH

The FINISH command terminates the current Micro Debug Editor run.

DEBUG COMMANDS

The debug commands are:

BREAK,yyy
CHANGE[,mnemonic]
EXECUTE[,0 or yyy]

BREAK,yyyy

yyyy is a control store address (0-1777, octal). If the user enters fewer than four digits for yyyy, the value entered is right-justified with zeros automatically filled to the left. Note that the editor uses only the rightmost eight bits of yyyy (0-377, octal).

The BREAK command sets a breakpoint at the specified location in the core memory buffer. The microprogram should then be written from the buffer into a WCS module and executed. When the breakpoint is encountered during debug execution, execution halts, the contents of the machine registers (A, B, Q, F, P, S1, S2, S3 S4) and flip-flops (Flag, Overflow, Extend) are displayed on the system console device, and the breakpoint is removed from the buffer.

Breakpoints should be set only where a JMP microinstruction is allowed. For example, a breakpoint should not be set immediately following a microinstruction that contains either an EOP or RPT micro-order. However, this responsibility is left entirely up to the user.

CHANGE[,mnemonic]

mnemonic is one of the following mnemonics:

A	(A-register)
B	(B-register)
Q	(Q-register)
F	(F-register)
P	(P-register)
S1	(Scratch Pad Register 1)
S2	(Scratch Pad Register 2)
S3	(Scratch Pad Register 3)
S4	(Scratch Pad Register 4)

O (Overflow flip-flop)
E (Extend flip-flop)
FLAG (Flag flip-flop)

The CHANGE command is used for altering the contents of any or all of the registers and flip-flops.

If the user omits mnemonic, the editor assumes that he wishes to alter the contents of all the registers and flipflops.

If the user specifies a mnemonic, the editor responds by printing

mnemonic xxxxxx <=

on the system console device, where xxxxxx is the octal representation of the current contents of the register or flip-flop.

The user then responds by entering an octal number representing the desired contents of the register.

If the user enters a CHANGE command with no mnemonic, the editor assumes that he wishes to alter the contents of all the registers and flip-flops. In this case, the above conversational process is done for each register and flip-flop. If the user does not wish to alter the contents of a particular register or flip-flop, he enters an asterisk (*) instead of the octal number.

EXECUTE[,0 or yyyy]

yyyy is a control store address (0-1777, octal). If the user enters fewer than four digits for yyyy, the value entered is right-justified with zeros automatically filled to the left. Note that the editor uses only the rightmost eight bits of yyyy (0-377, octal).

The EXECUTE command causes the contents of the core memory buffer to be written to a WCS module and then executes the WCS-

resident program. If the user has previously used a WRITE command, the EXECUTE statement automatically uses the same WCS module referenced by the WRITE command. If the user had not previously used a WRITE command, the editor first responds to the EXECUTE command by asking for the logical unit number of the WCS module.

EXECUTE,0 causes the WCS-resident microprogram to be executed from the beginning by way of the user's initialization program.

EXECUTE causes the WCS-resident microprogram to be executed from the point where it was last interrupted. This is used for restarting execution after a breakpoint has been encountered.

EXECUTE,yyyy causes the WCS-resident microprogram to be executed starting at the specified control store address. Note that the editor uses only the rightmost eight bits of yyyy (0-377, octal).

THE INITIALIZATION PROGRAM

When the Micro Debug Editor is to be run in the debug mode, the user must supply an initialization program. The initialization program is an assembly language program that performs whatever functions are necessary to call the microprogram (namely, preparing the necessary parameters in core memory and then executing a 105xxx macro instruction).

The name of the initialization program must be TEST. The program must also have the symbol MACRO as a declared entry point, where MACRO is the symbolic address of the 105xxx macro instruction. There should only be one 105xxx macro instruction in the initialization program. Table 9-2 shows the structure of an initialization program.

Table 9-2. Initialization Program

```
ASMB,R,B,L,T
        NAM TEST
        ENT  TEST, MACRO
TEST    NOP
          .
          .
          .
MACRO   OCT 105xxx
        DEF P1
        DEF P2
          .
          .
          .
        DEF Px
        JMP  TEST,I
P1      (constant definition statement)
P2      (constant definition statement)
          .
          .
          .
Px      (constant definition statement)
        END
```

Operating in the debug mode imposes the following four restrictions on the microcode that is being debugged. The first microinstruction must be a JMP to the start of the microprogram (i.e., the first microinstruction must be a primary jump table entry). The microcode being debugged must be less than 186_{10} locations in length. The microcode cannot access core memory location 0. The user cannot set a breakpoint in a subroutine.

LOADING THE MICRO DEBUG EDITOR

Refer to the *Basic Control System* manual (02116-9017).

1. Load the Basic Control System (BCS) using the Basic Binary Loader.

2. Load the HP Micro Debug Editor using the BCS Relocating Loader.

3. If the editor is to be run in the debug mode, load the initialization program (TEST) using the BCS Relocating Loader.

4. Load the Relocatable Library using the BCS Relocating Loader. If the editor is to be run in the normal mode, the user must force program execution at this point even though there is an undefined external symbol (TEST). This is done by entering 010 into switches 2-0 of the Switch Register.

5. Press the RUN switch. BCS responds by printing the loading map on the system printer device.

6. Press the RUN switch. The editor responds by typing COMMAND? on the system console device.

DEBUGGING A SMALL MICROPROGRAM

These operating instructions apply when the microprogram being debugged is smaller than 186_{10} locations in length. The appropriate Micro Debug Editor command mnemonic is shown in parentheses whenever the associated command is used.

1. Assemble the microprogram using the debug option.

2. Load the Micro Debug Editor and the initialization program.

3. Read the Microassembler output tape into core memory (LOAD).

4. Set a breakpoint (BREAK).

5. Enter an EXECUTE,0 command. This loads the contents of the core memory buffer into the WCS module (the editor will ask for the module's logical unit number) and then causes the initialization program to be executed. The initialization program, in turn, passes control to the microprogram. When the breakpoint is encountered, execution halts, the breakpoint is removed from the core memory buffer, and the contents of the machine registers and flip-flops are displayed on the system console device.

6. Enter any Micro Debug Editor commands.

   Usually at this point the user performs conversational editing (SHOW, MODIFY) and/or alters the contents of any or all of the registers and flip-flops (CHANGE). However, this is also the logical point at which one would terminate the entire Micro Debug Editor run (FINISH).

7. Set another breakpoint (BREAK).

8. Restart execution (EXECUTE or EXECUTE,yyy or EXECUTE,0).

   - EXECUTE restarts execution from the point where it was interrupted.
   - EXECUTE,yyy restarts execution from the specified WCS relative address.
   - EXECUTE,0 restarts execution from the beginning (by way of the initialization program).

