1

Utility Programs

9 GENERAL

The utility programs associated with the 8088 Copower Board can be divided into two sections:

- those running under CP/M and
- those running under MS-DOS

The names of these utilities and their related files are given in the following table:

CP/M Utilities	Related Files	
MSUTIL	MSUTIL.COM	
	MSUTIL.MSG	
	MSUTILT.MAC	
	MSUTILM.MAC	
TEST88	TEST88.COM	
MSCONF	MSCONF.COM	
	MSCONF.DAT	
	IBMKEY.TAB	) See Note below:
	PHIKEY.TAB	)
	CONFIG.MSG	
MSBOOT	MSBOOT.COM	

MS-DOS Utilities	Related Files
MSSORT/PCSORT	MSSORT.EXE PCSORT.EXE
CONFIG.SYS	CONFIG.SYS
ANSI.SYS	ANSI.SYS
CLEAR SCREEN	CLR.COM
REBOOT	REBOOT.COM
DEBUGGER	DEBUG.COM

Note: MSKEY.TAB = IBMKEY.TAB OR PHIKEY.TAB
depending on keyboard requirements

Utility Programs

9.1 CP/M Utilities

9.1.1 MSUTIL

The utility MSUTIL is an extension of the CP/M utility UTIL and is used to format and copy MS-DOS and CP/M floppy disks, and to transfer files between the two systems. It is also used to configure a hard disk for MS-DOS operation.

The program is driven by menus and prompts. A complete description of the formatting and copying procedures is not required. Points of interest are given in the following paragraphs.

The program is called up, under CP/M, with:

MSUTIL [CR]

and the following menu displayed:

SELECT:

1 - FORMAT FLOPPY DISK

2 - COPY FLOPPY DISK

3 - CONFIGURE HARD DISK FOR MS-DOS

to escape enter <ESC> OR <0>

FORMATTING WITH MSUTIL

All connected drives can be used for formatting. The program will access the drives in the order in which they are entered and continue until an empty drive is found. This will be indicated by the message 'DISK ERROR'.

Successful formatting will be indicated by the message 'READY:n'; where n = the drive number. The formatted disk can then be removed and a new disk inserted for formatting on the next cycle.

The program will format the following:

P2000C DOUBLE SIDED - CP/M or MS-DOS
P2000C 160K - CP/M
P2000M - CP/M
P2500 300K - CP/M
P2500 600K - CP/M
P3500 - CP/M
P5020 320K - CP/M
IBM-PC - CP/M-86 or 360K MS-DOS
RAINBOW 400K - CP/M
KAYPRO 200K - CP/M

It must be pointed out that, in the cases listed below, diskettes formatted with this utility may not be fully compatible with their host systems due to the difference in head widths:

P2000C 160K - CP/M
P2000M - CP/M
IBM-PC - CP/M-86 or 360K MS-DOS

After the whole disk has been formatted, the operating system is checked. If it is MS-DOS, the BPB (BIOS Parameter Block) is written into the first sector. Then two reset FATs (File Allocation Table) and an empty directory are written onto the disk.

With CP/M, no action is taken. It may lead to difficulties if you format IBM PC CP/M-86 disks because they use an extra media identifier byte on the last address of the boot sector. Its value should be 01. To make this correction, load MS-DOS, start DEBUG, put the CP/M-86 disk into drive B and type:

```
L1000,1,0,1 and press carriage return
E 11FF <CR>
E5 01 <CR>
W1000,1,0,1 <CR>
```

COPYING WITH MSUTIL

Selection of SOURCE and DESTINATION disks is made from the following list:

```
P2000C DOUBLE SIDED - CP/M or MS-DOS
P2000C 160K         - CP/M
P2000M              - CP/M
P2500  300K         - CP/M
P2500  600K         - CP/M
P3500               - CP/M
P5020  320K         - CP/M
IBM-PC              - CP/M-86 or 360K MS-DOS
RAINBOW 400K        - CP/M
KAYPRO  200K        - CP/M
```

Utility Programs

The destination disk must be formatted correctly.

If the source and destination disks are of the same type, it is possible to make a physical copy of the source disk by entering 'y' to the prompt:

"Make PHYSICAL (track to track) copy (y/n)?"

This will produce a complete copy, which will include unwanted 'deleted' files.

Entering 'n' to the above prompt - or if the source and destination disks are of different types - will produce a list of the files on the source disk and the possibility to copy all of the files (enter '*'), or a selected file (enter the number to the left of the file name). Leading zeros must be entered.

This method will only copy the actual files and the appropriate directory entries.

If your source disk is MS-DOS and has a subdirectory, you can select this subdirectory by its number. Then it will be displayed and can be selected for copying.

CONFIGURATION OF THE HARD DISK WITH MSUTIL

The P2000C MS-DOS implementation assumes that the hard disk has two partitions, one each for CP/M and MS-DOS.

CP/M is the primary partition and it is activated first in all cases. The MS-DOS partition is located in the upper part of the CP/M range and above it. The address of the MS-DOS partition is found in the CP/M directory under the filename MSDOS.ALL under USER-31 (normally not accessible).

To prevent CP/M writing into the MS-DOS area the directory entry of the MSDOS.ALL file must declare all MS-DOS blocks as occupied.

After entering the hard disk configuration program, the current CP/M implementation is examined to see if it contains one, and only one, configured hard disk volume in the lower range (5 MB low or 2 MB). If not, the message:

'No harddisk in this CP/M configuration' or
'Too many harddisk volumes in this CP/M configuration'

is displayed and the procedure aborts.

If the CP/M configuration contains 5MB high and/or 8MB hard disk volumes, the user is first warned that they will be overwritten. The procedure will only be continued if the user acknowledges the requirement by typing the word:

'YES'.

The user is asked to switch the hard disk on and press any key. The hard disk directory is then checked and if it already has the MSDOS.ALL file the user is warned that the whole MS-DOS part of his hard disk will be overwritten. In this case the user has to type 'YES' to continue the procedure.

The program calculates and displays following values:

- maximum hard disk size (HS)
- CP/M volume size (CS)
- highest allocated CP/M group (HC)

The maximum hard disk space (HS - HC) and the minimum hard disk space (HS - CS) are displayed. The user has to enter the required MS-DOS size (between the minimum and maximum values - see example below) and the program then:

- Puts the MSDOS.ALL file into the CP/M directory
- Configures the appropriate MS-DOS Driver Parameter Block and writes it into the booter sector.
- Writes the booter sector into the first MS-DOS location.
- Writes the first and second File Allocation Tables (FAT).
- Writes an MS-DOS directory with two 'deleted' entries and a 'name' entry.
  The 'deleted' entries are prepared for the IO.SYS and MSDOS.SYS entries and the 'name' entry is 'HARDDISK'.

If any of these steps fail, the reason is displayed together with the message:

'Program aborted, MSDOS not installed'.

