NCR

DEBUG Utility

DEBUG UTILITY CONTENTS

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW OF DEBUG

The Microsoft 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 <CONTROL-C>. <CONTROL-S> suspends the display, so that you can read it before the output scrolls away. Entering any key other than <CONTROL-C> or <CONTROL-S> restarts the display. All of these commands are consistent with the control character functions available at the MS-DOS command level.

1.2 HOW TO START DEBUG

DEBUG may be started two ways. By the first method, you type all commands in response to the DEBUG prompt (a hyphen). By the second method, you type all commands on the line used to start DEBUG.

Summary of Methods to Start DEBUG

Method 1	DEBUG
Method 2	DEBUG [<filespec> [<arglist>]]

1.2.1 Method 1: DEBUG

To start DEBUG using method 1, type:

```
DEBUG
```

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. On previous versions of DEBUG, you could overwrite this header. You can still overwrite the default header if no <filespec> 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.

1.2.2 Method 2: Command Line

To start DEBUG using a command line, type:

```
DEBUG [<filespec> [<arglist>]]
```

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

```
DEBUG FILE.EXE
```

DEBUG then 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 <filespec> is present. The <arglist> is a list of filename parameters and switches that are to be passed to the program <filespec> . Thus, when <filespec> is loaded into memory, it is loaded as if it had been started with the command:

```
<filespec>  <arglist>
```

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

CHAPTER 2 COMMANDS

2.1 COMMAND INFORMATION

Each DEBUG command consists of a single letter followed by one or more parameters. Additionally, the control characters and the special editing functions described in the **MS-DOS User's Guide,** 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 (ˆ) and the word "error."

For example:

```
dcs:100 cs:110
     ^ error
```

Any combination of uppercase and lowercase letters may be used in commands and parameters.

The DEBUG commands are summarized in Table 2.1 and are described in detail, with examples, following the description of command parameters.

Table 2.1 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> <record> <record>]]	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> <record> <record>]]	Write

2.2 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
```

PARAMETER	DEFINITION
<drive>	A one-digit hexadecimal value to indicate which drive a file will be 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:.
<byte>	A two-digit hexadecimal value to be placed in or read from an address or register.
<record>	A 1- to 3-digit hexadecimal value used to indicate the logical record number on the disk and the number of disk sectors to be written or loaded. Logical records correspond to sectors. However, their numbering differs since they represent the entire disk space.
<value>	A hexadecimal value up to four digits used to specify a port number or the number of times a command should repeat its functions.
<address>	A two-part designation consisting of either an alphabetic segment register designation or a four-digit segment address plus 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.

<range<

Two <address>es: e.g., <address> <address>; or one <address>, an L, and a <value>: e.g., <adress> L <value> where <value> is the number of lines the command should operate on, and LB0 is assumed. The last form cannot be used if another hex value follows the <range>, since the hex value would be interpreted as the second <address> of the <range>.
Examples:

```
CS:100 110
CS:100 L 10
CS:100
```

The following is illegal:

```
CS:100 CS:110
        ^
      error
```

The limit for <range> is 10 000 hex. To specify a <value> of 10 000 hex within four digits, type 0000 (or 0).

<list>

A series of <byte> values or of <string>s. <list> must be the last parameter on the command line.

Example:

```
fcs:100 42 45 52 54 41
```

<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.’

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.

NAME	Assemble
PURPOSE	Assembles 8086/8087/8088 mnemonics directly into memory.
SYNTAX	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 entered as 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 will automatically assemble 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
0100:505   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]
```

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 will assemble byte values directly into memory. The DW opcode will assemble 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
                    ; an FWAIT prefix
LD TBYTE PTR [BX]   ; This line will not
```

NAME	Compare
PURPOSE	Compares the portion of memory specified by <range> to a portion of the same size beginning at <address>.
SYNTAX	C<range> <address>

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

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

EXAMPLE The following commands have the same effect:

```
C100,1FF  300
   or
C100L100  300
```

Each command compares the block of memory from 100 to 1FFH with the block of memory from 300 to 3FFH.

NAME Dump

PURPOSE Displays the contents of the specified region of memory.

SYNTAX D[<range>]

COMMENTS If a range of addresses is specified, the contents of the range are displayed. If the D command is typed 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). Nonprinting 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.

If you type the command:

```
dcs:100 110
```

DEBUG displays the dump in the following format:

```
04BA:0100 42 45 52 54 41 . . . 4E 44 TOM SAWYER
```

If you type the following command:

```
D
```

the display is formatted as described above. Each line of the display begins with an address, incremented by 16 from the address on the previous line. Each subsequent D (typed without parameters) displays the bytes immediately following those last displayed.

If you type the command:

```
DCS:100 L 20
```

the display is formatted as described above, but 20H bytes are displayed.
If then you type the command:

```
DCS:100 115
```

the display is formatted as described above, but all the bytes in the range of lines from 100H to 115H in the CS segment are displayed.

NAME	Enter
PURPOSE	Enters byte values into memory at the specified <address>.
SYNTAX	E<address> [<list>]

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

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

1. Replace a byte value with a value you type. Simply type the value after the current value. If the value typed in is not a legal hexadecimal value or if more than two digits are typed, the illegal or extra character is not echoed.
2. Press the <SPACE> bar to advance to the next byte. To change the value, simply type 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. Type a hyphen (-) to return to the preceding byte. If you decide to change a byte behind the current position, typing the hyphen returns the current position to the previous byte. When the hyphen is typed, a new line is started with the address and its byte value displayed. NEWLINE
4. Press the <~~RETURN~~> key to terminate the Enter command. The <~~RETURN~~> key may be pressed at any byte position. NEWLINE

EXAMPLE Assume that the following command is typed:

```
ECS:100
```

DEBUG displays:

```
04BA:0100   EB.-
```

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

```
04BA:0100   EB.41-
```

To step through the subsequent bytes, press the <SPACE> bar to see:

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

To change BC to 42:

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

Now, realizing that 10 should be 6F, type the hyphen as many times as needed 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 the <~~RETURN~~ NEWLINE> key ends the Enter command and returns to the DEBUG command level.

NAME Fill

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

SYNTAX F<range> <list>

COMMENTS If the <range> contains more bytes than the number of values in the <list>, the <list> will be 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> will be ignored. If any of the memory in the <range> is not valid (bad or nonexistent), the error will occur in all succeeding locations.

EXAMPLE Assume that the following command is typed:

```
F04BA:100 L 100 42 45 52 54 41
```

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

NAME Go

PURPOSE Executes the program currently in memory.

SYNTAX

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

COMMENTS If only the Go command is typed, 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 typed 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.

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.

EXAMPLE

Assume that the following command is typed:

```
GCS:7550
```

The program currently in memory 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 type the Go command again, then the program executes just as if you had typed the filename at the MS-DOS command level. The only difference is that program execution begins at the instruction after the breakpoint rather than at the usual start address.

NAME	Hex
PURPOSE	Performs hexadecimal arithmetic on the two parameters specified.
SYNTAX	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 typed:

```
H19F 10A
```

DEBUG performs the calculations and then displays the result:

```
02A9   0095
```

NAME	Input
PURPOSE	Inputs and displays one byte from the port specified by <value>.
SYNTAX	I<value>
COMMENTS	A 16-bit port address is allowed.
EXAMPLE	Assume that you type the following command:

```
I2F8
```

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

```
42
```

NAME Load

PURPOSE Loads a file into memory.

SYNTAX L[<address> [<drive> <record> <record>]]

COMMENTS Set BX:CX to the number of bytes read. The file must have been named either when DEBUG was started or with the N command. Both the DEBUG invocation and the N command format a filename properly in the normal format of a file control block at CS:5C.

If the L command is typed without any parameters, DEBUG loads the file into memory beginning at address CS:100 and sets BX:CX to the number of bytes loaded. If the L command is typed with an address parameter, loading begins at the memory <address> specified. If L is typed with all parameters, absolute disk sectors are loaded, not a file. The <record>s are taken from the <drive> specified (the drive designation is numeric here-0=A:, 1=8:, 2=C:, etc.); DEBUG begins loading with the first <record> specified, and continues until the number of sectors specified in the second <record> have been loaded.

EXAMPLE Assume that the following commands are typed:

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

Now, to load FILE.COM, type:

```
L
```

DEBUG loads the file and then displays the DEBUG prompt. Assume that you want to load only portions of a file or certain records from a disk. To do this, type:

```
L04BA:100 2 0F 6D
```

DEBUG then loads 109 (6D hex) records beginning with logical record number 15 into memory beginning at address 04BA:0100. When the records have been loaded, DEBUG simply returns the - prompt.

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 will differ from its size in memory.

If the file named by the Name command or specified when DEBUG is started is a .HEX file, then typing 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.

NAME Move

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

SYNTAX M<range> <address>

COMMENTS Overlapping moves (i.e., 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 will not have new data written to them, the data there before the move will remain. 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.

EXAMPLE Assume that you type:

```
MCS:100 110 CS:500
```

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 type the D command, using the <address> typed for the M command, to review the results of the move.

NAME	Name
PURPOSE	Sets filenames.
SYNTAX	N<filename> [<filename> . . .]

COMMENTS The Name command performs two functions. First, Name is used to assign a filename for a later Load or Write command. Thus, if you start DEBUG without naming any file to be debugged, then the N<filename> command must be typed before a file can be loaded. Second, Name 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
-L
-G
```

Because of the effects of the Name command, Name will perform the following steps:

1. (N)ame assigns the filename FILE1.EXE to the filename 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 typed at the command level).

A more useful chain of commands might look like this:

```
-NFILE1.EXE
-L
-NFILE2.DAT FILE3.DAT
-G
```

Here, Name sets FILE1.EXE as the filename for the subsequent Load command. The Load command loads FILE1.EXE into memory, and then the Name 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 typed at the MS-DOS command level. Note that if a Write command were executed at this point, then FILE1.EXE - the file being debugged - would be saved with the name FILE2.DAT! To avoid such undesired results, you should always execute a Name command before either a Load or a Write.