9. When the breakpoint is encountered, repeat steps 6-8, above.

DEBUGGING A LARGE MICROPROGRAM

These operating instructions apply when the microprogram being debugged is larger than 186_{10} locations in length. The appropriate Micro Debug Editor command mnemonic is shown in parentheses whenever the associated command is used.

1. Break the microprogram into two or more segments in such a way that each segment is smaller than 186_{10} locations in length. Each segment must be able to be entered by using the same 105xxx macro instruction.

2. Assemble each segment separately using the debug option.

3. Load the Micro Debug Editor and the initialization program.

Segment #1

4. Read the Microassembler output tape for the segment into core memory (LOAD).

5. Set a breakpoint (BREAK).

6. Enter an EXECUTE,0 command. This causes the initialization program to be executed. The initialization program, in turn, passes control to the microprogram segment. When the breakpoint is encountered, execution halts, the breakpoint is removed from the core memory buffer, and the contents of the machine registers and flip-flops are displayed on the system console device.

7. Enter any Micro Debug Editor commands.

Usually at this point the user performs conversational editing (SHOW, MODIFY) and/or alters the contents of any or all of the registers and flip-flops (CHANGE). However, this is also the logical point at which one would initiate the debugging of Segment #2 (step 11) or terminate the entire Micro Debug Editor run (FINISH).

8. Set another breakpoint (BREAK).

9. Restart execution (EXECUTE or EXECUTE,yyy or EXECUTE,0).

   - EXECUTE restarts execution from the point where it was interrupted.
   - EXECUTE,yyy restarts execution from the specified WCS relative address.
   - EXECUTE,0 restarts execution from the beginning (by way of the initialization program).

10. When the breakpoint is encountered, repeat steps 7-9, above.

Segments #2 Through x

11. Read the Microassembler output tape for the segment into core memory (LOAD).

12. Set a breakpoint (BREAK).

13. Enter an EXECUTE,yyy command, where yyy is the WCS relative address of the first microinstruction to be executed. When the breakpoint is encountered, execution halts, the breakpoint is removed from the core memory buffer, and the contents of all the machine registers and flip-flops are displayed on the system console device.

14. Enter any Micro Debug Editor command.

    Usually at this point the user performs conversational editing (SHOW, MODIFY) and/or alters the contents of any or all of the registers and flip-flops (CHANGE). However, this is also the logical point at which one would initiate the debugging of the next segment (step 11) or terminate the entire Micro Debug Editor run (FINISH).

15. Set another breakpoint (BREAK).

16. Restart execution (EXECUTE or EXECUTE,yyy).

    - EXECUTE restarts execution from the point where it was interrupted.

    - EXECUTE,yyy restarts execution from the specified WCS relative address.

17. When the breakpoint is encountered, repeat steps 14-16, above.

PUNCHING MASK TAPES FROM AN OBJECT TAPE

1. Assemble the microprogram.

2. Load the Micro Debug Editor using the BCS Relocating Loader.

3. Read the Microassembler output tape into core memory (LOAD).

4. Punch the mask tapes (PREPARE).

5. Verify each mask tape, as follows:

a. Load the mask tape into the paper tape photoreader.

b. Enter a VERIFY command.

c. If the tape contains no errors, load the next tape in the photoreader and enter another VERIFY command, etc.

   If the tape contains errors, the editor prints a message to that effect on the system console device and allows the user to repunch the erroneous tape.

6. If all the mask tapes contain no errors, terminate the run (FINISH).

LOADING A MICROPROGRAM INTO WCS FROM AN OBJECT TAPE

1. Assemble the microprogram.

2. Load the Micro Debug Editor using the BCS Relocating Loader.

3. Read the Microassembler output tape into core memory (LOAD).

4. Write the microprogram into a WCS module (WRITE).

5. Terminate the Micro Debug Editor run (FINISH).

The HP Programmable ROM Writer is a program that uses the mask tapes produced by the HP Micro Debug Editor to permanently burn microcode into programmable ROM chips.

REQUIREMENTS

The HP Programmable ROM Writer is designed to operate in an 8K Basic Control System (BCS) environment and requires a system console device (either a teleprinter or a CRT console), a paper tape photo-reader, and an HP 12909A Programmable ROM Writer.

INITIAL PARAMETERS

When loaded, the Programmable ROM Writer prints

PROM WRITER CONTROL PROGRAM

on the system console device and then asks for a series of parameters, as follows:

```
ENTER PROM BURN PARAMETERS
CHIP INITIAL STATE (0 or 1)?
```

The user enters either a zero or a one, depending upon whether the chip initially contains all zeros or all ones.

MINIMUM BURN TIME (MILLISECONDS)?

The user enters a positive decimal integer specifying the length of time (in milliseconds) that each chip location is to be burned on the first attempt.

MAXIMUM BURN TIME (MILLISECONDS)?

The user enters a positive decimal integer specifying the length of time (in milliseconds) that each chip location is to be burned on the final retry.

MAXIMUM NUMBER OF RETRIES?

The user enters a positive decimal integer specifying the maximum number of times that the burning of a chip is to be retried. The initial burn attempt is performed using the minimum burn time. If retries are necessary, each is performed using a proportionately longer burn time. If the specified maximum number of retries are necessary, the final retry is performed using the maximum burn time. For example, if the user specifies a minimum burn time of 1 millisecond, a maximum burn time of 11 milliseconds, and a maximum number of retries of 5, the burn times of the various burn attempts is as follows:

Initial burn attempt:	1 ms
1st retry:	3 ms
2nd retry:	5 ms
3rd retry:	7 ms
4th retry:	9 ms
5th retry:	11 ms

WAIT TIME RATIO?

The user enters a positive decimal integer that determines the amount of "wait time" between successive burn passes, as follows:

"wait time" = RATIO • current burn time

For example, if the current burn time is 100 milliseconds (a tenth of a second) and the wait time ratio is 5, the program allows 500 milliseconds (half a second) between successive burn passes.

The information necessary for entering the above parameters is available in the documentation provided by the programmable ROM chip manufacturer.

GENERAL OPERATION

After the initial parameters have been entered, the program prints COMMAND? on the system console device. The user responds by entering one of the commands shown in Table 10-1.

If the user enters an illegal command, the program prints INPUT ERROR on the system console device and requests another command. In all cases, the first two characters are sufficient for the program to recognize the command.

After each command (except STOP) is executed, the program requests another command by printing COMMAND? on the system console device.

In the following discussions, the overall process of burning a programmable ROM chip is divided into two processes: set-up and burning. The processes are performed sequentially and in that order for every chip that is to be burned.