Example:

The user has a 10 MB hard disk and the lower CP/M range is 5 MB long. He has already used 2 MBytes in CP/M and wishes to reserve an additional 1 MByte space for CP/M. Thus he will have to select 7 MB as the MS-DOS size.

If all these steps are successful, the user is informed of the actions to take in order to use the hard disk as the MS-DOS booting device. All the details are in Chapter 2.9.

9.1.2 TEST88

The 8088 maintenance program checks the general hardware functions. These include:

- memory behaviour
- all interrupts (in both directions)
- refresh circuit
- common memory access lock
- the 8087 mathematics coprocessor

It is also possible to start a simple 8088 memory debugger from the maintenance program.

Utility Programs

Test Philosophy

The maintenance program is started from the CP/M environment by entering:

TEST88 [CR]

The program first carries out some checks to confirm that the 8088 Copower Board is present and operating. In the case of 'fatal' error, or if the board is not fitted, the following message is displayed:

'NO ANSWER FROM SLAVE BOARD'

This message could indicate that the board has not been fitted correctly. No further testing is possible!

If the initial checks are completed satisfactorily, the following message is displayed:

'IS THE 8087 MATH COPROCESSOR INSTALLED ? [Y or N]'

If Y(es) additional tests will be carried out, during the test procedure, to check the 8087.

On start-up, the user can make the following selections:

LONG MEMORY TEST	-	L
SHORT MEMORY TEST	-	S
INT. & COPROC. TEST	-	I
REFRESH & LOCK TEST	-	R
8088 MEMORY DEBUGGER	-	D
EXIT PROGRAM	-	0 or ESC

Utility Programs

Testing

SHORT TEST:

The short test is described in full.

The first function of the SHORT TEST is a check of the memory. After calculating the checksum of the Z80 memory the 8088 memory is filled with a test pattern.

The test program runs in the common (Z80) memory

The display line shows the letter 'Z', indicating that the program is running in Z80 memory, followed by a dot '.' as each 16K block is filled.

```
Z......
```

The memory contents are then checked.
As each block is checked, one dot is cleared from the right. If an error is found, the character 'e' will be displayed momentarily and the standard error message will be displayed at the end of the test.

The Z80 checksum is then verified. Any error is indicated with the message:

'checksum error:'

If no fault is found in the 8088 memory, the program is moved to the lower addresses in the 8088 memory and the test is repeated on locations above the program. The display line shows the letter 'E', indicating that the program is running in External (8088) memory, followed by a dot '.' as each block is filled.

ZE.....

As each block is checked, one dot is cleared from the right. If an error is found, the character 'e' will be displayed momentarily and the standard error message will be displayed at the end of the test.

The Z80 memory checksum is then verified again; any error is indicated by the display of the message 'checksum error'.

Note: Because of intermediate calculations, a memory error in 8088 memory will also cause a Z80 checksum error to be shown. If both errors are indicated on the same pass the checksum error can be ignored.

Following the memory test the refresh circuitry is checked. This is indicated in the display line as follows:

ZER

All interrupts are then generated in both directions. The display shows:

ZERI

False interrupt numbers are displayed with the common message in the form:

'GGG1XXOO'

This display is explained in the functional description.

On completion, a further pass is started with the memory test.

OTHER TEST ROUTINES:

The LONG test is similar to that described above except that all bit-combinations on all the memory addresses are checked.

The individual INT. & COPROC. TEST (interrupt and coprocessor), and REFRESH & LOCK TEST, check those functions only.

All tests can be terminated by entering 'ESC' or 'O'.

Utility Programs

Functional Description

REFRESH & LOCK TEST:

The program runs in common memory. The first 16K block of the 8088 memory is filled with a pseudo-random pattern and the program locks the 8088 memory. At this stage the 8088 awaits a fetch from common memory and the 8088 memory is not disturbed (no read or write). The refresh circuit is active to preserve memory contents.

After a few seconds the 8088 is released and the memory contents are verified. Any errors indicate a fault in the refresh circuitry or the dynamic memory.

INTERRUPT & COPROCESSOR TEST:

The whole 8088 memory is first filled with 'E5' with the instruction 'IN A, [E5]' to all unused memory locations.

In the event of a program crash the logic analyser can be triggered on an input instruction.

If the 8087 Math Coprocessor is present a very simple test is made at this stage.

The program then generates an interrupt table in the range 0 - 3FFH with a starting address for each 256 interrupt entries. Interrupt service routines are then generated in the range 400H - BFFH, pointed to by the interrupt table.

Each routine loads the register AL with its own interrupt number and jumps to a common routine which:
- sets a memory byte with the contents of AL
- sets a ready flag
- returns

The Z80 CPU generates an interrupt with a special number, waits for the ready flag and then checks the received number.

If the flag is not set within a set time, or if the received number is incorrect, the interrupt number is noted. An error summary is displayed at the end of the test; for example:

PASS 00215 INTERRUPT ERRORS ON : 0010XXXX

This would indicate that interrupts are false on the addresses:

00100000 - 00101111

Utility Programs

MEMORY TESTS:

The memory tests load the memory with a pseudo-random pattern. This is necessary as the sequence must be reproducible for verification.

The base pattern is 96H, inverted for each pass. This gives a base pattern of 96H for odd number passes and 69H for even number passes, allowing each bit to be checked in both directions.
(96H = 10010110: 69H = 01101001).

For the short test, the pattern is XORed with FF and this value is rotated one position left. The result is written to the next memory cell. One permutation is excluded to give a 7 byte cycle:

D2, A5, 4B, 2D, 5A, B4, 69

For the following short test pass, the values (inverted) will be:

2D, 5A, B4, D2, A5, 4B, 96

The long test works in a similar way, except that the XORed value is counted down from FF to 0 after every test cycle. For the first test the base pattern is the same as for the short test (96 XOR FF = 69). For the second cycle 96 XOR FE = 68. The permutation cycle is:

D0, A1, 43, 0D, 1A, 34, 68

For the following long test pass the values are again inverted.

The short test checks only one possibility for each memory cell while the long test checks every possible combination. For this reason the long test requires about 255 times longer than the short test.

The checksum verification tests for any unwanted interaction between the memories or the processors. A checksum error means that the 8088 has written into the common memory, indicating a memory separator circuit defect or processor interference.

In the event of an error the 8088 sends the character 'e', and a summary of any false bit patterns and addresses are displayed in the form:

```
PASS  00123:  ERROR AT ADDRESS:
              0000 11110000  XXXXXXXX:  G10G10XX
               (0)  (F) (0)  (00 - FF)
```

This indicates errors in the range 0F000 - 0F0FF. The bit summary:

```
bit 7 6 5 4 3 2 1 0
    - - - - - - - -
    G 1 0 G 1 0 X X
```

indicates that:
- bits 7 & 4 are always OK (G)
- bits 6 & 3 are stuck at 1 (1)
- bits 5 & 2 are stuck at 0 (0)
- bits 1 & 0 are faulty in both directions (X)

If no errors are found the following pass starts automatically. The display line is cleared but any error messages are scrolled and accumulated on the screen. If a printer is connected, all messages are printed.

