User's Manual

Version 0.3

86-DOS™

Disk Operating System for the 8086

Preliminary

Seattle Computer Products, Inc.
1114 Industry Drive, Seattle, WA. 98188
(206) 575-1830

86-DOS™ User's Manual

TABLE OF CONTENTS

SCP 86-DOS™

A single-user disk operating system for the 8086

Introduction

86-DOS provides the tools needed to develop programs for the 8086, as well as a hardware-independent environment in which to run these programs. It is a very modular system. At its core is the disk file manager and I/O device handler, and everything else is considered a "user program". This allows the system to be easily tailored to any custom requirements.

The disk file manager allows programs to create and access disk files by name. Files may be read or written sequentially or randomly, any number of records at a time. File space on the disk is allocated dynamically, so that no "compaction" phase is ever required.

A program called COMMAND provides the interface between the user at the console and the file manager. COMMAND allows the user to display the disk directory, rename, destroy, or copy files, and execute other programs, such as the assembler, editor, or source code translator.

The assembler reads a source file of Intel-like 8086 mnemonics from disk, and produces a listing file and an Intel hex format object file. The program HEX2BIN may be used to convert the hex file to executable binary form.

The editor is line oriented, suitable for creating and maintaining program files. "Dynamic" line numbers, which are not actually present in the text, are used to identify lines to be listed, deleted, edited, etc. Global searching and global text replacement are also provided. After editing a file, the original file (before editing) is preserved as a back-up.

The source code translator can translate most Z80 source code into 8086 source code acceptable to the assembler after minor manual correction. This provides a relatively quick and easy way to transport programs between the processors.

The debugger allows programs to be tested under careful control. It also provides direct access to disk files by name or to any physical location on the disk.

SPECIAL NOTE: 86-DOS is not related to the popular CP/M operating system of Digital Research. Disk directory formatting and space allocation are completely different and incompatible. 86-DOS does, however, provide a utility called RDCPM which will transfer files from CP/M disks to 86-DOS disks. Further, operating system calls and calling conventions have been provided which make possible automatic translation of Z80 programs written for CP/M into 8086 programs that run under 86-DOS.

COMMAND — The Command Interpreter

TABLE OF CONTENTS

When 86-DOS is first initialized after the "system boot", a program called COMMAND is loaded and executed. COMMAND provides the interface between the user and the operating system by accepting lines of input from the console and converting them to system calls.

COMMAND prompts with a letter indicating the "default" disk drive, followed by a colon. Up to 15 drives are allowed in the system, labeled "A" through "O". The default drive is the one that will be used by any disk operation that does not specify a drive explicitly. Initially, the default drive is A, but the user may change this by entering a line consisting of a valid drive letter followed by a colon.

Control Functions

While typing in a command from the console, a number of control functions will be recognized (note that ALL standard 86-DOS programs recognize these functions when giving them commands):

Ctrl-S suspends everything until any key is typed.

Ctrl-P sends all console output to the printer also.

Ctrl-N stops sending output to the printer.

Ctrl-C causes an exit from the current function.

Rubout, delete, backspace, Ctrl-H (7F hex or 08 hex): Backspace. Removes the last character from the input buffer and erases it from the console.

Linefeed, Ctrl-J (10 hex): Physical end-of-line. Outputs a carriage return and linefeed but does not effect the input buffer.

Ctrl-X (18 hex): Cancel line. Outputs a back slash, carriage return, and linefeed and resets the input buffer to empty. The template used by the special editing commands is not affected.

Special Editing Commands

These special editing commands involve a "template", which is an input line available to help construct the line currently being entered. There are two ways to obtain a template.

All standard 86-DOS programs automatically provide the last command entered as the template for the next command. This allows the command to be repeated easily, or an error in it to be corrected before the command is retried.

In addition, one of the editing commands is to convert that part of the line entered so far into the template, and restart the line entry. This allows an error near the start of a line to be corrected without retyping the rest of the line.

Each editing command is selected by typing ESCAPE and a letter. Since many terminals provide keys which produce such an "escape code" with a single keystroke, the letter used after the ESCAPE may be set for each command during 86-DOS customization. The standard escape sequences correspond to the special function keys of a VT-52 or similar terminal, as noted in each case by parentheses.

ESC S (F1) - Copy one character from the template to the new line.