Table 10-1. Commands

COMMAND	EFFECT
LOAD	Causes the program to read a mask tape into core memory and print the tape identity information on the system console device.
VTAPE	Causes the program to verify the contents of the mask tape by computing a checksum and comparing it against a checksum contained on the tape.
VCHIP	Causes the program to test a chip to be certain it contains all zeros or all ones.
BURN	Causes the program to burn a chip.
CREAD	Reads the contents of the chip mounted on the Programmable ROM Writer hardware into the core memory buffer.
STOP	Terminates an HP Programmable ROM Writer run.

SET-UP

The user mounts a programmable ROM chip on the HP 12909A Programmable ROM Writer, loads a mask tape in the paper tape photoreader, and then enters LOAD through the system console device. The program reads the mask tape into a buffer area in core memory and prints the tape identity information on the system console device. The user should examine the printed identity information to be certain that the proper tape has been loaded.

If the proper tape has been loaded, the user enters VTAPE through the system console device. The program verifies the contents of the tape by computing a checksum and comparing it against a checksum contained on the tape. If it detects an error, the program prints CHECKSUM ERROR on the system console device. In such a case, the user reloads the tape in the photoreader and re-enters the LOAD and VTAPE commands. If the checksum error persists, a new set of mask tapes must be produced using the HP Micro Debug Editor.

Note: If it is desired to duplicate a programmable ROM chip that is already burned, use the following set-up procedure instead of the above:

1) Mount the burned chip on the HP 12909A.

2) Enter a CREAD command.

3) Remove the burned chip from the HP 12909A.

4) Mount a new (unused) chip on the HP 12909A.

The user enters VCHIP through the system console device. The program tests the chip to be certain that it contains all zeros or all ones (as specified in the initial parameters). If the chip does not contain all zeros or all ones, the program prints BAD CHIP on the system console device. In such a case, the user discards the chip, mounts a new one, and re-enters the VCHIP command.

BURNING

The user enters BURN through the system console device. The program burns the chip using the minimum burn time. After the chip has been burned, the program reads the chip locations to see if they were burned properly. If any chip location was not burned properly during the first burn pass, a second pass is made using a longer burn time, etc. During the retry passes, only the erroneous chip locations are reburned. The

number of retry passes and the burn times are determined by the initial parameters entered by the user. If the chip still contains errors after the final burn retry, the program prints the following message on the system console device for each erroneous chip location:

ERROR AT chip-location CHIP = xxxx BUFFER = yyyy

where chip-location is an octal number (000-377) specifying what chip location is in error.

xxxx is a four-digit binary number showing the current state of the chip location.

yyyy is a four-digit binary number showing the current state of the associated core memory locations.

The user then enters BURN or STOP. BURN causes the program to burn the entire chip as described above. STOP terminates the Programmable ROM Writer run.

This is a library routine which makes it possible for FORTRAN and ALGOL programs to move object microcode from core memory to a Writable Control Store (WCS) module or from a WCS module to a core memory buffer. The routine is designed to operate in a minimum Basic Control System (BCS) environment.

CALLING SEQUENCES

In both FORTRAN and ALGOL there are two calling sequences: one for moving object microcode from a core memory buffer to a WCS module and one for moving object microcode from a WCS module to a core memory buffer.

CORE MEMORY TO WCS MODULE

The FORTRAN calling sequence for moving object microcode from a core memory buffer to a WCS module is:

CALL WWRIT (module,buffer-name,#-of-words)

where *module* is a decimal number specifying the unit reference number of the WCS module.

buffer-name is the array name of the core memory buffer.

#-of-words is a decimal number specifying the number of words to be moved. If *#-of-words* is positive, it specifies the number of WCS words to be moved; if it is negative, it specifies the number of core memory words to be moved.

Object microcode is stored in core memory such that each WCS word requires two buffer words. Bits 0-7 of the first buffer word of each pair contain three octal digits specifying the WCS location to be written into. Bits 8-15 of the same buffer word contain bits 0-7 of the specified WCS location. Bits 0-15 of the second buffer word of each pair contain bits 8-23 of the specified WCS location. When the object microcode is moved from the core memory buffer to the WCS module, only the specified WCS locations are altered (all other WCS locations are left unchanged).

The ALGOL calling sequence for moving object microcode from a core memory buffer to a WCS module is:

```
PROCEDURE WWRIT (A,B,C);
    INTEGER A,C;   ARRAY B;
        .
        .
        .
WWRIT (module,buffer-name,#-of-words);
```

where *module*,*buffer-name*, and *#-of-words* are described for FORTRAN, above.

WCS MODULE TO CORE MEMORY

The FORTRAN calling sequence for moving object microcode from a WCS module to a core memory buffer is:

```
CALL WREAD (module,buffer-name,#-of-words,wcs-address)
```

where *module* is a decimal number specifying the unit reference number of the WCS module.

buffer-name is the array name of the core memory buffer.

#-of-words is a decimal number specifying the number of words to be moved. If *#-of-words* is positive, it specifies the number of

WCS words to be moved; if it is negative, it specifies the number of core memory words to be read into.

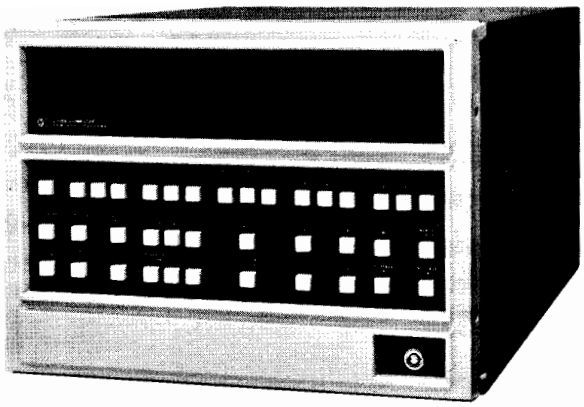
wcs-address is an octal number specifying the starting WCS location of the object microcode to be moved.

Object microcode is read into the core memory buffer in the format described earlier in this section. The WCS word residing at WCS location *wcs-address* is read into the first two buffer words, the WCS word residing at WCS location *wcs-address* + 1 is read into the next two buffer words, and so forth.

The ALGOL calling sequence for moving object microcode from a WCS module to a core memory buffer is:

```
PROCEDURE WREAD (A,B,C,D);
    INTEGER A,C,D;   ARRAY B;
        .
        .
        .
WREAD (module,buffer-name,#-of-words,wcs-address);
```

where *module*, *buffer-name*, *#-of-words*, and *wcs-address* are described for FORTRAN, above.



- **POWERFUL HARDWARE**
  A proven architecture implemented by a microprocessor in the heart of the control section.
- **EXPANDABLE MAINFRAME MEMORY**
  Lets you choose from 4K to 32K *all in mainframe.*
- **STANDARD FEATURES**
  Includes extended arithmetic instructions, power fail interrupt, memory parity check and memory protect.
- **FLEXIBLE INPUT/OUTPUT**
  14 internal I/O channels, externally expandable to 45.
- **FULL INTERRUPT SYSTEM**
  Interrupt priority easily established or changed for all devices.
- **COMPREHENSIVE SOFTWARE**
  Proven software packages for generating and executing your programs.