The program is not crash-proof as it runs in RAM. If it does not respond to 'ESC' or '0' (to terminate test), a total failure is indicated. The program cannot run in common memory.

8088 MEMORY DEBUGGER:

On entering the debugger, the following prompt message is displayed:

```
'Commands: C,D,F,G,M,S,SK,O,"esc", ?
 Entry: Command Start_address End_address Other_info'
```

The command code may be followed by a space (not necessary). The separator between the operands may be a space or a comma. An automatic caps-lock function is implemented, i.e., lower case characters cannot be entered.

The command line format differs from that in the P2000 Maintenance Program Debugger.

The commands, with examples, are shown below. 'CR' indicates the CARRIAGE RETURN key.

COMMAND C: - COMPARE MEMORY BLOCKS

Example: C 14000 14FFF 5000

This command will compare the block 14000 - 14FFF with the block 5000 - 5FFF. Any differences will be displayed.

COMMAND D: - DISPLAY MEMORY BLOCKS

Example: D 6000 607F

The content of locations 6000 to 607F is displayed (HEX and ASCII). The program stores the last displayed address and the length of the block. Therefore, to see the next block (6080 to 60FF), the user has only to type D and 'CR'. D and 'CR' as an initial entry displays the contents of locations 0000 to 00FF. (0000 is the first address of the default disk buffer.)

A long display may be stopped and restarted with CTRL-S. Any other character cancels the command.

COMMAND F: - FILL MEMORY BLOCK WITH ONE BYTE

Example: F 8233 875D 55

The memory locations 8233 to 875D will be filled with 55H.

COMMAND G: - GO TO AN ADDRESS (CALL A PROGRAM)

Example 1: G 5000

Starts 8088 program at location 5000 and returns immediately to the debugger.

Example 2:

Set up the following program in the 8088 memory from address 00000.

```
  0000 2E                  CS:
  0001 C6 06 0010 12       MOV  BYTE PTR [10],12H
  0006 F4                  HLT
```

In machine code this is:

```
          2E, C6, 06, 10, 00, 12, F4
```

The program will simply set the contents of address 10 to 12H.

See paragraph SET MEMORY !!

```
 S0 'CR'           SET memory from address 00000
 00000 XX 2E 'CR' CS:
 00001 XX C6 'CR'
 00002 XX 6  'CR'
 00003 XX 10 'CR'
 00004 XX 0  'CR'
 00005 XX 12 'CR'
 00006 XX F4 'CR'
 00007 XX .  'CR' quit SET command
```

```
S10 'CR'          SET memory from address 10
00010 XX FF 'CR'  set content of address 10 to FF
00011 XX .  'CR'  quit SET command

G 0 'CR'          start [GO TO] program at
                                      address 0
```

You will now return automatically to the 8088 Memory Debugger

```
S10  'CR'          check content of address 10,
                                   it should be 12H
00010 12  ---------address 10 is set (to 12H)
                                   by the program!!
```

COMMAND M: - MOVE MEMORY BLOCK

Example: M 4000 4FFF 5000

The block 4000 - 4FFF is copied to 5000 - 5FFF.

Utility Programs

COMMAND S: - SET MEMORY

Example: S 4000

The entered start_address and its contents will be displayed.

```
4000  21          - original contents 21H
```

The contents can be altered [enter new value], or left unchanged [enter 'CR']. The next address will then be displayed. To cancel the command, enter '.'

```
4000  21  31      - changes contents to  31H
4001  00  10      - changes contents to  10H
4002  50  CR      - leaves contents  as  50H
4003  32  3A      - changes contents to  3AH
4004  55  .       - cancels command
```

COMMAND SK: - SEEK STRING IN MEMORY

Example: SK CD F9 F6

All addresses where the string 'CD F9 F6' occur are displayed.

COMMAND 0 or "esc": -
EXIT FROM DEBUGGER INTO MENU TABLE

Note: The "0" or "esc" must be followed by CR to close the command line.

9.1.3 MSCONF

This program works in the same way as the normal P2000C configuration program (CONFIG) used in the CP/M environment, except that it uses the data file MSCONF.DAT instead of CONFIG.DAT.

The program is called up, under CP/M, with:

MSCONF [CR]

This utility should only be used when it is required to configure the P2000C in the PC mode. It is used to set up:

- internal codes
- screen codes
- keyboard codes
- printer tables

Use the program CONFIG when the normal P2000C configuration is required.

28

9.1.4 MSBOOT

The MS-DOS bootstrap program is entered with:

MSBOOT [CR]

The MSBOOT program is used to install software interface support drivers so that peripherals can be accessed from the 8088 Copower Board.

The program then attempts to load and execute a system disk for the MS-DOS operating system.

The MSBOOT program can be started automatically using the 'autostart' feature of CP/M-80 and set up during the normal CP/M configuration.

Utility Programs

When booting from a floppy disk it may be found convenient to boot the CP/M system from drive 2 with the MS-DOS system disk in drive 1. This will allow MSBOOT to proceed without having to remove the CP/M system disk.

Information for transferring the MSBOOT program to hard disk is given in the section on MSUTIL.

9.2 MS-DOS Utilities

9.2.1 MSSORT.EXE AND PCSORT.EXE

The appropriate version of the sort program, either MSSORT or PCSORT will be copied to SORT when the installation batch files are used to install the required mode. Run INSTIBM.BAT or INSTPHI.BAT as appropriate.

MSSORT

The program MSSORT.EXE is the normal MS-DOS sort program (SORT.EXE) and has a collating sequence from 0 to 255. It should always be used when in the MS-DOS P2000C mode.

PCSORT

PCSORT.EXE is a variation of the MS-DOS sort program (SORT.EXE) and has a collating sequence designed to suit the IBM PC character codes (for example, "A umlaut" will sort between A and B). This program should be used when in the MS-DOS PC mode.

9.2.2 CONFIG.SYS

This is the file that MS-DOS reads during startup to determine some settings which can be defined by the user. Some default settings have been supplied on the MS-DOS disk in the CONFIG.SYS, INSTIBM.SYS and INSTPHI.SYS files. As these settings may not be appropriate for your particular needs, the CONFIG.SYS file can be edited using the simple line editor EDLIN. After editing it is necessary to reboot MS-DOS before the new settings take effect.

More details of the CONFI.SYS file and how it is used by MS-DOS can be found in Appendix D of the MS-DOS User Guide.

Note: Do not confuse the above "SYS" files with ANSI.SYS which is not a text file.

9.2.3 ANSI.SYS

The program ANSI.SYS is an installable device driver which is used to convert the ANSI escape sequences for controlling the terminal into the control codes and escape sequences required by the P2000C terminal.

This driver does not allow the reassignment of keys or the attachment of text strings to the function keys, as is possible with the IBM PC ANSI driver.

Note: ANSI.SYS is a code file and should not be confused with the other "SYS" files which are text files.