ESC T (F2) - Must be followed by any character. Copies all characters from the template to the new line, up to but not including the next occurrence in the template of the specified character. If the specified

character does not occur, nothing is copied to the new line.

ESC U (F3) - Copy all remaining characters in the template to the new line.

ESC V (F4) - Skip over one character in the template.

ESC W (F5) - Must be followed by any character. Skips over all characters in the template, up to but not including the next occurrence in the template of the specified character. If the specified character does not occur, no characters are skipped.

ESC P (BLUE) - Enter insert mode. As additional characters are typed, the current position in the template will not advance.

ESC Q (RED) - Exit insert mode. The position in the template is advanced for each character typed. When editing begins, this mode is selected by default.

ESC R (GRAY) - Make the new line the template. Prints an "@", a carriage return, and a line feed. Buffer is set to empty and insert mode is turned off.

The general form of an input line for COMMAND is a command name followed by zero or more parameters. These parameters may be separated from the command name and each other by blanks, tabs, commas, semicolons, or "equals" signs. The number and meaning of the parameters is entirely dependent on the individual command.

File Names

Many commands accept a file name as a parameter. File names may have up to three parts: 1) an optional disk drive specifier; 2) the name; 3) an optional extension. The drive specifier, if present, consists of a valid drive letter followed by a colon. The name consists of 1 to 8 characters. The extension, if present, is a period followed by 1 to 3 characters. If the name or extension include a "?" in any position, then that position will match any character when a directory search is made for the file name. If a "*" appears in the name or extension, then the rest of the name or extension is filled in with "?". For example, the following pairs of names are equivalent:

.	????????.???
*.COM	????????.COM
PROG.*	PROG.???
FILE*.A*	FILE????.A??

Internal Commands

The commands themselves are of two types. The "internal commands" are built in to COMMAND and are executed immediately upon recognition. The "external commands", on the other hand, are kept on disk as program files and must first be loaded into memory before they can be executed. Any file on the disk with an extension of "COM" is considered to be a valid external command.

The internal commands are the following:

DIR	List directory entries
RENAME	Rename files
ERASE	Delete files
COPY	Copy a file
TYPE	Display the contents of a file
CLEAR	Format a disk directory, erasing all files

Each is described below, with all valid forms of their parameters shown. In this list, "drive" means a valid disk specifier (a drive letter followed by a colon), while "file" means a file name as described above.

```
DIR
DIR     drive
DIR     file
```

If no parameters are present, all directory entries on the default disk are listed at the console. If a drive specifier is present, then all entries on specified disk are listed. If a file name is given, then only matching files are listed. Files are listed with their size in bytes.

```
RENAME  file1   file2
```

Every matching occurence of the first file name is changed to the second name. A disk drive specifier on the second file name is ignored. If "?" or "*" characters appear in the second file name, then corresponding positions in the original name will be unchanged. Care should be taken not to give two files the same name.

```
------------
ERASE   file
------------
```

All files matching the given file name are deleted from the disk directory.

```
---------------------
COPY    file1
COPY    file1   drive
COPY    file1   file2
---------------------
```

Each file matching file1 is copied to another file. If a second file name is not given, then a copy with the same name is made on the specified (or default) drive. If a second file name is given, then the copy will be given this name. If the second file name has "?" or "*" characters in it, then corresponding positons in will be the same as the original file name. If the destination file already exists, it will be overwritten; otherwise, a new file will be created. Do not attempt to copy to a file of the same name on the same disk--the file will be destroyed.

```
------------
TYPE    file
------------
```

The specified file is transfered from the disk to the console. Tab characters are expanded with blanks to every eighth column, but otherwise no formatting is performed.

```
-------------
CLEAR   drive
-------------
```

Before any disk medium may be used with 86-DOS, its directory must be initialized to empty. If it is not, a "Bad FAT" message will probably result. To prevent catastrophe, the user will be prompted before formatting takes place. A "Y" response is required to continue.

External Commands

```
---------------------
RDCPM   file1
RDCPM   file1   drive
RDCPM   file1   file2
---------------------
```

Very similar to COPY, except that the source, file1, is assumed to be on a disk formatted by a CP/M-compatible system. The destination file must be on a different drive, assumed to be formatted by 86-DOS.

MAKRDCPM file

This program is used to modify RDCPM to handle the varying formats possible with CP/M 2. The specified file must contain drive parameter tables and a copy of RDCPM must be present on the same drive. Any existing parameter tables in RDCPM will be deleted and the new tables will be merged into RDCPM.

