HEWLETT hp PACKARD

2100 computer

dos microprogramming software

HEWLETT hp PACKARD

DOS MICROPROGRAMMING SOFTWARE

for

Hewlett-Packard Model 2100 Computer

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

02100-90146 Printed: May 1973

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

BCS/DOS Applicability of the *HP 2100 Computer Microprogramming Software* Manual.

Manual Section	Applies to
1	Both the BCS and DOS versions. However, the DOS version allows the use of absolute octal control store addresses as jump targets whereas the BCS version does not (this affects the discussion of the Skip field in the "Microinstruction Format" part of section 1).
2	The BCS version only. Section 2 for the DOS version is contained in this document.
3	Both the BCS and DOS versions.
4	Both the BCS and DOS versions.
5	The BCS version only. Section 5 for the DOS version is contained in this document.
6	Both the BCS and DOS versions.
7	Both the BCS and DOS versions.
8	Both the BCS and DOS versions.
9	The BCS version only. Section 9 for the DOS version is contained in this document.
10	The BCS version only. There is no DOS version of the Programmable ROM Writer.

PREFACE

This manual presents the Disc Operating System (DOS) version of the HP Microassembler, the HP Micro Debug Editor, and the HP Writable Control Store (WCS) Input/Output Utility Routine. The Basic Control System (BCS) versions of these programs are described in the *HP 2100 Computer Microprogramming Software* manual (02100-90133).

The DOS and BCS versions of the HP Microassembler are very similar to one another. Many of the sections of the above-mentioned manual also apply to the DOS version. The applicability of the various manual sections is summarized in the table on the facing page.

For a DOS environment, the descriptions of the HP Micro Debug Editor and the WCS Input/Output Utility Routine in this manual should be used instead of the corresponding descriptions in the *HP 2100 Computer Microprogramming Software* manual.

There is no DOS version of the HP Programmable ROM Writer program. Mask tapes generated by the HP DOS Micro Debug Editor can be used as input to the BCS version of the HP Programmable ROM Writer program.

CONTENTS

ILLUSTRATIONS

TABLES

The Disc Operating System (DOS) version of the HP Microassembler translates symbolic source language microinstructions into a machine language object microprogram. Source input can be read from a disc file, punched cards, or punched tape; the object program can be stored in a disc file or punched on tape in a format acceptable to the HP DOS 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 DOS Microassembler is designed to run in a minimum DOS environment.

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 and any $EXTERNALS assembler control statements that are present. In addition, it checks the validity of all assembler control statements, checks for duplicate labels, and (if necessary) generates appropriate error messages.

In the second pass, the Microassembler reads the entire source microprogram again 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 microprogram is created, the assembly listing is printed, and the object microprogram is stored on disc or punched on tape.

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. Fatal error messages, on the other hand, draw the microprogrammer's attention to errors which must be remedied in order for a correct assembly to be achieved. In either case, the assembly process continues after the error message has been printed. In the case of fatal error messages, the microprogrammer must correct the error after the assembly is complete and then reassemble the source microprogram. All warning and fatal error messages are presented in section 8, "Error Messages", of the *HP 2100 Computer Microprogramming Software* manual (02100-90133).

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 precedes the offending source statement. The assembly listing is discussed in greater detail later in this section.

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 the 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.

> 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.

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 $EXTERNALS assembler control statements.

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):

- Define external symbolic addresses ($EXTERNALS).
- Prevent binary output, i.e., the object microcode, from being sent to the DOS punch device ($NOPUNCH).
- Prevent the assembly listing from being sent to the DOS list device ($NOLIST).
- Reset the program location counter ($ORIGIN).
- Suppress the printing of all warning messages ($SUPPRESS).

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

ASSEMBLER OUTPUT

The Microassembler produces a printed assembly listing and an object microprogram. The object microprogram may either be stored in a disc file or punched on tape. When punched on tape, the object microprogram is punched as shown in figure 2-2. In either case, the object microprogram is in a format acceptable to the HP DOS Micro Debug Editor.

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

The symbols are listed in the order in which they were defined in the source microprogram. In the listing, an external symbol is easily identifiable by the "X" immediately following the associated absolute control store address. Specifically, the format of each line in the 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 of 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 microcode appears in print positions 11 through 20.

The object microcode is interpreted as follows:

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

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

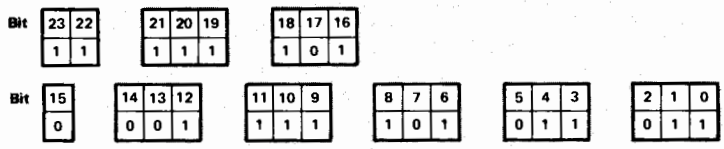
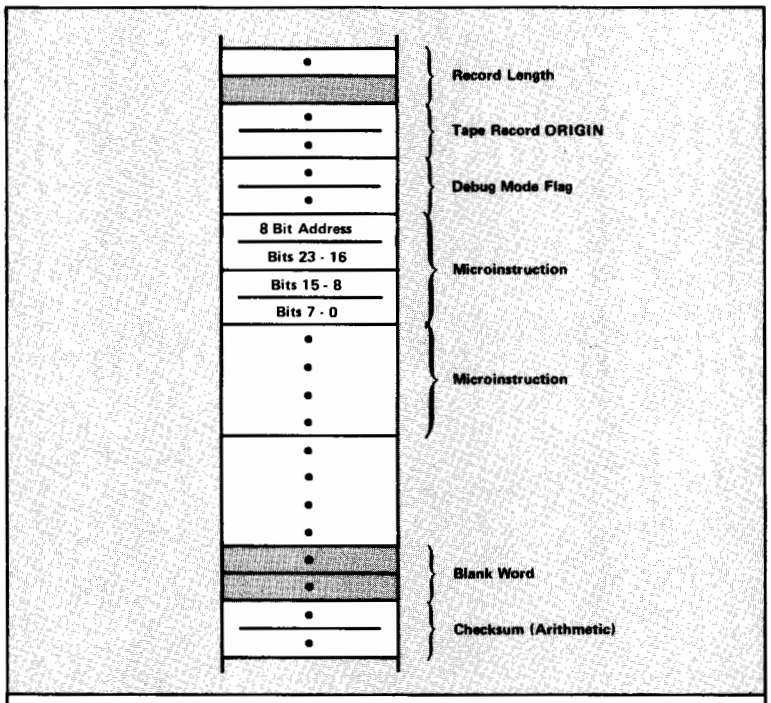