9.2.4 CLR.COM

The program CLR.COM is provided so that a clear screen function is available in the P2000C mode.

The program generates a HEX 0C character and sends it via a BIOS interrupt call to the Z80 terminal software.

The program will not operate correctly if the ANSI device driver is installed.

9.2.5 REBOOT.COM

The program REBOOT.COM is provided to allow a restart of the MS-DOS system without needing to go back to the CP/M environment, and therefore avoids the use of MSBOOT.

When the system is started from a floppy disk the user is prompted as to whether he wishes to load MS-DOS or return to the CP/M environment.

When the system is started from a hard disk the user is not prompted, and the system reloads the MS-DOS immediately. This is because the system is designed to allow an "AUTOSTART" through CP/M and into MS-DOS with no operator intervention other than pressing the RESET switch on the P2000C computer.

9.2.6 MS-DOS DEBUGGER

OVERVIEW OF DEBUG

The DEBUG Utility (DEBUG) is a debugging program that provides a controlled testing environment for binary and executable object files. Note that EDLIN is used to alter source files; DEBUG is EDLIN's counterpart for binary files. DEBUG eliminates the need to reassemble a program to see if a problem has been fixed by a minor change. It allows you to alter the contents of a file or the contents of a CPU register, and then to immediately reexecute a program to check on the validity of the changes.

All DEBUG commands may be aborted at any time by pressing <Ctrl>/<Break>. <Ctrl>/<Num Lock suspends the display, so that you can read it before the output scrolls away. Entering any key other than <Ctrl>/<Break> or <Ctrl>/<Num Lock> restarts the display.

HOW TO START DEBUG

DEBUG may be started two ways. In the first method, you start DEBUG and then enter all commands in response to the DEBUG prompt (a hyphen). In the second method, you enter all commands on the command line used to start DEBUG.

Summary of Methods to Start DEBUG

Method 1	**DEBUG<CR>**
Method 2	**DEBUG [<pathname> [<arglist>]]<CR>**

Method 1: DEBUG

To start DEBUG using method 1, enter:

DEBUG<CR>

DEBUG responds with the hyphen (-) prompt, signaling that it is ready to accept your commands. Since no filename has been specified, current memory, disk sectors, or disk files can be worked on by using other commands.

Warnings

1. When DEBUG (Version 2.0) is started, it sets up a program header at offset 0 in the program work area. You can overwrite the default header if no <pathname> is given to DEBUG. If you are debugging a .COM or .EXE file, however, do not tamper with the program header below address 5CH, or DEBUG will terminate.

2. Do not restart a program after the Program terminated normally message is displayed. You must reload the program with the N and L commands for it to run properly.

Method 2: Command Line

To start DEBUG using a command line, enter:

DEBUG [<pathname> [<arglist>]]<CR>

For example, if a <pathname> is specified, then the following is a typical command to start DEBUG:

DEBUG FILE.EXE<CR>

DEBUG loads FILE.EXE into memory starting at 100 hexadecimal in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed into memory.

An <arglist> may be specified if <pathname> is present. The <arglist> is a list of filename parameters that are to be passed to the program <pathname>. Thus, when <pathname> is loaded into memory, it is loaded as if it had been started with the command:

<pathname> <arglist>

Here, <pathname> is the file to be debugged, and the <arglist> is the rest of the command line that is used when <pathname> is invoked and loaded into memory.

PARAMETERS

All DEBUG commands accept parameters, except the Quit command. Parameters may be separated by delimiters (spaces or commas), but a delimiter is required only between two consecutive hexadecimal values. Thus, the following commands are equivalent:

```
DCS:100 110
D CS:100 110
D,CS:100,110
```

Table 10-1
DEBUG Command Parameters

PARAMETER	DEFINITION
<address>	A two-part designation consisting of either an alphabetic segment register designation or a four-digit segment address, and an offset value. The segment designation or segment address may be omitted, in which case the default segment is used. DS is the default segment for all commands except G, L, T, U, and W, for which the default segment is CS. All numeric values are hexadecimal. For example: CS:0100 04BA:0100 **The colon is required between a segment designation (whether numeric or alphabetic) and an offset.**
<byte>	A two-digit hexadecimal value to be placed in or read from an address or register.
<drive>	A one-digit hexadecimal value to indicate which drive a file is loaded from or written to. The valid values are 0-3. These values designate the drives as follows: 0=A:, 1=B:, 2=C:, 3=D:.
<list>	A series of <byte> values or of <string>s. <list> must be the last parameter on the command line. Example: FCS:100 L8 42 45 52 54 41 'ABC'

Table 10-1 (Continued)
DEBUG Command Parameters

PARAMETER	DEFINITION
<range>	An address range specifying a lower and upper address. Two formats can be used: <address> <address> or <address> L <value> where <value> is the number of bytes in hex to be processed. Example: CS:100 110 CS:100 L 10 The following is illegal: CS:100 CS:110 ^ Error The limit for <range> is 10000 hex. To specify a <value> of 10000 hex within four digits, enter 0000 (or 0).
<sector> <sector>	1- to 3-digit hexadecimal values used to indicate the starting relative sector on the disk and the number of disk sectors to be written or loaded. Relative sectors are obtained by counting the sectors on the disk surface. The first sector (relative sector 0) is at track 0, sector 1, head 0. Numbering continues for each sector on that track and head, and then continues with the first sector on the next head of the same track. When all the sectors on all heads of that track have been counted, numbering continues with the first sector on head 0 of the next track.
<string>	Any number of characters enclosed in quote marks. Quote marks may be either single (') or double("). If the delimiter quote marks must appear within a <string>, the quote marks must be doubled. For example, the following strings are legal: 'This is a "string" is okay.' 'This is a ''string'' is okay.' However, this string is illegal: 'This is a 'string' is not.'

Table 10-1 (Continued)
DEBUG Command Parameters

PARAMETER	DEFINITION
	Similarly, these strings are legal: "This is a 'string' is okay." "This is a ""string"" is okay." However, this string is illegal: "This is a "string" is not." Note that the double quote marks are not necessary in the following strings: "This is a ''string'' is not necessary." 'This is a ""string"" is not necessary.' The ASCII values of the characters in the string are used as a <list> of byte values.
<value>	A hexadecimal value up to four digits used to specify a port number, numbers to be added or subtracted (Hexarithmetic command), number of bytes, or the number of times a command should repeat its functions.

COMMANDS

Each DEBUG command consists of a single letter followed by parameters. Additionally, the control characters and the special editing functions described in Chapter 6 apply inside DEBUG.

If a syntax error occurs in a DEBUG command, DEBUG reprints the command line and indicates the error with an up-arrow (^) and the word Error."

For example:

```
DCS:100 CS:110
        ^ Error
```

Any combination of upper-case and lower-case letters may be used in commands and parameters.

The DEBUG commands are summarized in Table 10-2 and are described in detail on the following pages.