The file with the parameter tables has three parts. The first word (2 bytes) is the length of the tables. Next come 16 words which are the addresses of the parameter table for each drive. Any drive not in the system or not defined for CP/M use should have a word of zero. Last come the tables themselves, which are of two types: drive parameter tables and sector translation tables. Any drives with the same parameters may share the same tables.

The drive parameter tables are very similar to those used within CP/M 2. In fact, if a listing of the CP/M 2 Disk Parameter Blocks used is available, these may be used directly, with the addition of one parameter at the end. The added parameter is the address of the sector translation table, which in CP/M 2 is found in the Disk Parameter Header. (There is no equivalent to the Disk Parameter Header in the RDCPM tables.) The sector translation tables also perform the same function as in CP/M 2, except that RDCPM performs the translation rather than a user-written SECTRANS routine. The sector translation table must contain one-byte entries if there are less than 256 sectors per track, and two-byte entries if there are 256 or more sectors per track. Also, the sector translation table must have entries starting with zero, even if the sectors on the disk are numbered starting with one. Thus, to make a translation table for IBM disks, 1 must be subtracted from each entry in a CP/M translation table.

HEX2BIN file

The specified file is assumed to be in Intel hex format, as produced by the Assembler. If no extension is given, "HEX" is assumed. A file of the same name but with an extension of "COM" is produced which is the absolute binary equivalent of the hex input file, offset downward by 100 hex.

CHKDSK
CHKDSK drive

The directory of the specified (or default) drive is scanned and completely checked for consistency. If any errors are found, they are reported and corrective action is attempted. The following messages are possible:

"Error in allocation table for file <name>" - the file had a data block allocated to it that did not exist. To correct, the file is truncated short of the bad block.

"Non-recoverable directory error--file deleted: <name>" - Similar to above, except that no valid data blocks remained allocated to the file, so there is no point in keeping it in the directory.

"Files cross-linked: <name> <name>" - The same data block has been allocated to both files. One of the files could still be O.K., but the only way to tell is to view the data. No corrective action is taken automatically. The recommended procedure is to make copies of both files, then delete the originals. The copies must be checked to see if either are valid. Deleting one of the originals will permanently damage the other, so both must be copied before either are deleted.

"Directory error--incorrect size of file <name>" - The size of the file as kept in the directory did not correspond to the actual amount of space allocated. The size is readjusted to equal all of the allocated space. Note that the size is kept in bytes, while the allocation unit is much larger (several hundred bytes, typically). Thus a file may be shorter than the new adjusted size, since all of the last data block may not be used.

"xxxxxx bytes unallocated disk space freed" - Disk space was marked as allocated, yet a check showed it belonged to no file. The space is freed.

After any error messages, a status report appears, listing the number of disk files, the size of the disk, the amount of free space left on the disk, the total size of central memory, and the amount of memory available to a running program. It is recommended that CHKDSK be run on each diskette occasionally to verify the integrity of its directory structure.

SYS drive

Transfers a copy of the 86-DOS system from Drive A to the specified drive. The system is assumed to occupy 52 128-byte records at the beginning of the disk, which corresponds to all of tracks 0 and 1 on single-density IBM format diskettes. This size is kept in the fourth byte of SYS, and may be changed (with DEBUG) to any value desired.

Note that the "system" refers only to the core of 86-DOS, and not to any of the files also on the disk. At least the program COMMAND.COM must also be copied in order to boot from this disk.

EDLIN — The Line Editor

TABLE OF CONTENTS

The text editor is called with the command EDLIN <file>. If the file already exists, EDLIN will load it into memory. Otherwise, a new file will be created and the message "New file" will be displayed.

The text is divided into lines, each of which ends with an end-of-line mark and has less than 255 characters. Each line has a number, which represents the position of that line in the file. If lines are inserted or deleted at any point, then the numbers of all lines after that point will increase or decrease so that consecutive numbering is maintained. The line numbers are not actually present in the text.

Parameters

Each of the seven commands accepts some optional parameters. The standard forms for these parameters is given below, although their meaning is dependent upon the individual command.

<line>
This parameter references a numbered line of text. It appears on the command line BEFORE the command itself, and may be separated from other parameters and the command by commas or spaces. A <line> may be specified in any of the following ways:

1. By a line number (an integer less than 65534). If the number is greater than the largest existing line number, then the parameter will reference the first point after all existing text.