Figure 2-1. Object Code Illustration



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

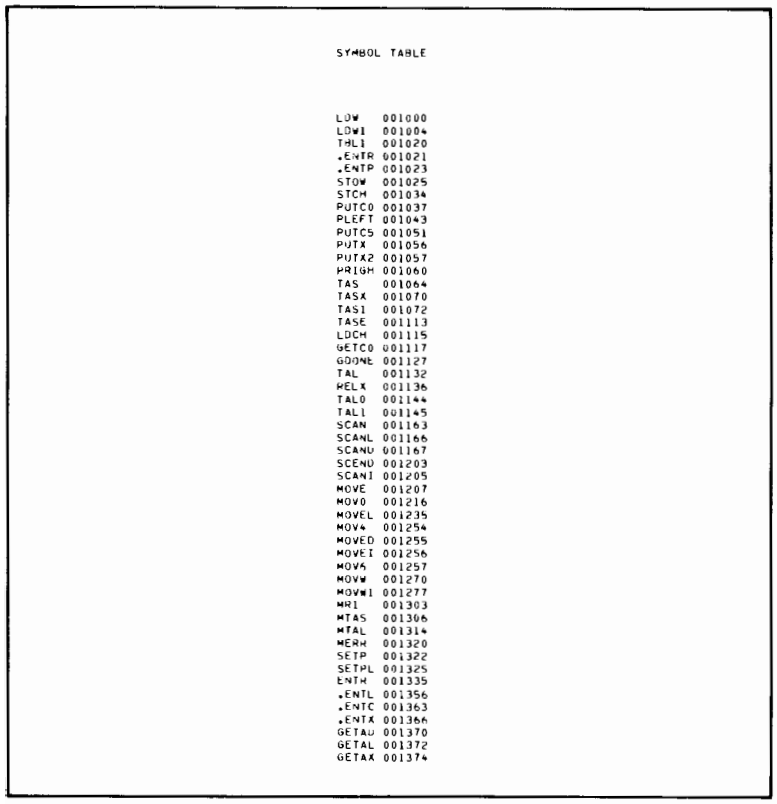
Each microinstruction consists of two computer words (four tape characters).

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

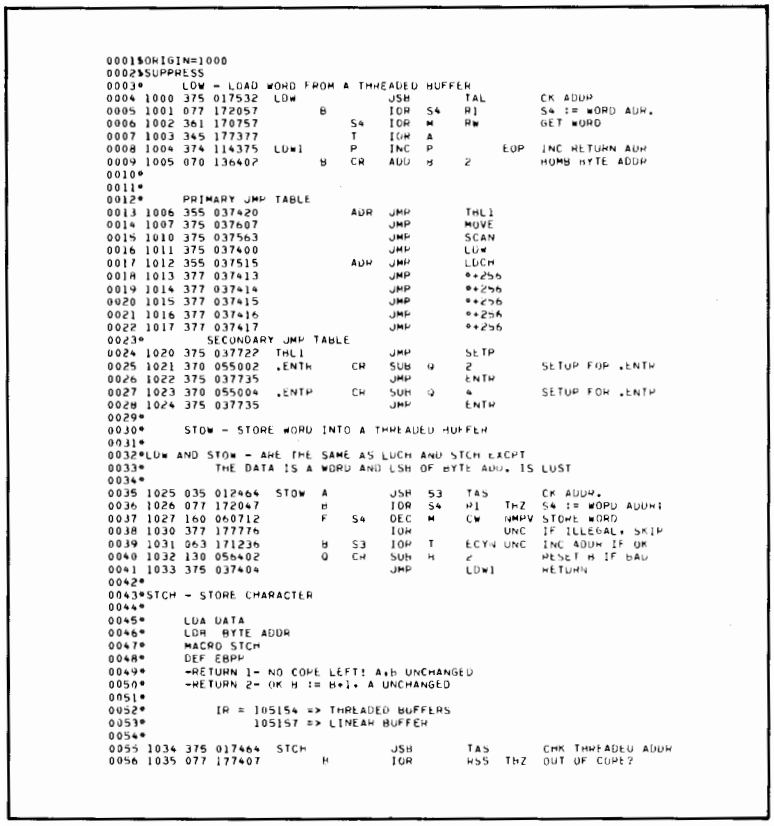


```
SYMBOL TABLE


LOW    001000
LOWI   001004
THLI   001020
.ENTR  001021
.ENTP  001023
STOW   001025
STCH   001034
PUTC0  001037
PLEFT  001043
PUTC5  001051
PUTX   001056
PUTX2  001057
PRIGH  001060
TAS    001064
TASX   001070
TAS1   001072
TASE   001113
LDCH   001115
GETC0  001117
GDONE  001127
TAL    001132
RELX   001136
TAL0   001144
TAL1   001145
SCAN   001163
SCANL  001166
SCANU  001167
SCEND  001203
SCANI  001205
MOVE   001207
MOV0   001216
MOVEL  001235
MOV4   001254
MOVED  001255
MOVEI  001256
MOV5   001257
MOVW   001270
MOVWI  001277
WRI    001303
MTAS   001306
MTAL   001314
WERR   001320
SETP   001322
SETPL  001325
ENTR   001335
.ENTL  001356
.ENTC  001363
.ENTX  001366
GETAU  001370
GETAL  001372
GETAX  001374
```