2100 computer

The Hewlett-Packard 2100 is a general-purpose digital computer designed for a wide range of small computer applications.

Features built-in to the 2100 include extended arithmetic instructions, power fail interrupt with automatic restart, memory parity check with interrupt and memory protect. Besides the standard built-in features, dual-channel Direct Memory Access (DMA) and Floating Point Hardware are also available. Under DMA control, data can be transferred to or from computer memory at rates greater than one million sixteen-bit words per second. Floating Point Hardware provides a typical ten-fold speed increase for scientific, computer bound algorithms.

A minimum 2100 provides 4096 words of core memory, self-contained power supply and 14 input/output channels.

You can select a wide range of memory sizes up to 32K words, all in mainframe. By including an HP 2155 Extender, you add another 31 input/output channels and power supply.

The 2100 automatically inherits a comprehensive range of proven software packages, including assemblers, compilers, operating systems and subroutines. A complete line of standard computer peripherals and I/O interface kits are also available, permitting complete systems to be tailored around the 2100. Added to these capabilities, you can also depend on the HP reputation for high quality and world-wide customer support. The result is a cost-effective computer that can meet your data processing problems today and continue meeting them as your needs expand.

UNITED STATES

ALABAMA
P.O. Box 4207
2003 Byrd Spring Road S.W.
Huntsville 35802
Tel: (205) 881-4591
TWX: 810-726-2204

ARIZONA
2336 E. Magnolia St.
Phoenix 85034
Tel: (602) 252-5061
TWX: 910-951-1330

5737 East Broadway
Tucson 85716
Tel: (602) 298-2313
TWX: 910-952-1162

CALIFORNIA
1430 East Orangethorpe Ave.
Fullerton 92631
Tel: (714) 870-1000

3939 Lankershim Boulevard
North Hollywood 91604
Tel: (213) 877-1282
TWX: 910-499-2170

1101 Embarcadero Road
Palo Alto 94303
Tel: (415) 327-6500
TWX: 910-373-1280

2220 Watt Ave.
Sacramento 95825
Tel: (916) 482-1463
TWX: 910-367-2092

9606 Aero Drive
San Diego 92123
Tel: (714) 279-3200
TWX: 910-335-2000

COLORADO
7965 East Prentice
Englewood 80110
Tel: (303) 771-3455
TWX: 910-935-0705

CONNECTICUT
12 Lunar Drive
New Haven 06525
Tel: (203) 389-6551
TWX: 710-465-2029

FLORIDA
P.O. Box 24210
2806 W. Oakland Park Blvd.
Ft. Lauderdale 33307
Tel: (305) 731-2020
TWX: 510-955-4099

P.O. Box 13910
6177 Lake Ellenor Dr.
Orlando, 32809
Tel: (305) 859-2900
TWX: 810-850-0113

GEORGIA
P.O. Box 28234
450 Interstate North
Atlanta 30328
Tel: (404) 436-6181
TWX: 810-766-4890

ILLINOIS
5500 Howard Street
Skokie 60076
Tel: (312) 677-0400
TWX: 910-223-3613

INDIANA
3839 Meadows Drive
Indianapolis 46205
Tel: (317) 546-4891
TWX: 810-341-3263

LOUISIANA
P.O. Box 856
1942 Williams Boulevard
Kenner 70062
Tel: (504) 721-6201
TWX: 810-955-5524

MARYLAND
6707 Whitestone Road
Baltimore 21207
Tel: (301) 944-5400
TWX: 710-862-9157

P.O. Box 1648
2 Choke Cherry Road
Rockville 20850
Tel: (301) 948-6370
TWX: 710-828-9684

MASSACHUSETTS
32 Hartwell Ave.
Lexington 02173
Tel: (617) 861-8960
TWX: 710-326-6904

MICHIGAN
21840 West Nine Mile Road
Southfield 48075
Tel: (313) 353-9100
TWX: 810-224-4882

MINNESOTA
2459 University Avenue
St. Paul 55114
Tel: (612) 645-9461
TWX: 910-563-3734

MISSOURI
11131 Colorado Ave.
Kansas City 64137
Tel: (816) 763-8000
TWX: 910-771-2087

148 Weldon Parkway
Maryland Heights 63043
Tel: (314) 567-1455
TWX: 910-764-0830

NEW JERSEY
W. 120 Century Road
Paramus 07652
Tel: (201) 265-5000
TWX: 710-990-4951

1060 N. Kings Highway
Cherry Hill 08034
Tel: (609) 667-4000
TWX: 710-892-4945

NEW MEXICO
P.O. Box 8366
Station C
6501 Lomas Boulevard N.E.
Albuquerque 87108
Tel: (505) 265-3713
TWX: 910-989-1665

156 Wyatt Drive
Las Cruces 88001
Tel: (505) 526-2485
TWX: 910-983-0550

NEW YORK
1702 Central Avenue
Albany 12205
Tel: (518) 869-8462
TWX: 710-441-8270

1219 Campville Road
Endicott 13760
Tel: (607) 754-0050
TWX: 510-252-0890

82 Washington Street
Poughkeepsie 12601
Tel: (914) 454-7330
TWX: 510-248-0012

39 Saginaw Drive
Rochester 14623
Tel: (716) 473-9500
TWX: 510-253-5981

5858 East Molloy Road
Syracuse 13211
Tel: (315) 454-2486
TWX: 710-541-0482

1 Crossways Park West
Woodbury 11797
Tel: (516) 921-0300
TWX: 510-223-0811

NORTH CAROLINA
P.O. Box 5188
1923 North Main Street
High Point 27262
Tel: (919) 885-8101
TWX: 510-926-1516

OHIO
25575 Center Ridge Road
Cleveland 44145
Tel: (216) 835-0300
TWX: 810-427-9129

3460 South Dixie Drive
Dayton 45439
Tel: (513) 298-0351
TWX: 810-459-1925

1120 Morse Road
Columbus 43229
Tel: (614) 846-1300

OKLAHOMA
2919 United Founders Boulevard
Oklahoma City 73112
Tel: (405) 848-2801
TWX: 910-830-6862

OREGON
Westhills Mall, Suite 158
4475 S.W. Scholls Ferry Road
Portland 97225
Tel: (503) 292-9171
TWX: 910-464-6103

PENNSYLVANIA
2500 Moss Side Boulevard
Monroeville 15146
Tel: (412) 271-0724
TWX: 710-797-3650

1021 8th Avenue
King of Prussia Industrial Park
King of Prussia 19406
Tel: (215) 265-7000
TWX: 510-660-2670

RHODE ISLAND
873 Waterman Ave.
East Providence 02914
Tel: (401) 434-5535
TWX: 710-381-7573