2. By a period ("."), which means the "current line". The current line is marked by an asterisk ("*") when lines are listed, and is generally set to the last line edited, as noted in the command descriptions below.

3. By a "#", which means "after the last line" (This is the same as specifying a line number greater than any existing one.)

If nothing is entered when a <line> parameter is called for, each command has a standard default value as specified in the command descriptions below.

<?>
This parameter may appear on the line before the Search or Replace commands to cause Edlin to ask the user if the correct text string has been found. This parameter is specified by the presence of the "?" character.

<string>
This parameter may be any text terminated by a Ctrl-Z or <return>. The termination character is not considered part of the text. Ctrl-L may be used within the text to refer to the end-of-line mark. <string> is used only in the Search and Replace commands, appearing immediately after the letter specifying the command or immediately after the previous <string>.

Commands

Each editing command is specified by a single letter, except for the Edit Line command, which is specified by the absence of any other command letter. The command letter may optionally be preceded by <line> and <?> parameters (whose meaning is dependent upon the command). For the Search and Replace commands, the command letter is immediately followed by one or two <string>s. For the other commands, everything after the command letter is ignored.

Control Functions

While entering the command line, a number of control functions are recognized:

Ctrl-S suspends everything until any key is typed.

Ctrl-P sends all console output to the printer also.

Ctrl-N stops sending output to the printer.

Ctrl-C causes an exit from the current function.

Rubout, delete, backspace, Ctrl-H (7F hex or 08 hex): Backspace. Removes the last character from the input buffer and erases it from the console.

Linefeed, Ctrl-J (10 hex): Physical end-of-line. Outputs a carriage return and linefeed but does not effect the input buffer.

Ctrl-X (18 hex): Cancel line. Outputs a back slash, carriage return, and linefeed and resets the input buffer to empty. The template used by the special editing commands is not affected.

Special Editing Commands

These special editing commands involve a "template", which is an input line available to help construct the line currently being entered. There are two ways to obtain a template.

All standard 86-DOS programs automatically provide the last command entered as the template for the next command. This allows the command to be repeated easily, or an error in it to be corrected before the command is retried.

In addition, one of the editing commands is to convert that part of the line entered so far into the template, and restart the line entry. This allows an error near the start of a line to be corrected without retyping the rest of the line.

Each editing command is selected by typing ESCAPE and a letter. Since many terminals provide keys which produce such an "escape code" with a single keystroke, the letter used after the ESCAPE may be set for each command during 86-DOS customization. The standard escape sequences correspond to the special function keys of a VT-52 or similar terminal, as noted in each case by parentheses.

ESC S (F1) - Copy one character from the template to the new line.

ESC T (F2) - Must be followed by any character. Copies all characters from the template to the new line, up to but not including the next occurrence in the template of the specified character. If the specified character does not occur, nothing is copied to the new line.

ESC U (F3) - Copy all remaining characters in the template to the new line.

ESC V (F4) - Skip over one character in the template.

ESC W (F5) - Must be followed by any character. Skips over all characters in the template, up to but not including the next occurrence in the template of the specified character. If the specified character does not occur, no characters are skipped.

ESC P (BLUE) - Enter insert mode. As additional characters are typed, the current position in the template will not advance.

ESC Q (RED) - Exit insert mode. The position in the template is advanced for each character typed. When editing begins, this mode is selected by default.

ESC R (GRAY) - Make the new line the template. Prints an "@", a carriage return, and a line feed. Buffer is set to empty and insert mode is turned off.

All commands may terminated by typing Ctrl-C at any time. If the Edit Line command is aborted in this manner, no changes will be made to the line. If the Insert Text command is aborted, the line currently being entered will not be saved but any previously entered lines will.

Edit Line: <line>

If no line number is specified (i.e., the command line is blank), the line after the current line is edited.

First the line to be edited will be displayed with its line number, then below it its line number will appear as a prompt for corrections. If no changes are desired, simply type <return>. Otherwise, a new line may be entered with the old one as a "template". All editing operations listed above for commands apply.

Delete Lines: <line>,<line> D

Deletes the specified lines, and everything between. If the first <line> is omitted, it is assumed to be the current line. If the second <line> is omitted, it is assumed to be the same as the first (i.e., one line is deleted). The line immediately after the deleted text will become the current line (it will now have the same line number as the first line deleted).