Figure 2-3. Symbol Table Listing

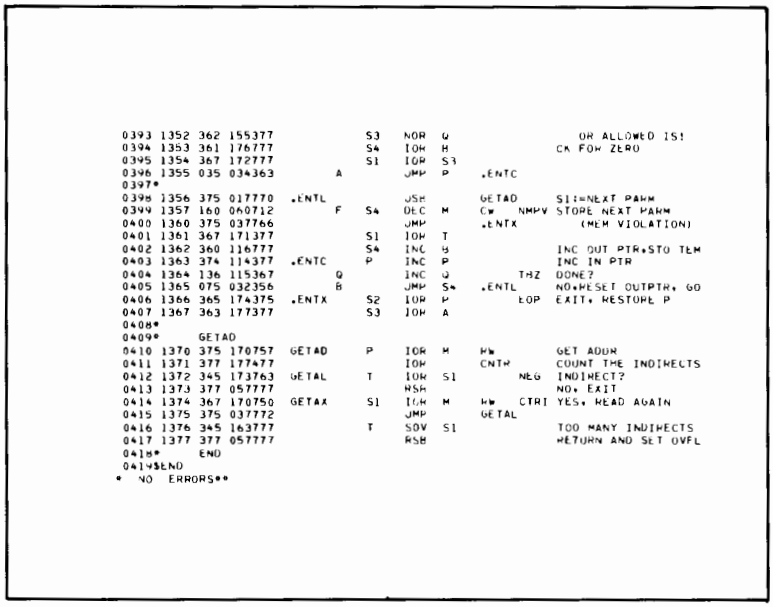
The source language microinstruction is then printed, starting in print position 25, exactly as it appears in the source input. Note that if a teleprinter or an 80-column line printer is used as the DOS list device, the source statements are truncated after character positions 48 and 56, respectively, of the source statement.



```
0001$ORIGIN=1000
0002$SUPPRESS
0003*     LDW - LOAD WORD FROM A THREADED BUFFER
0004 1000 375 017532  LDW             JSB       TAL       CK ADDR
0005 1001 077 172057          B       IOR   S4   R1        S4 := WORD ADR.
0006 1002 361 170757              S4  IOR   M    RM        GET WORD
0007 1003 345 177377              T   IOR   A
0008 1004 374 114375  LDW1        P   INC   P         EOP  INC RETURN ADR
0009 1005 070 136402          B   CR  ADD   B    2         HOMB BYTE ADDR
0010*
0011*
0012*     PRIMARY JMP TABLE
0013 1006 355 037420              ADR JMP       THL1
0014 1007 375 037607                  JMP       MOVE
0015 1010 375 037563                  JMP       SCAN
0016 1011 375 037400                  JMP       LDW
0017 1012 355 037515              ADR JMP       LDCH
0018 1013 377 037413                  JMP       *+256
0019 1014 377 037414                  JMP       *+256
0020 1015 377 037415                  JMP       *+256
0021 1016 377 037416                  JMP       *+256
0022 1017 377 037417                  JMP       *+256
0023*         SECONDARY JMP TABLE
0024 1020 375 037722  THL1            JMP       SETP
0025 1021 370 055002  .ENTW      CR   SUB   Q    2         SETUP FOR .ENTW
0026 1022 375 037735                  JMP       ENTW
0027 1023 370 055004  .ENTP      CH   SUB   Q    4         SETUP FOR .ENTP
0028 1024 375 037735                  JMP       ENTW
0029*
0030*     STOW - STORE WORD INTO A THREADED BUFFER
0031*
0032*LDW AND STOW - ARE THE SAME AS LDCH AND STCH EXCPT
0033*          THE DATA IS A WORD AND LSH OF BYTE ADD. IS LOST
0034*
0035 1025 035 012464  STOW  A         JSB   S3    TAS       CK ADDR.
0036 1026 077 172047        B         IOR   S4    R1   THZ  S4 := WORD ADDR1
0037 1027 160 060712        F   S4    DEC   M     CW   NMPV STORE WORD
0038 1030 377 177776                  IOR                UNC  IF ILLEGAL, SKIP
0039 1031 063 171236        B   S3    IOR   T     ECYN UNC  INC ADDR IF OK
0040 1032 130 056402        Q   CH    SUB   H     2         RESET B IF BAD
0041 1033 375 037404                  JMP       LDW1      RETURN
0042*
0043*STCH - STORE CHARACTER
0044*
0045*     LDA DATA
0046*     LDA  BYTE ADDR
0047*     MACRO STCH
0048*     DEF EBPP
0049*     -RETURN 1- NO CORE LEFT! A,B UNCHANGED
0050*     -RETURN 2- OK B := B+1. A UNCHANGED
0051*
0052*          IR = 105154 => THREADED BUFFERS
0053*               105157 => LINEAR BUFFER
0054*
0055 1034 375 017464  STCH            JSB       TAS       CHK THREADED ADDR
0056 1035 077 177407        B         IOR       RSS  THZ  OUT OF CORE?
```