***TENNESSEE**
Memphis
Tel: (901) 274-7472

TEXAS
P.O. Box 1270
201 E. Arapaho Rd.
Richardson 75080
Tel: (214) 231-6101
TWX: 910-867-4723

P.O. Box 27409
6300 **Westpark Drive**
Suite 100
Houston 77027
Tel: (713) 781-6000
TWX: 910-881-2645

231 Billy Mitchell Road
San Antonio 78226
Tel: (512) 434-4171
TWX: 910-871-1170

UTAH
2890 South Main Street
Salt Lake City 84115
Tel: (801) 487-0715
TWX: 910-925-5681

VERMONT
P.O. Box 2287
Kennedy Drive
South Burlington 05401
Tel: (802) 658-4455
TWX: 510-299-0025

VIRGINIA
P.O. Box 6514
2111 Spencer Road
Richmond 23230
Tel: (703) 285-3431
TWX: 710-956-0157

WASHINGTON
433-108th N.E.
Bellevue 98004
Tel: (206) 454-3971
TWX: 910-443-2303

***WEST VIRGINIA**
Charleston
Tel: (304) 768-1232

FOR U.S. AREAS NOT LISTED:
Contact the regional office nearest you: Atlanta, Georgia . . . North Hollywood, California . . . Paramus, New Jersey . . . Skokie, Illinois. Their complete addresses are listed above.

*Service Only

SALES & SERVICE OFFICES

CANADA

ALBERTA
Hewlett-Packard (Canada) Ltd.
11748 Kingsway Ave.
Edmonton
Tel: (403) 452-3670
TWX: 610-831-2431

BRITISH COLUMBIA
Hewlett-Packard (Canada) Ltd.
4519 Canada Way
North Burnaby 2
Tel: (604) 433-8213
TWX: 610-922-5059

MANITOBA
Hewlett-Packard (Canada) Ltd.
513 Century St.
Winnipeg
Tel: (204) 786-7581
TWX: 610-671-3531

NOVA SCOTIA
Hewlett-Packard (Canada) Ltd.
2745 Dutch Village Rd.
Suite 206
Halifax
Tel: (902) 455-0511
TWX: 610-271-4482

ONTARIO
Hewlett-Packard (Canada) Ltd.
1785 Woodward Dr.
Ottawa 3
Tel: (613) 255-6180, 255-6530
TWX: 610-562-1952

Hewlett-Packard (Canada) Ltd.
50 Galaxy Blvd.
Rexdale
Tel: (416) 677-9611
TWX: 610-492-4246

QUEBEC
Hewlett-Packard (Canada) Ltd.
275 Hymus Boulevard
Pointe Claire
Tel: (514) 697-4232
TWX: 610-422-3022
Telex: 01-20607

FOR CANADIAN AREAS NOT LISTED:
Contact Hewlett-Packard (Canada) Ltd. in Pointe Claire, at the complete address listed above.

CENTRAL AND SOUTH AMERICA

ARGENTINA
Hewlett-Packard Argentina S.A.C.e.I
Lavalle 1171 - 3°
Buenos Aires
Tel: 35-0436, 35-0627, 35-0431
Telex: 012-1009
Cable: HEWPACK ARG

BRAZIL
Hewlett-Packard Do Brasil I.e.C Ltda.
Rua Frei Caneca 1119
Sao Paulo - 3, SP
Tel: 288-7111, 287-5858
Cable: HEWPACK Sao Paulo

Hewlett-Packard Do Brasil
Praca Dom Feliciano 78
Salas 806/808
Porto Alegre
Rio Grande do Sul (RS)-Brasil
Tel: 25-8470
Cable: HEWPACK Porto Alegre

Hewlett-Packard Do Brasil
I.e.C. Ltda.
Rua da Matriz 29
Botafogo ZC-02
Rio de Janeiro, GB
Tel: 246-4417, 246-2919
Cable: HEWPACK Rio de Janeiro

CHILE
Héctor Calcagni y Cia, Ltda.
Bustos, 1932-3er Piso
Casilla 13942
Santiago
Tel: 423 96
Cable: CALCAGNI Santiago

COLOMBIA
Instrumentacion
Henrik A. Langebaek & Kier S.A.
Carrera 7 No. 48-59
Apartado Aereo 6287
Bogota, 1 D.E.
Tel: 45-78-06, 45-55-46
Cable: AARIS Bogota
Telex: 44400INSTCO

COSTA RICA
Lic. Alfredo Gallegos Gurdián
Apartado 10159
San José
Tel: 21-86-13
Cable: GALGUR San José

ECUADOR
Laboratorios de Radio-Ingenieria
Calle Guayaquil 1246
Post Office Box 3199
Quito
Tel: 212-496; 219-185
Cable: HORVATH Quito

EL SALVADOR
Electronic Associates
Apartado Postal 1682
Centro Comercial Gigante
San Salvador, El Salvador C.A.
Paseo Escalon 4649-4° Piso
Tel: 23-44-60, 23-32-37
Cable: ELECAS

GUATEMALA
IPESA
5a via 2-01, Zona 4
Guatemala City

MEXICO
Hewlett-Packard Mexicana, S.A. de C.V.
622 Adolfo Prieto
Col. del Valle
Mexico 12, D.F.
Tel: 543-4232; 523-1874
Telex: 017-74-507

NICARAGUA
Roberto Terán G.
Apartado Postal 689
Edificio Terán
Managua
Tel: 3451, 3452
Cable: ROTERAN Managua

PANAMA
Electrónico Balboa, S.A.
P.O. Box 4929
Ave. Manuel Espinosa No. 13-50
Bldg. Alina
Panama City
Tel: 230833
Telex: 3481003, Curundu, Canal Zone
Cable: ELECTRON Panama City

PARAGUAY
Z.T. Melamed S.R.L.
Division: Aparatos y Equipos Medicos
Salon de Exposicion y Escritorio:
Chile 482
Edificio Victoria—Planta Baja
Asuncion, Paraguay
Tel: 4-5069, 4-6272
Cable: RAMEL

PERU
Compañia Electro Medica S.A.
Ave. Enrique Canaual 312
San Isidro
Casilla 1030
Lima
Tel: 22-3900
Cable: ELMED Lima

PUERTO RICO
San Juan Electronics, Inc.
P.O. Box 5167
Ponce de Leon 154
Pda. 3-PTA de Tierra
San Juan 00906
Tel: (809) 725-3342, 722-3342
Cable: SATRONICS San Juan
Telex: SATRON 3450 332

URUGUAY
Pablo Ferrando S.A.
Comercial e Industrial
Avenida Italia 2877
Casilla de Correo 370
Montevideo
Tel: 40-3102
Cable: RAOIUM Montevideo