List Text: <line>,<line> L

Lists the specified range of lines. If the first <line> is omitted, the current line is assumed. If the second <line> is omitted, then 23 lines are listed, from 11 above the first parameter to 11 below it. The current line is not changed, but if the current line is listed, it is marked with an asterisk ("*") after the line number.

Insert Text: <line> I

New lines of text will be entered immediately before the specified line. After entering this command, each new line will be prompted by a line number. To exit this insert mode, type a Ctrl-Z as the first character of a line, or simply type Ctrl-C. The line immediately following the new text will become the current line. If <line> is not specified, the default is the current line number. If <line> is greater than the total number of lines of text (as with "#"), then the lines are appended to the end.

Search Text: <line>,<line> <?> S<string>

If the first <line> is omitted, 1 is assumed. If the second <line> is omitted, then "#" is assumed. If the <string> is omitted (or has zero length), the command is terminated immediately.

The specified range of lines is searched for the specified string. If the string is not found within the range, a message to that effect is displayed; otherwise, the line with the match is displayed. If the <?> parameter does not appear, the line of the match becomes the current line and the command terminates. Otherwise, the user is asked if this is the correct occurence. The one-letter response may be a "Y" or <return> for yes, or anything else for no, in which case the search continues.

Replace Text: <line>,<line> <?> R<string><string>

If the first <line> is omitted, 1 is assumed. If the second <line> is omitted, then "#" is assumed. If the first <string> is omitted (or has zero length), the command is terminated immediately. If the second <string> is omitted, then it is considered to be "null", which will effectively delete proper occurences of the first <string>.

If the <?> is not specified, then all occurences of the first <string> within the range are replaced by the second, and for each replacement, the line is displayed.

If the <?> is specified, then the specified range is searched for the first <string>. If found, the line is displayed with the matching <string> replaced by the second <string>. The user is then asked if this is correct. A "Y" or <return> may be entered if so, or any other key if not, and the change is or is not made permanent as appropriate. In either case, the search then continues for the next occurence, and the process is repeated.

End Editing: E

All text is saved on the disk and EDLIN terminates. The original file, before editing, is retained on the disk but its extension is changed to "BAK".

Error Messages

Errors While Starting Up

"Cannot edit .BAK file--rename file" - Files with an extension of BAK cannot be edited because that extension is reserved for backup copies. The file must be renamed with any other extension.

"No room in directory for file" - This means the system returned an error when an attempt to create the file was made. Besides a full directory, this could also mean an illegal disk drive or file name was used, since these will also return an error when a create file request is made.

"File too big to fit into memory" - Currently, the entire file to be edited must fit into available memory. EDLIN will not use memory outside the 64K segment in which it resides, so the maximum file size is 64K minus size of EDLIN, regardless of how much memory is actually present.

"No end-of-file mark found in file" - After loading the file, EDLIN scans from the end for an end-of-file mark (1A hex), and truncates the file after that point. If no end-of-file mark is found, there is nothing to edit.

Errors While Editing

"Entry error" - There is a syntax error in the last command entered.

"Memory full" - Causes Insert Text mode to abort because there is insufficient space left in memory for the last line entered.

"Line too long" - Causes the Replace command to abort because performing the requested replacement would have made a line longer than 254 characters, which is not allowed.

ASM — The Resident Assembler

This resident assembler allows programs written in 8086 mnemonics to be assembled for running on the 8086 processor. The input file of Intel-like 8086 mnemonics is read from disk; and optional object file is written to disk in Intel hex format, and the print listing may be sent to the disk, to the console, or it may be suppressed. Mnemonics differ from Intel's only to provide sufficient information to assemble a source line independent of context. Pseudo-ops provided are ALIGN, DB, DM, DS, DW, EQU, IF/END IF, ORG and PUT. The assembler automatically takes full advantage of the 8086's addressing modes, using special short forms wherever possible.

TABLE OF CONTENTS

Calling the Assembler

The assembler is invoked with the command ASM FILENAME , which will assemble the 8086 source file named FILENAME.ASM. The extension "ASM" is always assumed and may not be overridden. This is the simplest form of the command. It assumes FILENAME.ASM resides on the current drive, and will write the Intel hex object file, named FILENAME.HEX, and the assembler listing, FILENAME.PRN, to the current drive.