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

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 microprogram will require (maximum allowable = 400_8 per control store module).



```
0393 1352 362 155377            S3   NOR  Q              OR ALLOWED IS1
0394 1353 361 176777            S4   IOR  M           CK FOR ZERO
0395 1354 367 172777            S1   IOR  S3
0396 1355 035 034363       A         JMP  P     .ENTC
0397*
0398 1356 375 017770  .ENTL           JSB        GETAD      S1:=NEXT PARM
0399 1357 160 060712       F    S4   DEC  M     CW    NMPV STORE NEXT PARM
0400 1360 375 037766                 JMP        .ENTX         (MEM VIOLATION)
0401 1361 367 171377            S1   IOR  T
0402 1362 360 116777            S4   INC  S4             INC OUT PTR,STO TEM
0403 1363 374 114377  .ENTC    P    INC  P              INC IN PTR
0404 1364 136 115367       Q         INC  Q         THZ  DONE?
0405 1365 075 032356       B         JMP  S*    .ENTL     NO,RESET OUTPTR, GO
0406 1366 365 174375  .ENTX      S2   IOR  P         EOP  EXIT, RESTORE P
0407 1367 363 177377            S3   IOR  A
0408*
0409*      GETAD
0410 1370 375 170757  GETAD    P    IOR  M     RW        GET ADDR
0411 1371 377 177477                 IOR        CNTR      COUNT THE INDIRECTS
0412 1372 345 173763  GETAL    T    IOR  S1        NEG   INDIRECT?
0413 1373 377 057777                 RSB                  NO, EXIT
0414 1374 367 170750  GETAX    S1   IOR  M     RW    CTRI YES, READ AGAIN
0415 1375 375 037772                 JMP        GETAL
0416 1376 345 163777            T    SOV  S1             TOO MANY INDIRECTS
0417 1377 377 057777                 RSB                  RETURN AND SET OVFL
0418*      END
0419$END
*  NO  ERRORS**
```

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

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

ERROR xx

WARNING xx IN LINE yy

ERROR xx IN LINE yy

where *xx* is the message number (see section 8 of the *HP 2100 Computer Microprogramming Software* manual, 02100-90133) and *yy* is the number of the erroneous line in the source microprogram listing.

OPERATING INSTRUCTIONS

The HP DOS Microassembler is loaded like any other DOS program by using the :PROG directive. The format of the :PROG directive and the meaning of the various parameters is as follows:

:PROG MICRO,P_1,P_2,P_3,P_4

where P_1 = the logical unit number of the source input device

P_2 = the logical unit number of the DOS list device

P_3 = the logical unit number of the DOS punch device

P_4 = the maximum number of lines per page for the source microprogram listing

All four parameters are optional and positional. A parameter is omitted by entering the bounding commas in two successive character positions.

The default values for the "P" parameters are as follows:

$P_1 = 5$

$P_2 = 6$

$P_3 = 4$

$P_4 = 56$

The DOS version of the HP Microassembler has a different set of assembler control statements than the BCS version. For the DOS version, this section should be used instead of section 5 of the *HP 2100 Computer Microprogramming Software* manual (02100-90133).

The eight statements described in this section control the assembly process. Each assembler control statement must begin in character position 1 and (except as shown for the $EXTERNALS statement) should not contain embedded blanks.

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

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 micro-program.

If an erroneous assembler control statement is detected, the Micro-assembler prints ERROR 0002 and the erroneous statement on the DOS list device and then proceeds with the assembly process (except for being printed, the erroneous statement is ignored). If necessary, the programmer should correct the erroneous statement at the completion of the assembly and then reassemble the microprogram.

The assembler control statements are as follows:

$DEBUG

This statement is included merely to make the object microcode compatible with the BCS version of the HP Micro Debug Editor. Refer to the *HP 2100 Computer Microprogramming Software* handbook for

the various implications of $DEBUG. If the object code is to be debugged using the DOS version of the editor, the $DEBUG statement serves no purpose.

$END

This statement signals the end of the source microprogram.

$EXTERNALS=name$_1$␣address$_1$, . . . ,name$_n$␣address$_n$

␣ is a space, *name*$_1$ through *name*$_n$ are symbols, and *address*$_1$ through *address*$_n$ are one to four digit octal control store addresses. If any of the addresses consist of fewer than four digits, the Microassembler automatically interprets them as being right-justified with zero-padding to the left. This statement assigns symbolic addresses to control store addresses which are external to the microprogram being assembled. 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.

$FILE=filename

This statement specifies the name of the disc file into which the object microcode is to be written. The disc file must previously have been created using a :STORE,B,filename directive.

$NOLIST

This statement prevents listing output from being sent to the DOS list device (the printing of warning and fatal error messages is not affected).

$NOPUNCH

This statement prevents binary output from being sent to the DOS punch device.

$ORIGIN=xxxx

xxxx is a one to four digit octal control store address. If the address consists of fewer than four digits, the Microassembler automatically interprets it as being right-justified with zero-padding to the left. This statement sets the program location counter in the Microassembler to the specified value. If the source microprogram contains more than one $ORIGIN statement, the specified control store addresses must occur in ascending order.

$SUPPRESS

This statement suppresses the printing of warning messages (the printing of fatal error messages is not affected).

The HP DOS Micro Debug Editor is a program that makes it possible for the microprogrammer to load object microcode into a Writable Control Store (WCS) module, debug WCS-resident microcode through the use of breakpoints, and generate a set of six mask paper tapes that can be used for "burning" a set of programmable ROM chips. The editor is designed to operate in a minimum DOS environment.

Specifically, the microprogrammer can do any of the following:

- Read an object microprogram from either a disc file or an object tape into a core memory buffer.
- Set a breakpoint in the core memory buffer.
- Move object microcode from the core memory buffer to a WCS module.
- Execute WCS-resident microcode.
- Alter the contents of any or all of the machine registers.
- Move object microcode from a WCS module to the core memory buffer.
- Display the contents of any WCS word in the core memory buffer on the system console device.
- Alter any WCS word in the core memory buffer.
- Move the contents of the core memory buffer to a disc file or punch it on tape.

To debug a microprogram using the Micro Debug Editor, the microprogrammer must previously have loaded an assembly language initialization program, named TEST. The initialization program is described as a separate topic later in this section. Briefly, however, it is used at the start of debug execution to pass parameters and control to the microprogram that is being debugged.

HP DOS MICRO DEBUG EDITOR COMMANDS

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

There are thirteen Micro Debug Editor commands. They are shown in table 9-1. In all cases except MOVE, the first character of the command name is sufficient to identify the command to the editor (for example, to terminate a Micro Debug Editor run, the microprogrammer may enter F, FI, FIN, FINI, FINIS, or FINISH). The command name MOVE may not be abbreviated.