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

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

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

EXAMPLE A typical use of the Name command is:

```
DEBUG PROG.COM
-NPARAM1 PARAM2/C
-G
-
```

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

```
PROG PARAM1 PARAM2/C
```

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

NAME	Output
PURPOSE	Sends the <byte> specified to the output port specified by <value>.
SYNTAX	0<value> <byte>
COMMENTS	A 16-bit port address is allowed.
EXAMPLE	Type:

```
02F8 4F
```

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

NAME Quit

PURPOSE Terminates the DEBUG utility.

SYNTAX 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, type:

NEWLINE

Q<~~RETU~~RN>

DEBUG has been terminated, and control returns to the MS-DOS command level.

NAME Register

PURPOSE Displays the contents of one or more CPU registers.

SYNTAX R[<register-name>]

COMMENTS If no <register-name> is typed, the R command dumps the register save area and displays the contents of all registers and flags.
If a register name is typed, the 16-byte value of that register is displayed in hexadecimal, and then a colon appears as a prompt. You then either type a <value> to change the register, or simply press the <~~RE-~~
NEWLINE ~~TU~~RN> key 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
DX	DS	PC	the 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, type 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

Whenever you type 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 the <~~RETURN~~> key. Flags for which new values were not entered remain unchanged.

NEWLINE

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 Type:

```
R
```

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 type:

```
RF
```

DEBUG will display the flags:

```
NV UP DI NG NZ AC PE NC - -
```

Now, type any valid flag designation, in any order, with or without spaces.

For example:

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

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

```
RF
NV UP EI PL NZ AC PE CY - -
```

Press <~~RETURN~~ NEWLINE> to leave the flags this way, or to specify different flag values.

NAME	Search
PURPOSE	Searches the <range< specified for the <list> of bytes specified.
SYNTAX	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 type:

```
SCS:100 110 41
```

DEBUG will display a response similar to this:

```
04BA:0104
04BA:010D
-type:
```

NAME	Trace
PURPOSE	Executes one instruction and displays the contents of all registers and flags, and the decoded instruction.
SYNTAX	T[=<address>] [<value>]
COMMENTS	If the optional =<address> is typed, 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	TYPE:

```
        T
```

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 CS=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 type

```
T=011A 10
```

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 <CONTROL-S> suspends the display at any point, so that you can study the registers and flags for any instruction.

NAME	Unassemble
PURPOSE	Disassembles bytes and displays the source statements that correspond to them, with addresses and byte values.
SYNTAX	U[<range>]
COMMENTS	The display of disassembled code looks like a listing for an assembled file. If you type the U command without parameters, 20 hexadecimal bytes are disassembled at the first address after that displayed by the previous Unassemble command. If you type the U command with the <range> parameter, then DEBUG disassembles all bytes in the range. If the <range> is given as an <address> only, then 20H bytes are disassembled instead of 80H.

EXAMPLE

Type:

```
U04BA:100 L10
```

DEBUG disassembles 16 bytes beginning at address 04BA:0100:

```
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 type

```
004ba:0100 0108
```

The display will show:

```
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 typed for the changed locations, the new instructions viewed, and the disassembled code used to edit the source file.

NAME	Write
PURPOSE	Wirtes the file being debugged to a disk file.
SYNTAX	W[<address> [<drive> <record> <records>]]
COMMENTS	If you type W with no parameters, BX:CX must already be set to the number of bytes to be written; the file is written beginning from CS:100. If the W command is typed with just an address, then the file is written beginning at that address. If a G or T command has been used, BX:CX must be reset before using the Write command without parameters. Note that if a file is loaded and modified, the name, length, and starting address are all set correctly to save the modified file (as long as the length has not changed). The file must have been named either with the DEBUG invocation command or with the N command (refer to the Name command earlier in this manual). Both the DEBUG invocation and the N command format a filename properly in the normal format of a file control block at CS:5C. If the W command is typed with parameters, the write begins from the memory address specified; the file is written to the <drive> specified (the drive designation is numeric here-0=A:, 1=B:, 2=C:, etc.); DEBUG writes the file beginning at the logical record number specified by the first <record>; DEBUG continues to write the file until the number of sectors specified in the second <record> have been written.

WARNING

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

EXAMPLE Type:

```
W
```

DEBUG will write the file to disk and then display the DEBUG prompt. Two examples are shown below.

```
W
--
```

```
WCS:100 1 37 2B
```

DEBUG writes out the contents of memory, beginning with the address CS:100 to the disk in drive B:. The data written out starts in disk logical record number 37H and consists of 2BH records. When the write is complete, DEBUG displays the prompt:

```
WCS:100 1 37 2B
--
```

2.3 ERROR MESSAGES

During the DEBUG session, you may receive any of the following error messages. Each error terminates the DEBUG command under which it occurred, but does not terminate DEBUG itself.

ERROR CODE	DEFINITION
BF	Bad flag You attempted to alter a flag, but the characters typed 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. Retype the Go command with ten or fewer breakpoints.
BR	Bad register You typed the R command with an invalid register name. See the Register command for the list of valid register names.
DF	Double flag You typed two values for one flag. You may specify a flag value only once per RF command.