The first variation of this form is to precede the file name with a drive specifier and a colon, such as ASM B:FILENAME , which will cause the specified drive to be searched for the source file, but the object and listing files will still be written to the current drive.

The most general form is ASM FILENAME.<DRIVE ASSIGNMENT>. The <DRIVE ASSIGNMENT> is a 3-letter extension not related to the actual extension to the source file, which is always ASM. Instead, it is used as follows:

1. The first letter is the name of the drive on which the source file will be found. This overrides a disk specifier which precedes the file name ("B:").

2. The second letter is the name of the drive to which the hex object file will be written, or "Z" if no object file is desired.

3. The third letter is the name of the drive to which the listing file will be written, or "X" to send the listing to the console, or "Z" if no listing file is desired. Assembling with no listing is much faster since the source file will not be read from disk a second time.

If a listing is selected, then an alphabetical symbol table dump may be appended to it by typing "S" after the file name and extension.

Examples:

```
ASM FILENAME.ABA
        Source - Drive A
        Object - Drive B
        Listing - Drive A

ASM FILENAME.AAZ
        Source - Drive A
        Object - Drive A
        No Listing

ASM FILENAME.BZX S
        Source - Drive B
        Object - None
        Listing (with symbol table dump) - Console
```

Several errors will cause the assembler to print an error message and abort:

FILE NOT FOUND - The source file was not found on the specified disk. Probably a misspelling or wrong disk.

BAD DISK SPECIFIER - The file name's extension contained an illegal character. Only "A"-"W" and possibly "X" or "Z" are legal.

NO DIRECTORY SPACE - The object or listing file could not be created.

DISK WRITE ERROR - Probably insufficient space on disk for object or listing files.

INSUFFICIENT MEMORY - Memory requirements increase with source program size due to storage required by the symbol table and by the intermediate code. Requirements can be reduced by using shorter labels, by defining labels before they are used, and by reducing the total number of program lines.

Source Code Format

Input to the assembler is a sequence of lines, where each line is terminated with ASCII carriage return and linefeed characters. The assembler accepts lines of any length, but does no list formatting so line length may be limited by your list device. Upper and lower case characters are completely equivalent and may be mixed freely.

Each line may include up to four fields, which may be separated from each other by any number of spaces or tabs (control-I). Fields must appear in order, as follows:

1. Label field (optional) - If present, it must either begin with the first character on the line or be followed immediately by a colon. A label begins with a letter and may be followed by any number of letters or digits, up to a total length of 80 characters, all of which are significant.

2. Opcode field (optional) - If present, it must begin AFTER the first character on the line (otherwise it would be mistaken for a label).

3. Operand field - This field is present only as required by the opcode field.

4. Comment field (optional) - If present, it must begin with a semicolon (;).

Since all fields are optional, lines may be blank, may have labels only, may have comments only, etc.

Bus lock (LOCK), string repeat (REP), and segment override (SEG) prefixes are treated as separate opcodes and must appear on the line preceding the opcode they are to prefix.

Operands

Each operand is one of the following types:
1. A Register
2. A Value
3. An Address

---

1. REG - A register:

AX, BX, CX, DX, AL, AH, BL, BH, CL, CH, DL, DH, DI, SI, DI, SP, BP, CS, DS, ES, SS. Most instructions have limitations on which registers may be used.

---

2. VALUE

An expression involving additon or subtraction of constants or labels. Terms of the expression may be:

a) A decimal constant ("486").

b) A hex constant, which must begin with a digit from 0 to 9, and end with an "H" ("0F9H").

c) A string constant. In general, this is any number of characters enclosed by either single (') or double (") quotes. Since the opening and closing quotes must be the same, the other type may appear in the string freely. If the same quote as opened the string needs to appear within it, it must be given as two adjacent quotes. Examples:

"TEST" is the same as 'TEST'

"'" is the same as ''''

Control characters except control-Z (1AH) may appear in the string, but this may have a strange effect on the listing.