INPUT COMMANDS

The input commands are:

LOAD[,x]

READ,x

LOAD[,x] *Load core buffer from device.*

Table 9-1. HP DOS 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]	
	VERIFY[,x]	
Termination Command:	FINISH	
Debug Commands:	BREAK,yyyy	
	CHANGE[,mnemonic]	
	EXECUTE[,0 or yyyy]	
	MOVE,xxxx	

x is the logical unit number of the system mass storage device (x=2) or of a punched tape reader. This command reads the object microcode from the specified device into the core memory buffer. If the system mass storage device is specified, the editor prints the message ENTER FILENAME on the system console device. The microprogrammer responds by entering the name of the disc file which contains the object microcode to be debugged (this name is the one supplied at assembly time using the $FILE assembler control statement).

If *x* is omitted, it is assumed to be 5.

READ,x *Load core buffer from WCS.*

x is the logical unit number of a WCS module. This command reads the contents of the specified WCS module into the core memory buffer.

EDIT COMMANDS

The edit commands are:

SHOW,xxxx[,yyyy]

MODIFY,xxxx[,yyyy]

SHOW,xxxx[,yyyy] *Display core buffer location(s).*

xxxx and *yyyy* are octal control store addresses (0-1777). *xxxx* is the address of the first WCS word to be displayed and *yyyy* is the address of the final WCS word to be displayed. If omitted, *yyyy* is assumed to be the same as *xxxx*. If the microprogrammer enters fewer than four digits for *xxxx* or *yyyy*, the value entered is automatically interpreted as being right-justified with zero-padding to the left. Note that the editor uses only the rightmost eight bits of *xxxx* and *yyyy* ($0\text{-}377_8$).

The SHOW command displays the specified buffer-resident WCS words on the system console device in the following format:

aaaa mmm nnnnnn

where *aaaa* is the control store address (0-1777) of the WCS word being displayed, *mmm* is the octal representation of bits 23-16 of the WCS word, and *nnnnnn* is the octal representation of bits 15-0 of the WCS word.

MODIFY,xxxx[,yyyy] *Alter core buffer location(s).*

xxxx and *yyyy* are octal control store addresses (0-1777). *xxxx* is the address of the first WCS word to be modified and *yyyy* is the address of the final WCS word to be modified. If omitted, *yyyy* is assumed to be the same as *xxxx*. If the microprogrammer enters fewer than four digits for *xxxx* or *yyyy*, the value entered is automatically interpreted as being right-justified with zero-padding to the left. Note that the editor uses only the rightmost eight bits of *xxxx* and *yyyy* ($0\text{-}377_8$).

The MODIFY command allows the microprogrammer to alter the specified buffer-resident WCS words.

In response to the MODIFY command, the editor prints the following on the system console device:

aaaa mmm nnnnnn<=

where *aaaa* is the control store address (0-1777) of the WCS word to be altered, *mmm* is the octal representation of bits 23-16 of the WCS word, and *nnnnnn* is the octal representation of bits 15-0 of the WCS word.

The microprogrammer then enters:

mmm,nnnnnn

where *mmm* is the octal representation of the desired state of bits 23-16 of the WCS word and *nnnnnn* is the octal representation of the desired state of bits 15-0 of the WCS word. If the microprogrammer enters fewer than three digits for *mmm*, or fewer than six digits for *nnnnnn*, the value entered is automatically interpreted as being right-justified with zero-padding to the left. If it is desired to leave *mmm* or *nnnnnn* unchanged, the microprogrammer enters an asterisk instead of the respective octal number.

Examples:

*,123456 means that bits 23-16 of the WCS word are not to be modified and bits 15-0 are to be set to the value 123456_8.

123,* means that bits 23-16 of the WCS word are to be set to the value 123_8 and bits 15-0 are not to be modified.

6,123 is equivalent to entering 006,000123.

If the microprogrammer specifies that a series of WCS words are to be modified, the editor responds by printing *aaaa mmm nnnnnn*<= for the next WCS word in the series, and so forth. If the microprogrammer does not wish to modify a particular WCS word in the series, he enters *,*.

Instead of entering the value as described above, the microprogrammer may enter the octal equivalent for the individual fields of the WCS word. The format of this type of entry is as follows:

$$Fd_1\text{-}d_2\text{-}d_3\text{-}d_4\text{-}d_5\text{-}d_6$$

where d_1 through d_6 are the octal equivalents for the R-bus, S-bus, Function, Store, Special, and Skip fields. For example, to modify a WCS word so that it contains the microinstruction:

F S1 DEC M CW NMPV

the microprogrammer would enter the following character string:

F3-13-6-1-14-12

After the last specified WCS word has been modified, the editor prints COMMAND? on the system console device.

Note that the MODIFY command and the associated entries modify the buffer-resident WCS words (not the actual WCS module locations). To update the WCS module to the revised state, the microprogrammer must move the contents of the core memory buffer to the WCS module (using either the WRITE or EXECUTE command).

OUTPUT COMMANDS

The output commands are:

DUMP[,x]

PREPARE[,x]

VERIFY[,x]

WRITE,x

DUMP[,x] *Dump core buffer to device.*

x is the logical unit number of the system mass storage device (x=2) or of a tape punch. This command sends the contents of the core memory buffer to the specified device. If the system mass storage device is specified, the editor prints the message ENTER FILENAME on the system console device. The microprogrammer responds by entering the name of an existing disc binary data file (previously created using a :STORE,B,filename directive). If *x* is the logical unit number of a tape punch, the object microcode is punched in the same format as an object tape produced by the HP Microassembler.

If *x* is omitted, it is assumed to be 4.