VENEZUELA
Hewlett-Packard De Venezuela C.A.
Apartado 50933
Caracas
Tel: 71.88.05, 71.88.69, 71.99.30
Cable: HEWPACK Caracas
Telex: 21146 HEWPACK

FOR AREAS NOT LISTED,

CONTACT:
Hewlett-Packard INTERCONTINENTAL
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 493-1501
TWX: 910-373-1267
Cable: HEWPACK Palo Alto
Telex: 034-8300, 034-8493

EUROPE

AUSTRIA
Hewlett-Packard Ges.m.b.H
Innstrasse 23/2
Postfach 45
A-1204 **Vienna**
Tel: (0222) 33 66 06 to 09
Cable: HEWPAK Vienna
Telex: 75923 hewpak a

BELGIUM
Hewlett-Packard Benelux
S.A./N.V.
Avenue du Col-Vert, 1
B-1170 **Brussels**
Tel: (02) 72 22 40
Cable: PALOBEN Brussels
Telex: 23 494

DENMARK
Hewlett-Packard A/S
Datavej 38
DK-3460 **Birkerod**
Tel: (01) 81 66 40
Cable: HEWPACK AS
Telex: 16640 hp as

Hewlett-Packard A/S
Torvet 9
DK-8600 **Silkeborg**
Tel: (06)-82-71-66
Telex: 16640 hp as
Cable: HEWPACKAS

FINLAND
Hewlett-Packard Oy
Bulevardi 26
P.O. Box 12185
SF-00120 **Helsinki** 12
Tel: 13-730
Cable: HEWPACKOY-Helsinki
Telex: 17-1563 hel

FRANCE
Hewlett-Packard France
Quartier de Courtaboeuf
Boite Postale No. 6
F-91 **Orsay**
Tel: (1) 907 78 25
Cable: HEWPACK Orsay
Telex: 60048

Hewlett-Packard France
4 Quai des Etroits
F-69 **Lyon** 5ème
Tel: (78) 42 63 45
Cable: HEWPACK Lyon
Telex: 31617

Hewlett-Packard France
29 rue de la Gare
F-31 **Blagnac**
Tel: (61) 85 82 29
Telex: 51957

GERMAN FEDERAL REPUBLIC
Hewlett-Packard Vertriebs-GmbH
Berliner Strasse 117
Postfach 560 140
D-6 Nieder-Eschbach/Ffm 56
Tel: (0611) 50-04-1
Cable: HEWPACKSA Frankfurt
Telex: 41 32 49 FRA

Hewlett-Packard Vertriebs-GmbH
Herrenbergerstrasse 110
D-7030 **Böblingen**, Württemberg
Tel: (07031) 66 72 86-87
Cable: HEPAK Böblingen
Telex: 72 65 739 bbn

Hewlett-Packard Vertriebs-GmbH
Vogelsanger Weg 38
D-4 **Düsseldorf**
Tel: (0211) 63 80 31/35
Telex: 85/86 533 hpdd d

Hewlett-Packard Vertriebs-GmbH
Wendenstr. 23
D-2 **Hamburg** 1
Tel: (0411) 24 05 51/52
Cable: HEWPACKSA Hamburg
Telex: 21 53 32 hphh d

Hewlett-Packard Vertriebs-GmbH
Unterhachinger Strasse 28
ISAR Center
D-8012 **Ottobrunn**
Tel: (0811) 60 13 061-7
Telex: 52 49 85
Cable: HEWPACKSA Müchen

(West Berlin)
Hewlett-Packard Vertriebs-GmbH
Wilmersdorfer Strasse 113/114
D-1000 **Berlin W.** 12
Tel: (0311) 3137046
Telex: 18 34 05 hpbln d

GREECE
Kostas Karayannis
18, Ermou Street
Athens 126
Tel: 3230-303
Cable: RAKAR Athens
Telex: 21 59 62 rkar gr

IRELAND
Hewlett-Packard Ltd.
224 Bath Road
Slough, SL1 4 DS, Bucks
Tel: Slough 753-33341
Cable: HEWPIE Slough
Telex: 84413

ITALY
Hewlett-Packard Italiana S.p.A.
Via Amerigo Vespucci 2
I-20124 **Milan**
Tel: (2) 6251 (10 lines)
Cable: HEWPACKIT Milan
Telex: 32046

Hewlett-Packard Italiana S.p.A.
Piazza Marconi
I-00144 **Rome** - Eur
Tel: (6) 5912544/5, 5915947
Cable: HEWPACKIT Rome
Telex: 61514

Hewlett-Packard Italiana S.p.A.
Vicolo Pastori, 3
I-35100 **Padova**
Tel: (49) 66 40 62
Telex: 32046 via Milan

Hewlett-Packard Italiana S.p.A.
Via Colli, 24
I-10129 **Turin**
Tel: (11) 53 82 64
Telex: 32046 via Milan

LUXEMBURG
Hewlett-Packard Benelux
S.A./N.V.
Avenue du Col-Vert, 1
B-1170 **Brussels**
Tel: (03/02) 72 22 40
Cable: PALOBEN Brussels
Telex: 23 494

NETHERLANDS
Hewlett-Packard Benelux, N.V.
Weerdestein 117
P.O. Box 7825
Amsterdam, Z 11
Tel: 020-42 77 77
Cable: PALOBEN Amsterdam
Telex: 13 216 hepa nl

NORWAY
Hewlett-Packard Norge A/S
Box 149
Nesveien 13
N-1344 **Haslum**
Tel: (02)-53 83 60
Telex: 16621

PORTUGAL
Telectra-Empresa Tecnica de Equipamentos
Electricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
P-**Lisbon** 1
Tel: (19) 68 60 72
Cable: TELECTRA Lisbon
Telex: 1598

SPAIN
Hewlett-Packard Española, S.A.
Jerez No 8
Madrid 16
Tel: 458 26 00
Telex: 23515 hpe

Hewlett-Packard Española, S.A.
Milanesado 21-23
E-**Barcelona** 17
Tel: (3) 203 62 00

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvägen 1-3
Fack
S-161 20 **Bromma** 20
Tel: (08) 98 12 50
Cable: MEASUREMENTS Stockholm
Telex: 10721

Hewlett-Packard Sverige AB
Hagakersgatan 9C
S-431 41 **Mölndal**
Tel: (031) 27 68 00/01
Telex: 21 312 hpmindl s

SWITZERLAND
Hewlett Packard (Schweiz) AG
Zürcherstrasse 20
P.O. Box 64
CH-8952 **Schlieren** Zurich
Tel: (01) 98 18 21/24
Cable: HPAG CH
Telex: 53933

Hewlett-Packard (Schweiz) AG
Rue du Bois-du-Lan 7
P.O. Box 85
1217 Meyrin 2 **Geneva**
Tel: (022) 41 54 00
Cable: HEWPACKSA Geneva
Telex: 27333 hpsa ch