Table 10-2
DEBUG Commands

DEBUG Command	Function
A[<address>]	Assemble
C<range> <address>	Compare
D[<range>]	Dump
E<address> [<list>]	Enter
F<range> <list>	Fill
G[=<address> [<address>...]]	Go
H<value> <value>	Hex
I<value>	Input
L[<address> [<drive> <sector> <sector>]]	Load
M<range> <address>	Move
N<filename> [<filename>]	Name
O<value> <byte>	Output
Q	Quit
R[<register-name>]	Register
S<range> <list>	Search
T[=<address>] [<value>]	Trace
U[<range>]	Unassemble
W[<address> [<drive> <sector> <sector>]]	Write

Utility Programs

A (Assemble) Command

Purpose Assembles 8086/8087/8088 mnemonics directly into memory.

Format A[<address>]

Comments If a syntax error is found, DEBUG responds with

```
^Error
```

and redisplays the current assembly address.

All numeric values are hexadecimal and must be 1-4 characters. Prefix mnemonics must be specified in front of the opcode to which they refer. They may also be entered on a separate line.

The segment override mnemonics are CS:, DS:, ES:, and SS:. The mnemonic for the far return is RETF. String manipulation mnemonics must explicitly state the string size. For example, use MOVSW to move word strings and MOVSB to move byte strings.

The assembler automatically assembles short, near, or far jumps and calls, depending on byte displacement to the destination address. These may be overridden with the NEAR or FAR prefix. For example:

```
0100:0500 JMP   502          ; a 2-byte short jump
0100:0502 JMP   NEAR 505     ; a 3-byte near jump
0100:0505 JMP   FAR  50A     ; a 5-byte far jump
```

The NEAR prefix may be abbreviated to NE, but the FAR prefix cannot be abbreviated.

DEBUG cannot tell whether some operands refer to a word memory location or to a byte memory location. In this case, the data type must be explicitly stated with the prefix "WORD PTR" or "BYTE PTR". Acceptable abbreviations are "WO" and "BY". For example:

```
NEG     BYTE PTR [128]
DEC     WO [SI]
```

A (Assemble) Command

DEBUG also cannot tell whether an operand refers to a memory location or to an immediate operand. DEBUG uses the common convention that operands enclosed in square brackets refer to memory. For example:

```
MOV     AX,21           ; Load AX with 21H
MOV     AX,[21]         ; Load AX with the
                        ; contents of memory
                        ; location 21H
```

Two popular pseudo-instructions are available with Assemble. The DB opcode assembles byte values directly into memory. The DW opcode assembles word values directly into memory. For example:

```
DB      1,2,3,4,"THIS IS AN EXAMPLE"
DB      'THIS IS A QUOTE: "'
DB      "THIS IS A QUOTE: '"
DW      1000,2000,3000,"BACH"
```

Assemble supports all forms of register indirect commands. For example:

```
ADD     BX,34[BP+2].[SI-1]
POP     [BP+DI]
PUSH    [SI]
```

All opcode synonyms are also supported. For example:

```
LOOPZ   100
LOOPE   100
JA      200
JNBE    200
```

For 8087 opcodes, the WAIT or FWAIT must be explicitly specified. For example:

```
FWAIT FADD ST,ST(3)     ; This line will assemble
                        ; a FWAIT prefix
LD TBYTE PTR [BX]       ; This line will not
```

Utility Programs

C (Compare) Command

Purpose Compares the portion of memory specified by <range> to a portion of the same size beginning at <address>.

Format C<range> <address>

Comments If the two areas of memory are identical, there is no display and DEBUG returns with the (-) prompt. If there are differences, they are displayed in this format:

<address1> <byte1> <byte2> <address2>

If only an offset is entered for <range>, the C command assumes the segment contained in the DS register.

Examples The following commands have the same effect:

C100,1FF 300<CR>

or

C100L100 300<CR>

Each command compares the block of memory from 100 to 1FFH with the block of memory from 300 to 3FFH. The segment in the DS register is used by default.

Utility Programs

D (Dump) Command

Purpose Displays the contents of the specified region of memory.

Format D[<range>]

Comments If a range of addresses is specified, the contents of the range are displayed. If the D command is entered without parameters, 128 bytes are displayed at the first address (DS:100) after the address displayed by the previous Dump command.

The dump is displayed in two portions: a hexadecimal dump (each byte is shown in hexadecimal value) and an ASCII dump (the bytes are shown in ASCII characters). Non-printing characters are denoted by a period (.) in the ASCII portion of the display. Each display line shows 16 bytes with a hyphen between the eighth and ninth bytes. At times, displays are split in this manual to fit them on the page. Each displayed line begins on a 16-byte boundary.

Examples If you enter the command:

DCS:100 10F<CR>

DEBUG displays the sixteen memory locations from CS:100 to CS:10F in the following format.

```
04BA:0100 43 61 6E 6E ... 73 70  Cannot open resp
```

If you enter the following command:

D<CR>

the next 128 bytes are displayed. Each line of the display begins with an address, incremented by 16 from the address on the previous line. Each subsequent D (entered without parameters) displays the 128 bytes immediately following those last displayed.

Utility Programs

E (Enter) Command

Purpose Enters byte values into memory at the specified <address>.

Format E<address> [<list>]

Comments If the optional <list> of values is entered, the replacement of byte values occurs automatically. (If an error occurs, no byte values are changed.)

If the <address> is entered without the optional <list>, DEBUG displays the address and its contents, then repeats the address on the next line and waits for your input. At this point, the Enter command waits for you to perform one of the following actions:

1. Replace the byte value with a value you enter. Enter the value after the current value. If the value entered in is not a legal hexadecimal value or if more than two digits are entered, the illegal or extra character is not echoed.

2. Press the <Space Bar> to advance to the next byte. To change the value, enter the new value as described in (1.) above. If you space beyond an 8-byte boundary, DEBUG starts a new display line with the address displayed at the beginning.

3. Enter hyphen (-) to return to the preceding byte. If you decide to change a byte behind the current position, entering the hyphen returns the current position to the previous byte. When the hyphen is entered, a new line is started with the address and its byte value displayed.

4. Press <CR> to terminate the Enter command. The <CR> key may be pressed at any byte position.

Example Assume that the following command is entered:

ECS:100<CR>

Suppose DEBUG displays:

04BA:0100 EB._

To change this value to 41, enter 41 as shown:

04BA:0100 EB.**41**_

E (Enter) Command

To step through the subsequent bytes, press the <Space Bar> three times to see:

```
04BA:0100  EB.41    10.    00.    BC._
```

To change BC to 42, enter 42 as shown:

```
04BA:0100  EB.41    10.    00.    BC.42_
```

Now, realizing that 10 should be 6F, enter the hyphen twice to return to byte 0101 (value 10), then replace 10 with 6F:

```
04BA:0100  EB.41    10.    00.    BC.42-
04BA:0102  00.-_
04BA:0101  10.6F_
```

Pressing <CR> changes any entered values in memory, ends the Enter command, and returns to the DEBUG command level.

Utility Programs

F (Fill) Command

Purpose Fills the addresses in the <range> with the values in the <list>.