PREPARE[,x] *Punch mask tapes.*

x is the logical unit number of a tape punch. This 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 microprogrammer to enter the tape's I.D. header information. The microprogrammer may then enter up to three lines of information (any characters). For tapes two through six, the microprogrammer is given the option of duplicating the I.D. lines used on the previous tape. A mask tape contains the same four bits of every WCS word. The first tape punched contains bits 23 through 20 of all WCS words, the second contains bits 19 through 16, the third contains bits 15 through 12, the fourth contains bits 11 through 8, the fifth contains bits 7 through 4, and the sixth contains bits 3 through 0.

If *x* is omitted, it is assumed to be 4.

VERIFY[,x] *Verify mask tapes.*

x is the logical unit number of a tape reader device. This command reads a mask tape through the specified device and compares the contents of the tape against the contents of the core memory buffer. In response to the 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.

If no errors are detected during the verify process, the editor prints COMMAND? on the system console device and the microprogrammer proceeds with the next command. If errors are detected, the editor prints

BAD MASK TAPE
DO YOU WANT TO REPUNCH THIS TAPE?

The microprogrammer responds by entering Y or N. If he enters N, the editor prints COMMAND? and the microprogrammer proceeds with the next command. If he enters Y, the editor prints ENTER PUNCH LOGICAL UNIT # and the microprogrammer enters the logical unit number of the tape punch. The editor then asks the microprogrammer to enter up to three lines of alphanumeric tape I.D. information (any characters), repunches the tape, and prints COMMAND? on the system console device.

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

WRITE,x *Dump core buffer to WCS.*

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.

TERMINATION COMMAND

The termination command is:

FINISH

FINISH *Terminate Micro Debug Editor run.*

This command terminates the current Micro Debug Editor run.

DEBUG COMMANDS

The debug commands are:

BREAK,xxxx

CHANGE[,mnemonic]

EXECUTE[,0 or ,xxxx]

MOVE,xxxx

BREAK,xxxx *Set breakpoint in core buffer.*

xxxx is an octal control store address ($0\text{-}1777_8$). If the microprogrammer enters fewer than four digits for *xxxx*, the value entered is automatically interpreted as being right-justified with zero-padding to the left. Note that the editor uses only the rightmost eight bits of *xxxx* ($0\text{-}377_8$).

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

Breakpoints should be set only where JMP microinstructions are 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 microprogrammer.

Since the editor's load/dump routine uses core memory location 0 for temporary storage, the microprogrammer should be aware that every time a breakpoint is executed the contents of that location are destroyed. Also, since the load/dump routine occupies control store locations 272_8 through 377_8, the microprogrammer cannot set a breakpoint above control store location 271_8 unless the load/dump routine is first moved through the use of the MOVE command. The Micro Debug Editor does not allow the microprogrammer to set a breakpoint within the bounds of the load/dump routine no matter where the routine resides.

The load/dump routine executes an EOP micro-order. Among other things, the EOP clears the JSB flip-flop. Consequently, if the breakpoint occurred within a subroutine, execution *must not* be resarted within the subroutine because the RSB at the end of the subroutine will not function 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.

CHANGE[,mnemonic] *Alter register(s).*

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 machine registers and flip-flops.

If the microprogrammer includes a mnemonic in the CHANGE command, 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 microprogrammer then enters an octal number representing the desired contents of the register or flip-flop.

If the microprogrammer enters a CHANGE command with no mnemonic, the editor assumes that he wishes to alter the contents of all the machine registers and flip-flops. In this case, the above conversational process is performed for each register and flip-flop. If the microprogrammer 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 ,xxxx] *Dump core buffer to WCS and execute.*

xxxx is an octal control store address (0-1777). If the microprogrammer enters fewer than four digits for *xxxx*, the value entered is automatically interpreted as being right-justified with zero-padding to the left. Note that the editor uses only the rightmost eight bits of *xxxx* ($0\text{-}377_8$).

The EXECUTE command causes the contents of the core memory buffer to be written into a WCS module and then executes the WCS-resident microcode. If the microprogrammer has previously used a WRITE command, the EXECUTE command automatically uses the same WCS module referenced by the WRITE command. If the microprogrammer has *not* previously used a WRITE command, the editor responds to the EXECUTE command by asking for the logical unit number of the WCS module.

EXECUTE,0 causes the WCS-resident microcode to be executed from the beginning by way of the initialization program. The first time the EXECUTE command is used during a Micro Debug Editor run, only the form EXECUTE,0 is accepted by the editor. Once the EXECUTE,0 has been used, all three forms of the command are acceptable.

EXECUTE causes the WCS-resident microcode to be executed from the point where it was last interrupted by a breakpoint.

EXECUTE,xxxx causes the WCS-resident microcode to be executed starting at the specified control store address.

MOVE,xxxx *Relocate load/dump routine within WCS.*

xxxx is a one to four digit octal control store address (0-1777). If the microprogrammer enters fewer than four digits for *xxxx*, the value entered is automatically interpreted as being right-justified with zero-padding to the left. Note that the editor uses only the rightmost eight bits of *xxxx* ($0\text{-}377_8$). The command mnemonic (MOVE) *may not be abbreviated.*

When an EXECUTE command is entered, the contents of the core memory buffer are copied into the WCS module and the WCS-resident microcode is then executed. However, the Micro Debug Editor's load/dump microprogram is also automatically written into WCS locations 272_8 through 377_8 *after* the contents of the core memory buffer have been copied to the WCS module.

The MOVE command supplies an address to be used instead of 272_8. Thus, the microprogrammer can control where the load/dump routine is located in the WCS module. Note that once a MOVE command is entered, the address supplied is used for the remainder of the Micro Debug Editor run unless the address is altered by another MOVE command.

CAUTION

Be very careful when using this command. If the load/dump routine is accidentally moved so that it overlays the breakpoint or the primary jump table, then the routine will be executed as though it were part of the user's microcode.

THE INITIALIZATION PROGRAM

In order to debug microcode using the Micro Debug Editor, the microprogrammer must supply an initialization program. The initialization program is an assembly language program that performs whatever functions are necessary to call the microprogram which is being debugged (namely, preparing the required parameters in core memory and then executing a 105xxx macro instruction).