TURKEY
Telekom Engineering Bureau
P.O. Box 376
Karaköy
Istanbul
Tel: 49 40 40
Cable: TELEMATION Istanbul

UNITED KINGDOM
Hewlett-Packard Ltd.
224 Bath Road
Slough, SL1 4 DS, Bucks
Tel: Slough (0753) 33341
Cable: HEWPIE Slough
Telex: 84413

Hewlett-Packard Ltd.
"The Graftons"
Stamford New Road
Altrincham, Cheshire
Tel: (061) 928-8626
Telex: 668068

SOCIALIST COUNTRIES PLEASE CONTACT:
Hewlett-Packard Ges.m.b.H
Innstrasse 23/2
Postfach 45
A-1204 **Vienna**, Austria
Tel: (0222) 33 66 06-09
Cable: HEWPACK Vienna
Telex: 75923 hewpak a

ALL OTHER EUROPEAN COUNTRIES CONTACT:
Hewlett-Packard S.A.
Rue du Bois-du-Lan 7
P.O. Box 85
CH-1217 Meyrin 2 **Geneva**
Switzerland
Tel: (022) 41 54 00
Cable: HEWPACKSA Geneva
Telex: 2.24.86

AFRICA, ASIA, AUSTRALIA

ANGOLA
Telectra Empresa Técnia
de Equipamentos Eléctricos
SAR
Rua de Barbosa Rodrigues
42-1°
Box 6487
Luanda
Cable: TELECTRA Luanda

AUSTRALIA
Hewlett-Packard Australia
Pty. Ltd.
22-26 Weir Street
Glen Iris, 3146
Victoria
Tel: 20-1371 (6 lines)
Cable: HEWPARD Melbourne
Telex: 31 024

Hewlett-Packard Australia
Pty. Ltd.
61 Alexander Street
Crows Nest 2065
New South Wales
Tel: 43-7866
Cable: HEWPARD Sydney
Telex: 21561

Hewlett-Packard Australia
Pty. Ltd.
97 Churchill Road
Prospect 5082
South Australia
Tel: 65-2366
Cable: HEWPARD Adelaide

Hewlett Packard Australia
Pty. Ltd.
2nd Floor, Suite 13
Casablanca Buildings
196 Adelaide Terrace
Perth, W.A. 6000
Tel: 21-3330
Cable: HEWPARD Perth

Hewlett-Packard Australia
Pty. Ltd.
10 Woolley Street
P.O. Box 191
Dickson A.C.T. 2602
Tel: 49-8194
Cable: HEWPARD Canberra ACT

Hewlett-Packard Australia
Pty. Ltd.
6 Harvard Street
P.O. Box 135
Kenmore 4069 Queensland
Tel: 70-4050

CEYLON
United Electricals Ltd.
P.O. Box 681
Yahala Building
Staples Street
Colombo 2
Tel: 5496
Cable: HOTPOINT Colombo

CYPRUS
Kypronics
19 Gregorios & Xenopoulos Road
P.O. Box 1152
Nicosia
Tel: 6282-75628
Cable: HE-I-NAMI

ETHIOPIA
African Salespower & Agency
Private Ltd., Co.
P. O. Box 718
58/59 Cunningham St.
Addis Ababa
Tel: 12285
Cable: ASACO Addisababa

HONG KONG
Schmidt & Co. (Hong Kong) Ltd.
P.O. Box 297
1511, Prince's Building 15th Floor
10, Chater Road
Hong Kong
Tel: 240168, 232735
Cable: SCHMIDTCO Hong Kong

INDIA
Blue Star Ltd.
Kasturi Buildings
Jamshedji Tata Rd.
Bombay 20BR, India
Tel: 29 50 21
Telex: 3751
Cable: BLUEFROST

Blue Star Ltd.
Band Box House
Prabhadevi
Bombay 25DD, India
Tel: 45 73 01
Telex: 3751
Cable: BLUESTAR

Blue Star Ltd.
14/40 Civil Lines
Kanpur, India
Tel: 6 88 82
Cable: BLUESTAR

Blue Star, Ltd.
7 Hare Street
P.O. Box 506
Calcutta 1, India
Tel: 23-0131
Telex: 655
Cable: BLUESTAR

Blue Star Ltd.
Blue Star House,
34 Ring Road
Lajpat Nagar
New Delhi 24, India
Tel: 62 32 76
Telex: 463
Cable: BLUESTAR

Blue Star, Ltd.
Blue Star House
11/11A Magarath Road
Bangalore, 25
Tel: 51473
Telex: 430
Cable: BLUESTAR

Blue Star, Ltd.
1-1-117/1
Sarojini Devi Road
Secunderabad 3
Tel: 7 63 91, 7 73 93
Cable: BLUEFROST

Blue Star, Ltd.
23/24 Second Line Beach
Madras 1, India
Tel: 2 39 55
Telex: 379
Cable: BLUESTAR

Blue Star, Ltd.
1B Kaiser Bungalow
Dindli Road
Jamshedpur, India
Tel: 38 04
Cable: BLUESTAR

INDONESIA
Bah Bolon Trading Coy. N.V.
Djalah Merdeka 29
Bandung
Tel: 4915; 51560
Cable: !LMU
Telex: 08-809

IRAN
Multicorp International Ltd.
Avenue Soraya 130
P.O. Box 1212
Teheran
Tel: 83 10 35-39
Cable: MULTICORP Tehran
Telex: 2876 service tn

ISRAEL
Electronics & Engineering
Div. of Motorola Israel Ltd.
17 Aminadav Street
Tel-Aviv
Tel: 36941 (3 lines)
Cable: BASTEL Tel-Aviv
Telex: MOTIL IL

JAPAN
Yokogawa-Hewlett-Packard Ltd.
Ohashi Building
1-59-1 Yoyogi
Shibuya-ku, **Tokyo**
Tel: 03-370-2281/92
Telex: 232-2024YHP
Cable: YHPMARKET TOK 23-724

Yokogawa-Hewlett-Packard Ltd.
Nisei Ibaragi Bldg.
2-2-8 Kasuga
Ibaragi-Shi
Osaka
Tel: (0726) 23-1641
Telex: 5332-385 YHP OSAKA

Yokogawa-Hewlett-Packard Ltd.
Ito Building
No. 59, Kotori-cho
Nakamura-ku, **Nagoya** City
Tel: (052) 551-0215

Yokogawa-Hewlett-Packard Ltd.
Nitto Bldg.
2-4-2 Shinohara-Kita
Kohoku-ku
Yokohama 222
Tel: 045-432-1504
Telex: 382-3204 YHP YOK

JORDAN
Constantin E. Macridis
Clemenceau Street
P.O. Box 7213
Beirut, Lebanon
Tel: 220846
Cable: ELECTRONUCLEAR Beirut