Format F<range> <list>

Comments If the <range> contains more bytes than the number of values in the <list>, the <list> is used repeatedly until all bytes in the <range> are filled. If the <list> contains more values than the number of bytes in the <range>, the extra values in the <list> are ignored. If any of the memory in the <range> is not valid (bad or nonexistent), an error occurs in all succeeding locations.

Example Assume that the following command is entered:

F04BA:100 L 100 42 45 52 54 41<CR>

DEBUG fills memory locations 04BA:100 through 04BA:1FF with the bytes specified. The five values are repeated until all 100H bytes are filled.

48

Utility Programs

G (Go) Command

Purpose Executes the program currently in memory.

Format G[=<address> [<address>...]]

Comments If only the Go command is entered, the program executes as if the program had run outside DEBUG.

If =<address> is set, execution begins at the address specified. The equal sign (=) is required, so that DEBUG can distinguish the start =<address> from the breakpoint <address>es.

With the other optional addresses set, execution stops at the first <address> encountered, regardless of that address' position in the list of addresses to halt execution or program branching. When program execution reaches a breakpoint, the registers, flags, and decoded instruction are displayed for the last instruction executed. (The result is the same as if you had entered the Register command for the breakpoint address.)

Up to ten breakpoints may be set. Breakpoints may be set only at addresses containing the first byte of an 8086 opcode. If more than ten breakpoints are set, DEBUG returns the BP error message (see the error message listing at the end of this chapter).

The user stack pointer must be valid and have 6 bytes available for this command. The G command uses an IRET instruction to cause a jump to the program under test. The user stack pointer is set, and the user flags, Code Segment register, and Instruction Pointer are pushed on the user stack. (Thus, if the user stack is not valid or is too small, the operating system may crash.) An interrupt code (0CCH) is placed at the specified breakpoint address(es).

When an instruction with the breakpoint code is encountered, all breakpoint addresses are restored to their original instructions. If execution is not halted at one of the breakpoints, the interrupt codes are not replaced with the original instructions.

G (Go) Command

Example Assume that the following command is entered:

GCS:7550<CR>

The program currently in memory begins with the current instruction (current address of CS:IP) and executes up to the address 7550 in the CS segment. DEBUG then displays registers and flags, after which the Go command is terminated.

After a breakpoint has been encountered, if you enter the Go command again, the program executes just as if you had entered the filename at the DOS command level. The only difference is that program execution begins at the instruction after the breakpoint rather than at the usual start address.

**H (Hex)
Command**

Purpose Performs hexadecimal arithmetic on the two values specified.

Format H<value> <value>

Comments First, DEBUG adds the two parameters, then subtracts the second parameter from the first. The results of the arithmetic are displayed on one line; first the sum, then the difference.

Example Assume that the following command is entered:

H19F 10A<CR>

DEBUG performs the calculations and then displays the result:

02A9 0095

**I (Input)
Command**

Purpose Inputs and displays one byte from the port specified by <value>.

Format I<value>

Comments A 16-bit port address is allowed.

Example Assume that you enter the following command:

I2F8<CR>

Assume also that the byte at the port is 42H. DEBUG inputs the byte and displays the value:

42

L (Load) Command

Purpose Loads a file into memory.

Format L[<address> [<drive> <sector> <sector>]]

Comments If only the L command is entered (no parameters), DEBUG loads into memory starting at location CS:100 the file whose filespec is in the file control block at CS:5C. A file is placed in the file control block by entering the filespec on the command line when DEBUG is started, or by using the NAME command. The L command sets the BX and CX registers to the number of bytes that have been loaded into memory.

If the L command is entered with only the <address> parameter, the file specified in the file control block is loaded into memory at the specified address.

Files with a file extension of .COM are always loaded into memory at location CS:100, regardless of a specified <address> in the L command.

If the file has a .EXE extension, it is relocated to the load address specified in the header of the .EXE file: the <address> parameter is always ignored for .EXE files. The header itself is stripped off the .EXE file before it is loaded into memory. Thus the size of an .EXE file on disk differs from its size in memory.

If the file named by the Name command or specified when DEBUG is started is a .HEX file, then entering the L command with no parameters causes DEBUG to load the file beginning at the address specified in the .HEX file. If the L command includes the option <address>, DEBUG adds the <address> specified in the L command to the address found in the .HEX file to determine the start address for loading the file.

If L is entered with all the parameters, absolute disk sectors are loaded into memory. The sectors are taken from the <drive> specified (the drive designation is numeric where 0=A:, 1=B:, 2=C:, and 3=D:); DEBUG begins loading with the first <sector> specified, and continues until the number of sectors specified in the second <sector> have been loaded.

The maximum number of sectors that can be loaded by an L command is hex 80.

L (Load) Command

Example Assume that the following commands are entered:

```
A>DEBUG<CR>
-NFILE.COM
```

Now, to load FILE.COM, enter:

```
L<CR>
```

FILE.COM is loaded into memory at CS:100 and DEBUG returns the (-) prompt.

The command:

```
LCS:100 1 0F 6D<CR>
```

would load 109 (6D hex) sectors from drive B beginning with logical sector 15 (0F hex) into memory starting at address CS:0100. When the records have been loaded, DEBUG returns the (-) prompt.

Utility Programs

M (Move) Command

Purpose — Moves the block of memory specified by <range> to the location beginning at the <address> specified.

Format — M<range> <address>

Comments — Overlapping moves (moves where part of the block overlaps some of the current addresses) are always performed without loss of data. Addresses that could be overwritten are moved first. The sequence for moves from higher addresses to lower addresses is to move the data beginning at the block's lowest address and then to work towards the highest. The sequence for moves from lower addresses to higher addresses is to move the data beginning at the block's highest address and to work towards the lowest.

Note that if the addresses in the block being moved do not have new data moved to them, the data there before the move remains. The M command copies the data from one area into another, in the sequence described, and writes over the new addresses. This is why the sequence of the move is important.

If only an offset is entered for the starting address of the <range> parameter or for the <address> parameter, the M command uses the segment contained in the DS register.

Example — Assume that you enter:

MCS:100 110 CS:500<CR>

DEBUG first moves address CS:110 to address CS:510, then CS:10F to CS:50F, and so on until CS:100 is moved to CS:500. You should enter the Dump command, using the <address> entered for the M command, to review the results of the move.

-N (Name) Command

Purpose Sets filenames.

Format N<filespec> [<filespec>...]

Comments The N command performs two functions. First, N is used to assign a filename for a later Load or Write command. If you start DEBUG without naming any file to be debugged, the N<filespec> command must be entered before a file can be loaded. Second, N is used to assign filename parameters to the file being debugged. In this case, Name accepts a list of parameters that are used by the file being debugged.

These two functions overlap. Consider the following set of DEBUG commands:

```
-NFILE1.EXE<CR>
-L<CR>
-G<CR>
```

Because of the effects of the N command, Name performs the following steps:

1. (N)ame assigns the filename FILE1.EXE to the file to be used in any later Load or Write commands.