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

The only restriction on the microprogram being debugged is that 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).

Table 9-2. Initialization Program

```
ASMB,R,B,L,T
        NAM TEST, 7
        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 INSTRUCTIONS

INITIATING A MICRO DEBUG EDITOR RUN

Refer to the *Disc Operating System* manual (02116-91748).

1. If an initialization program is required, assemble it using the HP DOS Assembler.

2. Using the DOS Relocating Loader, load the HP DOS Micro Debug Editor and (if required) the initialization program. If an initialization program is not required, the operator must force

the loading of the editor even though there are two undefined external symbols (TEST and MACRO) by entering :GO,1 in response to the message UNDEFINED EXTS.

3. If requested to do so by the :PROG,LOADR directive, the loader prints a list of entry point addresses for each program and a load map.

4. After the loader prints the message LOADING COMPLETE on the system console device, enter the DOS directive :PROG,MDE. The editor responds by printing COMMAND? on the system console device.

DEBUGGING A SMALL MICROPROGRAM

These operating instructions apply when the microprogram being debugged is smaller than 272_8 locations. The appropriate Micro Debug Editor command mnemonic is shown in parentheses whenever the associated command is used.

1. Assemble the microprogram using the HP DOS Microassembler.

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

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

4. Set a breakpoint (BREAK).

5. Enter EXECUTE,0. 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 microprogrammer performs conversational editing (SHOW, MODIFY) and/or alters the contents of any or all of the machine 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,0 or EXECUTE,xxxx).

   - EXECUTE restarts execution from the point where it was interrupted.
   - EXECUTE,0 restarts execution from the beginning by way of the initialization program.
   - EXECUTE,xxxx restarts execution at the specified control store address.

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

DEBUGGING A LARGE SINGLE-MODULE MICROPROGRAM

These operating instructions apply when the microprogram being debugged is larger than 272_8 locations but still fits into one WCS module. The appropriate Micro Debug Editor command mnemonic is shown in parentheses whenever the associated command is used.

1. Assemble the microprogram using the HP DOS Micro-assembler.

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

3. Read the Microassembler output into core memory (LOAD). Debug the lower half of the microprogram as described in steps 4 through 9.

4. Set a breakpoint (BREAK).

5. Enter EXECUTE,0. 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 microprogrammer performs conversational editing (SHOW, MODIFY) and/or alters the contents of any or all of the machine registers and flip-flops (CHANGE). However, this is also the logical point at which the microprogrammer would initiate the debugging of the upper half of the microprogram (step 10).

7. Set another breakpoint (BREAK).

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

   - EXECUTE restarts execution from the point where it was interrupted.
   - EXECUTE,0 restarts execution from the beginning by way of the initialization program.
   - EXECUTE,xxxx restarts execution at the specified control store address.

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

10. Using the MOVE command, relocate the Micro Debug Editor's load/dump routine to the lower half of the WCS module. Debug the upper half of the microprogram as described in steps 11 through 16.

11. Set a breakpoint (BREAK).

12. Enter EXECUTE,xxxx where *xxxx* is the address of the first control store location of the upper half of 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.

13. Enter any Micro Debug Editor commands

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

14. Set another breakpoint (BREAK).

15. Restart execution (EXECUTE or EXECUTE,xxxx).

    - EXECUTE restarts execution from the point where it was interrupted.
    - EXECUTE,xxxx restarts execution at the specified control store location.

16. When the breakpoint is encountered, repeat steps 13 through 15.

DEBUGGING A MULTI-MODULE MICROPROGRAM

These operating instructions apply when the entire microprogram to be debugged requires more than one WCS module. 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 fits into a WCS module, each is entered using the same 105xxx macro instruction, and each runs independently of the others. Assemble the segments separately using the HP DOS Microassembler.

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

Segment #1

3. Load and debug segment #1 using either of the previously-discussed methods of debugging (as appropriate).

   The final breakpoint should be set after the last executable instruction in the segment. When that breakpoint is encountered, proceed with the debugging of segment #2.

Segments #2 Through x

4. Load and debug the segment using either of the previously-discussed methods of debugging (as appropriate).

   The final breakpoint should be set after the last executable instruction in the segment. When that breakpoint is encountered, proceed with the debugging of the next segment.

PUNCHING MASK TAPES

These operating instructions apply when it is desired to generate a set of six mask tapes to be used for "burning" a set of programmable ROM chips. The appropriate Micro Debug Editor command mnemonic is shown in parentheses whenever the associated command is used.

1. Assemble the microprogram using the HP DOS Microassembler.

2. Load the HP DOS Micro Debug Editor and (if it is desired to debug the microprogram) the initialization program.

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

4. If desired, debug the microprogram as described earlier in this section. When debugging is complete, *do not enter a FINISH command.* Instead, punch the mask tapes by using the PREPARE command.

5. Verify each mask tape as follows:

   a. Load the mask tape into the tape photoreader.

   b. Enter a VERIFY command.

   c. If the tape contains no errors, load the next tape into the tape photoreader and enter another VERIFY command, and so forth.

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

6. After all the mask tapes have been verified, terminate the run (FINISH).

LOADING A WCS MODULE

These operating instructions apply when it is desired to transfer object microcode (in the form of either a disc file or punched tape) to a WCS module. The appropriate Micro Debug Editor command mnemonic is shown in parentheses whenever the associated command is used.

1. Assemble the microprogram using the HP DOS Microassembler.
2. Load the HP DOS Micro Debug Editor.
3. Read the Microassembler output into core memory (LOAD).
4. Move the object microcode from the core memory buffer to the WCS module (WRITE).
5. Terminate the run (FINISH).

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 DOS 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 the logical unit 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 (see figure 11-1). Bits 8-15 of the first buffer word of each pair contains the relative address ($000\text{-}377_8$) of the WCS location to be written into. Bits 0-7 of the same buffer word contain bits 16-23 of the WCS word. Bits 0-15 of the second buffer word of each pair contain bits 0-15 of the WCS word. 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).

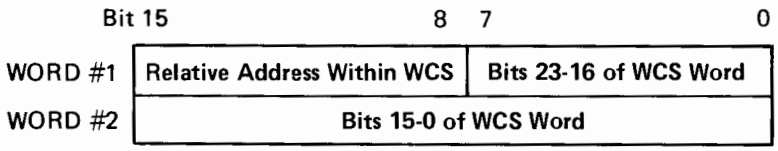


Figure 11-1. *WCS Word Core Memory Format*

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 as 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 the logical unit 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.

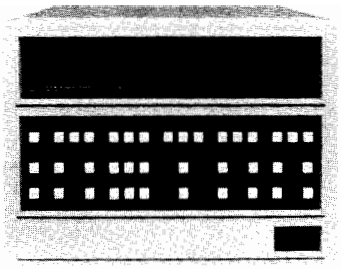
The ALGOL calling sequence for moving object microcode from a WCS module to a core memory buffer is:

```
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 as described for FORTRAN, above.

INDEX (Continued)

INDEX (Continued)



- **POWERFUL HARDWARE**
  A proven architecture implemented by a microprocessor in the heart of the control section.
- **EXPANDABLE MAINFRAME MEMORY**
  Lets you choose up to 32K *all in mainframe.*
- **STANDARD FEATURES**
  2100A and 2100S both include extended arithmetic instructions, power fail interrupt, memory parity check, and memory protect (2100S also includes floating point instructions, two-channel DMA, a crystal-controlled programmable time base generator, and a buffered teleprinter communications channel).
- **FLEXIBLE INPUT/OUTPUT**
  2100A has 14 internal I/O channels, externally expandable to 45; 2100S has 12, expandable to 43.
- **FULL INTERRUPT SYSTEM**
  Interrupt priority easily established or changed for all devices.
- **COMPREHENSIVE SOFTWARE**
  Proven software packages for generating and executing your programs.

2100 computers

The Hewlett-Packard 2100A and 2100S are general-purpose digital computers designed for a wide range of small computer applications.

Standard features of the 2100A include extended arithmetic instructions, power fail interrupt with automatic restart, memory parity check with interrupt, and memory protect. Available as options are dual-channel direct memory access (DMA), floating point hardware, multiplexed input/output (I/O), communications channels accommodating a variety of I/O speeds and devices, writable control store (WCS) modules, a programmable ROM writer, and a full line of systems peripherals and I/O interfaces.

Standard features of the 2100S include extended arithmetic instructions, power fail interrupt with automatic restart, memory parity check with interrupt, memory protect, dual-channel DMA, floating point hardware, crystal-controlled programmable time base generator, and a buffered teleprinter communications channel. Available as options are multiplexed I/O, communications channels accommodating a variety of I/O speeds and devices, WCS modules, a programmable ROM writer, and a full line of systems peripherals and I/O interfaces.

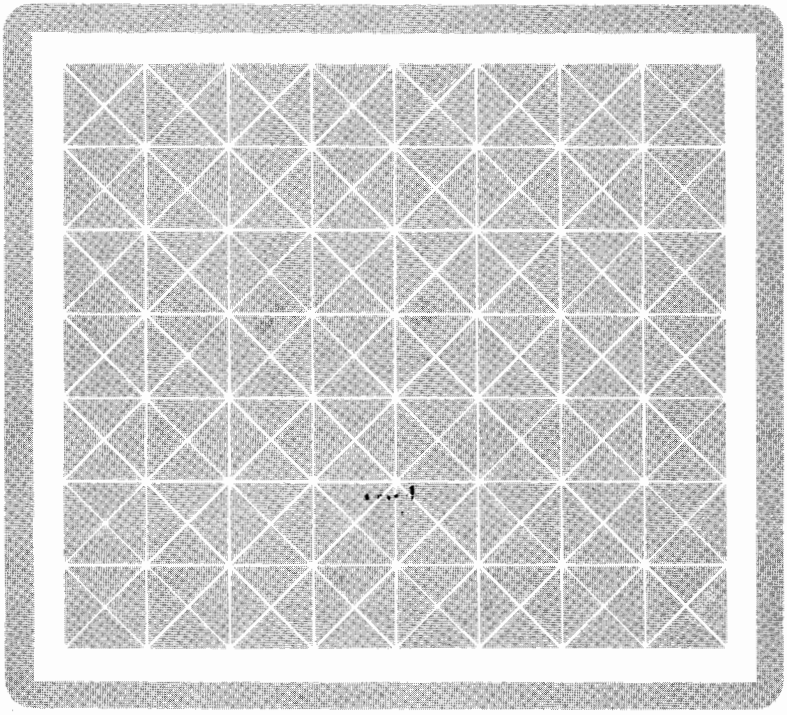
Under DMA control, data can be transferred to or from the computer memory at rates greater than one million sixteen-bit words per second. The floating point hardware typically provides a ten-fold speed increase for scientific, compute-bound algorithms.

A minimum 2100A includes 4,096 words of core memory, self-contained power supply, and 14 I/O channels. A minimum 2100S includes 16,384 words of core memory, self-contained power supply, and 12 I/O channels. The core memory size of each may be expanded to 32,768 words, all in the mainframe. Through the use of an HP 2155A I/O Extender Unit, another 31 I/O channels and power supply can also be added to each.

The 2100A and 2100S have a comprehensive range of proven software packages, including assemblers, compilers, operating systems, and subroutines.

In addition to the above-mentioned capabilities, you can depend on the HP reputation for high quality and world-wide customer support. The net result is a cost-effective computer which meets your data processing needs today and will continue meeting them as your needs expand.

HEWLETT hp PACKARD



02100-90146

Printed in U.S.A. 5/73