KENYA
Kenya Kinetics
P.O. Box 18311
Nairobi, Kenya
Tel: 57726
Cable: PROTON

KOREA
Amtraco Corporation
Industrial Products Div.
Seoul P.O. Box 1103
8th floor, DaeKyung Bldg.
107 Sejong Ro
Chongro-Ku, Seoul
Tel: 73-8924-7
Cable: AMTRACO Seoul

LEBANON
Constantin E. Macridis
Clemenceau Street
P.O. Box 7213
Beirut
Tel: 220846
Cable: ELECTRONUCLEAR Beirut

MALAYSIA
MECOMB Malaysia Ltd.
2 Lorong 13/6A
Section 13
Petaling Jaya, Selangor
Cable: MECOMB Kuala Lumpur

MOZAMBIQUE
A. N. Goncalves, LDA.
4.1 Apt. 14 Av. D. Luis
P.O. Box 107
Lourenco Marques
Cable: NEGON

NEW ZEALAND
Hewlett-Packard (N.Z.) Ltd.
94-96 Dixson St.
P.O. Box 9443
Wellington, N.Z.
Tel: 56-559
Cable: HEWPACK Wellington

Hewlett Packard (N.Z.) Ltd.
Box 51092
Pukuranga
Tel: 56-9837
Cable: HEWPACK, Auckland

PAKISTAN
Mushko & Company, Ltd.
Oosman Chambers
Abdullah Haroon Road
Karachi 3
Tel: 511027, 512927
Cable: COOPERATOR Karachi

Mushko & Company, Ltd.
38B, Satellite Town
Rawalpindi
Tel: 41924
Cable: FEMUS Rawalpindi

PHILIPPINES
Electromex Inc.
5th Floor, Architects
Center Bldg.
Ayala Ave., Makati, Rizal
C.C.P.O. Box 1028
Makati, Rizal
Tel: 86-18-87, 87-76-77
Cable: ELEMEX Manila

SINGAPORE
Mechanical and Combustion
Engineering Company Ltd.
9, Jalan Kilang
Red Hill Industrial Estate
Singapore, 3
Tel: 642361-3; 632611
Cable: MECOMB Singapore

AFRICA, ASIA, AUSTRALIA (Continued)

Hewlett-Packard Far East
Area Office
P.O. Box 87
Alexandra Post Office
Singapore 3
Tel: 633022
Cable: HEWPACK SINGAPORE

SOUTH AFRICA
Hewlett Packard South Africa
(Pty.), Ltd.
P.O. Box 31716
Braamfontein Transvaal
Milnerton
30 De Beer Street
Johannesburg
Tel: 725-2080, 725-2030
Telex: 0226 JH
Cable: HEWPACK Johannesburg

Hewlett Packard South Africa
(Pty.), Ltd.
Breecastle House
Bree Street
Cape Town
Tel: 3-6019, 3-6545
Cable: HEWPACK Cape Town
Telex: 5-0006

Hewlett Packard South Africa
(Pty.), Ltd.
641 Ridge Road, Durban
P.O. Box 99
Overport, Natal
Tel: 88-6102
Telex: 567954
Cable: HEWPACK

TAIWAN
Hewlett Packard Taiwan
39 Chung Shiao West Road
Sec. 1
Overseas Insurance
Corp. Bldg. 7th Floor
Taipei
Tel: 389160,1,2, 375121,
Ext. 240-249
Telex: TP824 HEWPACK
Cable: HEWPACK Taipei

THAILAND
UNIMESA Co., Ltd.
Chongkoinee Building
56 Suriwongse Road
Bangkok
Tel: 37956, 31300, 31307,
37540
Cable: UNIMESA Bangkok

UGANDA
Uganda Tele-Electric Co., Ltd.
P.O. Box 4449
Kampala
Tel: 57279
Cable: COMCO Kampala

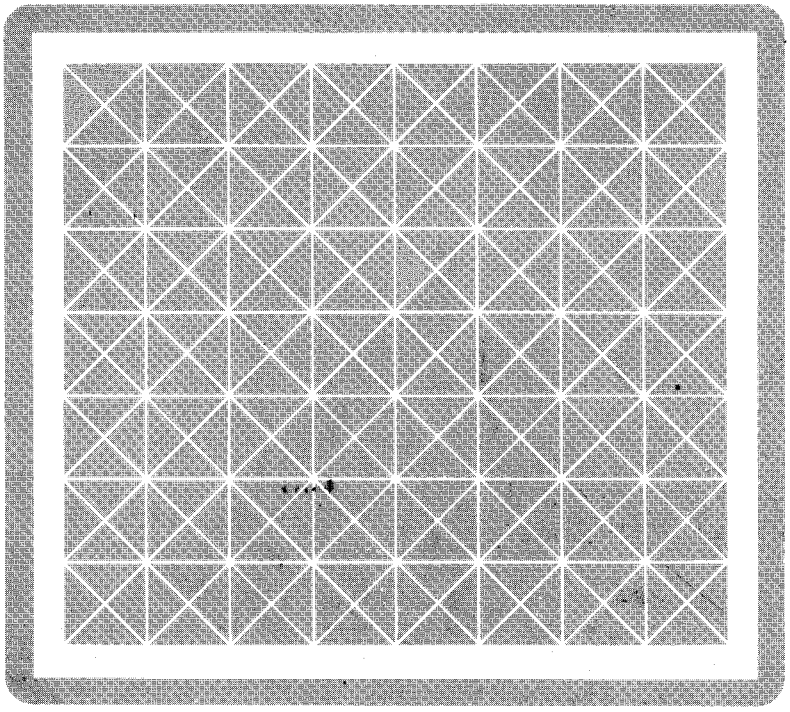
VIETNAM
Peninsular Trading Inc.
P.O. Box H-3
216 Hien-Vuong
Saigon
Tel: 20-805, 93398
Cable: PENTRA, SAIGON 242

ZAMBIA
R. J. Tilbury (Zambia) Ltd.
P.O. Box 2792
Lusaka
Zambia, Central Africa
Tel: 73793
Cable: ARJAYTEE, Lusaka

MEDITERRANEAN AND MIDDLE EAST COUNTRIES NOT SHOWN PLEASE CONTACT:
Hewlett-Packard
Co-ordination Office for
Mediterranean and Middle
East Operations
Via Marocco, 7
I-00144 Rome-Eur, Italy
Tel: (6) 59 40 29
Cable: HEWPACKIT Rome
Telex: 61514

OTHER AREAS NOT LISTED, CONTACT:
Hewlett-Packard
INTERCONTINENTAL
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 326-7000
(Feb. 71 493-1501)
TWX: 910-373-1267
Cable: HEWPACK Palo Alto
Telex: 034-8300, 034-8493

HEWLETT hp PACKARD



02100-90133

Printed in U.S.A. 8/72