2. (N)ame also assigns the filename FILE1.EXE to the first filename parameter used by any program that is later debugged.

3. (L)oad loads FILE1.EXE into memory.

4. (G)o causes FILE1.EXE to be executed with FILE1.EXE as the single filename parameter (that is, FILE1.EXE is executed as if FILE1.EXE had been entered at the command level).

A more useful chain of commands might look like this:

```
-NFILE1.EXE<CR>
-L<CR>
-NFILE2.DAT FILE3.DAT<CR>
-G<CR>
```

N (Name) Command

Here, Name sets FILE1.EXE as the filename for the subsequent Load command. The Load command loads FILE1.EXE into memory, and then the N command is used again, this time to specify the parameters to be used by FILE1.EXE. Finally, when the Go command is executed, FILE1.EXE is executed as if FILE1 FILE2.DAT FILE3.DAT had been entered at the MS-DOS command level. Note that if a Write command were executed at this point, FILE1.EXE (the file being debugged) would be saved with the name FILE2.DAT! To avoid such undesired results, you should always execute a N command before either a Load or a Write.

There are four regions of memory that can be affected by the N command:

```
CS:5C  FCB for file 1
CS:6C  FCB for file 2
CS:80  Count of characters
CS:81  All characters entered
```

A File Control Block (FCB) for the first filename parameter given to the N command is set up at CS:5C. If a second filename parameter is entered, then an FCB is set up for it beginning at CS:6C. The number of characters entered in the N command (exclusive of the first character N) is given at location CS:80. The actual stream of characters given by the N command (again, exclusive of the character N) begins at CS:81. Note that this stream of characters may contain parameters and delimiters that would be legal in any command entered at the MS-DOS command level.

Example A typical use of the N command is:

```
A>DEBUG PROG.COM<CR>
-NPARAM1 PARAM2/C<CR>
-G<CR>
```

In this case, the Go command executes the file in memory as if the following command line had been entered:

```
PROG PARAM1 PARAM2/C<CR>
```

Testing and debugging therefore reflect a normal runtime environment for PROG.COM.

Utility Programs

O (Output)
Command

Purpose Sends the <byte> specified to the output port specified by <value>.

Format O<value> <byte>

Comments A 16-bit port address is allowed.

Example Enter:

O2F8 4F<CR>

DEBUG outputs the byte value 4F to output port 2F8.

Q (Quit)
Command

Purpose Terminates the DEBUG utility.

Format Q

Comments The Q command takes no parameters and exits DEBUG without saving the file currently being operated on. You are returned to the MS-DOS command level.

Example To end the debugging session, enter:

Q<CR>

DEBUG is terminated and control returns to the MS-DOS command level.

R (Register) Command

Purpose Displays the contents of one or more CPU registers.

Format R[<register-name>]

Comments If no <register-name> is entered, the R command dumps th register save area and displays the contents of all registers and flags.

If a register name is entered, the 16-bit value of that register is displayed in hexadecimal, and then a colon appears as a prompt. You then either enter a <value> to change the register, or press <CR> if no change is wanted.

The only valid <register-name>s are:

AX	BP	SS	
BX	SI	CS	
CX	DI	IP	(IP and PC both refer to the
DX	DS	PC	Instruction Pointer).
SP	ES	F	

Any other entry for <register-name> results in a BR Error message.

If F is entered as the <register-name>, DEBUG displays each flag with a two-character alphabetic code. To alter any flag, enter the opposite two-letter code. The flags are either set or cleared.

The flags are listed below with their codes for SET and CLEAR:

FLAG NAME	SET	CLEAR
Overflow	OV	NV
Direction	DN Decrement	UP Increment
Interrupt	EI Enabled	DI Disabled
Sign	NG Negative	PL Plus
Zero	ZR	NZ
Auxiliary Carry	AC	NA
Parity	PE Even	PO Odd
Carry	CY	NC

R (Register) Command

Whenever you enter the command RF, the flags are displayed in the order shown above in a row at the beginning of a line. At the end of the list of flags, DEBUG displays a hyphen (-). You may enter new flag values as alphabetic pairs. The new flag values can be entered in any order. You do not have to leave spaces between the flag entries. To exit the R command, press <CR>. Flags for which new values were not entered remain unchanged.

If more than one value is entered for a flag, DEBUG returns a DF Error message. If you enter a flag code other than those shown above, DEBUG returns a BF Error message. In both cases, the flags up to the error in the list are changed; flags at and after the error are not.

At startup, the segment registers are set to the bottom of free memory, the Instruction Pointer is set to 0100H, all flags are cleared, and the remaining registers are set to zero.

Example Enter:

R<CR>

DEBUG displays all registers, flags, and the decoded instruction for the current location. If the location is CS:11A, then the display will look similar to this:

```
AX=0E00 BX=00FF CX=0007 DX=01FF SP=039D BP=0000
SI=005C DI=0000 DS=04BA ES=04BA SS=04BA CS=04BA
IP=011A  NV UP DI NG NZ AC PE NC
04BA:011A  CD21           INT     21
```

If you enter:

RF<CR>

DEBUG might display the following flag settings:

NV UP DI NG NZ AC PE NC -

R (Register) Command

To change the settings, enter any valid flag designation, in any order, with or without spaces.

For example:

NV UP DI NG NZ AC PE NC - PLEICY<CR>

DEBUG responds only with the DEBUG prompt. To see the changes, enter either the R or RF command:

RF<CR>

DEBUG displays the new settings:

NV UP EI PL NZ AC PE CY - _

Press <CR> to leave the flags this way.

S (Search) Command

Purpose — Searches the <range> specified for the <list> of bytes specified.

Format — S<range> <list>

Comments — The <list> may contain one or more bytes, each separated by a space or comma. If the <list> contains more than one byte, only the first address of the byte string is returned. If the <list> contains only one byte, all addresses of the byte in the <range> are displayed.

Example — If you enter:

SCS:100 110 41<CR>

DEBUG displays a response similar to this:

04BA:0104
04BA:010D

60

Utility Programs

T (Trace) Command

Purpose Executes one instruction and displays the contents of all registers and flags, and the decoded instruction.

Format T[=<address>] [<value>]

Comments If the optional =<address> is entered, tracing occurs at the =<address> specified. The optional <value> causes DEBUG to execute and trace the number of steps specified by <value>.

The T command uses the hardware trace mode of the 8086 or 8088 microprocessor. Consequently, you may also trace instructions stored in ROM (Read Only Memory).

Example Enter:

T<CR>

DEBUG returns a display of the registers, flags, and decoded instruction for that one instruction. Assume that the current position is 04BA:011A; DEBUG might return the display:

```
AX=0E00 BX=00FF CX=0007 DX=01FF SP=039D BP=0000
SI=005C DI=0000 DS=04BA ES=04BA SS=04BA CS=04BA
IP=011A  NV UP DI NG NZ AC PE NC
04BA:011A  CD21          INT     21
```

If you enter:

T=011A 10<CR>

DEBUG executes sixteen (10 hex) instructions beginning at 011A in the current segment, and then displays all registers and flags for each instruction as it is executed. The display scrolls away until the last instruction is executed. Then the display stops, and you can see the register and flag values for the last few instructions performed. Remember that <Ctrl>/<Num Lock> suspends the display at any point, so that you can study the registers and flags for any instruction.

Utility Programs

U (Unassemble) Command

Purpose Disassembles bytes and displays the source statements that correspond to them, with addresses and byte values.

Format U[<range>]

Comments The display of disassembled code looks like a listing for an assembled file. If you enter the U command without parameters, 20 hexadecimal bytes are disassembled at the first address after that displayed by the previous Unassemble command. If you enter the U command with the <range> parameter, DEBUG disassembles all bytes in the range. If the <range> is given as an <address> only, then 20H bytes are disassembled starting at that <address>.

Example Enter:

```
U04BA:100 L10<CR>
```

DEBUG disassembles 16 bytes beginning at address 04BA:0100 in the following format:

```
04BA:0100  206472      AND   [SI+72],AH
04BA:0103  69          DB    69
04BA:0104  7665        JBE   016B
04BA:0106  207370      AND   [BP+DI+70],DH
04BA:0109  65          DB    65
04BA:010A  63          DB    63
04BA:010B  69          DB    69
04BA:010C  66          DB    66
04BA:010D  69          DB    69
04BA:010E  63          DB    63
04BA:010F  61          DB    61
```

If you enter:

```
U04BA:0100 0108<CR>
```

The display shows:

```
04BA:0100  206472      AND   [SI+72],AH
04BA:0103  69          DB    69
04BA:0104  7665        JBE   016B
04BA:0106  207370      AND   [BP+DI+70],DH
```

If the bytes in some addresses are altered, the disassembler alters the instruction statements. The U command can be entered for the changed locations, the new instructions viewed, and the disassembled code used to edit the source file.

W (Write)
Command

Purpose Writes the file being debugged to a disk file.

Format W[<address> [<drive> <sector> <sector>]]

Comments If you enter the W command with no parameters, the file in memory starting at memory location CS:100 is written to disk using the filespec contained in the file control block at CS:5C. The BX and CX registers must be set to the number of bytes to be written.

If the W command is entered with just the <address> parameter, the file starting at memory location <address> is written to disk using the filespec contained in the file control block at CS:5C. Again, the BX and CX registers must be set to the number of bytes to be written.

Note, if a Go or Trace command has been used, the values placed in the BX and CX registers when the file was loaded might have been changed.

When a file is loaded, edited under DEBUG, and then written back to disk with the same filespec, the edited file is written over the original file.

If the W command is entered with all of the parameters, data is written from memory location <address> to the specified <drive> (the drive designation is numeric, where 0=A:, 1=B:, 2=C:, and 3=D:). DEBUG writes the data to disk beginning at the logical sector number specified by the first <sector> parameter and continues to write until the number of sectors specified in the second <sector> parameter have been written.

WARNING Writing to absolute sectors is EXTREMELY dangerous because the process bypasses the file handler.

W (Write) Command

Examples If you enter:

W<CR>

DEBUG writes the file starting at memory location CS:100 to disk using the filespec contained in the file control block at CS:5C. The number of bytes written to disk are contained in the BX and CX registers. DEBUG then display the DEBUG prompt.

Entering:

WCS:100 1 37 2B<CR>

DEBUG writes the contents of memory beginning at address CS:100 to the disk in drive B:. 2BH sectors of data are written to disk starting at logical sector number 37H.

ERROR MESSAGES

Duringthe DEBUG session,you may receive any of thefollowing errormessages. Each error terminates the DEBUG commandunder which it occurred, but does not terminate DEBUG itself.

Table 10-3
DEBUG Error Codes

ERROR CODE	DEFINITION
BF	Bad flag You attempted to alter a flag, but the characters entered were not one of the acceptable pairs of flag values. See the Register command for the list of acceptable flag entries.
BP	Too many breakpoints You specified more than ten breakpoints as parameters to the G command. Reenter the G command with ten or fewer breakpoints.
BR	Bad register You entered the R command with an invalid register name. See the Register command for the list of valid register names.
DF	Double flag You entered two values for one flag. You may specify a flag value only once per RF command.

Hardware Installation

10 INSTALLATION

10.1 Terminalboard Software

In order to make use of the 8088 Copower Board, it is necessary to have the correct release of the terminalboard software. The Copower Board is compatible with all releases of the terminal-board software from release 1.3 onwards.

To check your terminalboard software release it is simply necessary to press RESET and, before the IPL message is displayed, press the ESC key.

The software release will be displayed as:

TERMINAL SW-Rel: x.x

If the indicated release is lower than 1.3, it will be necessary to replace the terminalboard ROM.
Fitting instructions are given on the following page. Please contact your dealer if you require further information!

10.2 Fitting the Board

With your 8088 Copower Board you will find the P2000C Fitting Instructions for Optional Boards. If you have a copy of this document which was prepared before the 8088 Copower Board was introduced, the method of fitting it is the same as that for the P 2092 Memory Extension Board.

Please follow these instructions carefully.

10.2.1 FITTING THE TERMINALBOARD ROM

The ROM is fitted to a 28 pin I/C socket on the terminal board.

First, remove the sub-chassis as shown in steps 1 and 2 of the Fitting Instructions and identify the existing ROM. This is item 7409 and is located at the bottom right hand corner of the terminal board.

Note the position of the notch on the body of the existing I/C and then carefully remove it. Fit the new I/C, making sure that the pins are not damaged and that the notch is in the same position as on the original.

When the P2000C has been re-assembled, the I/C can be tested by pressing ESC and RESET. The software release number will be displayed and testing carried out.

10.3 System Upgrades

The 8088 Copower Board can be fitted with:

- 2 banks of 64KBit chips or
- 1 or 2 banks of 256KBit chips

These configurations will give:
128, 256 or 512 KBytes of additional RAM.

If the board is upgraded, i.e., if more or larger RAMs are fitted, ensure that all chips are fitted with the same orientation as those originally fitted.

In addition to the RAM, the 8088 is wired to accept an 8087 Mathematics Coprocessor.
Full instructions for fitting this will be supplied by your dealer when the device is purchased.

This maths coprocessor can only be used for application programs that use the special 8087 instructions. Fitting the device will have no effect on normal programs.

10.4 Testing After Installation

A complete check of the 8088 Copower Board can be carried out by running the program TEST88.

The operation of this program is described in Chapter 9 of this manual.

10.5 Restrictions

If 512KBytes of RAM are fitted together with the 8087 math processor, the environmental specification is down-graded and the equipment must not be used with an ambient temperature above 25` C.

In all cases ensure that the P2000C is operated with the tilt-bar down, to allow for efficient air cooling, and that the cooling slots on the top of the machine are not obstructed.