Note that multi-character strings are meaningful only for the DB, DM, and DW pseudo-ops. All other expressions are limited to one character strings.

d) A label. No more than one undefined label may appear in an expression, and undefined labels may only be added, not subtracted. An undefined label is one which has not yet appeared in the label field as the source code is scanned from the beginning to the current line.

e) "$". This special symbol means the value of the location counter at the start of this instruction.

f) "RET". This special symbol means "the address of a nearby RETurn instruction". The purpose of this is to allow conditional returns without requiring a label to be put on the RETurn instruction. For example,

```
CMP     AL,20H
JC      RET
```

The jump instruction effectively means "return if carry", yet no label named RET need appear--in fact, a label would be ignored. If no RETurn instruction appears within the range of the jump, then a "value error", number 65 hex, will occur. Only RETurn instructions with no operands will be the target of the jump (intra-segment return without adding to stack pointer).

Generally, a VALUE consists of:

a) An optional leading + or -.

b) A term.

c) Zero or more additional terms, each preceded by a + or -.
An exception to this is that no terms may precede a multi-character string in an expression. Terms may follow the string, in which case whey will be added to or subtracted from the value of the last character.
Examples:

Legal: "Time" + 80H

Illegal: -"Time" or 80H + "Time"

---

3. [ADDR]

A valid 8086 address expression enclosed within brackets. The address expression may be:

a) A VALUE, as defined above.

b) A base register (BP or BX).

c) An index register (SI or DI).

d) The sum of any of the above, as limited by valid 8086 addressing modes.

Examples of Operands

Legal:

```
        -3+ 17H

        SCOPE+4

        [ bx + COUNT-2 ]

        [SI+ARRAY+BX-OFFSET]     ;OFFSET must have already been defined

        [DI]

        [ NEXT]
```

Illegal:

```
        12+BX                    ;Register not allowed in VALUE

        9C01                     ;Needs trailing "H"

        [Count - BX]             ;Can't subtract register

        [BX+BP]                  ;Only one base register at a time

        [ARRAY+BX+OFFSET]        ;Both labels are forward referenced (Note 1)

        COUNT-DIF                ;DIF is forward referenced (Note 2)
```

Note 1. This problem could be corrected like this:

```
      MOV      AX,[BX + ARRAYPLUSOFFSET]
          .
          .
          .
ARRAY:
          .
OFFSET:
          .
          .
          .
ARRAYPLUSOFFSET:  EQU   ARRAY + OFFSET
```

Note 2. This problem could be corrected like this:

```
          MOV      AX,COUNT + MINUSDIF
          .
          .
          .
DIF:
          .
          .
          .
MINUSDIF:EQU     -DIF
```

Pseudo-Ops

ALIGN

ALIGN assures that the next location counter address is even, i.e., aligned on a word boundary. If the location counter is currently odd, both it and the PUT address are incremented; otherwise they are unchanged. See PUT and ORG for an explanation of these terms.

```
DB      VALUE
DB      VALUE, VALUE, VALUE, . . ., VALUE
```

DB (Define Byte) is used to tell the assembler to reserve one or more bytes as data in the object code. Each value listed is placed in sequence in object code, where a multi-character string is equivalent to a sequence of one-character strings. Values must be in the range -256 to +255. Example:

```
        DB      "Message in quotes",0DH,0AH,-1
```

```
DM      VALUE
DM      VALUE, VALUE, VALUE, . . ., VALUE
```

DM (Define Message) is nearly identical to DB, except that the most significant bit (bit 7) of the last byte is set to one. This can be a convenient way to terminate an ASCII message since this bit would not otherwise be significant. Example:

```
        DM      'Message in quotes',0DH,0AH
```
is equivalent to
```
        DB      'Message in quotes',0DH,0AH+80H
```

```
DS      VALUE
```

DS (Define Storage) is used to tell the assembler to reserve VALUE bytes of the object code as storage. Any labels appearing in the expression for VALUE must have already been defined.

```
----------------------------------------
DW      VALUE
DW      VALUE, VALUE, VALUE, . . ., VALUE
----------------------------------------
```

DW (Define Word) is used to tell the assembler to reserve one or more 16-bit words as data in the object code. It is very similar to DB, except that each value occupies two bytes instead of one. Since a multi-character string is equivalent to a sequence of one-character strings,

```
        DW      'TEST'
```

is equivalent to

```
        DB      'T',0,'E',0,'S',0,'T',0
```

because the high byte of the 16-bit constant represented by 'T' is always zero.

```
---------------------
LABEL:  EQU     VALUE
---------------------
```

EQU (Equate) assigns the VALUE to the label. The label MUST be on the same line as the EQU. Three common uses of this operation are:

1. To assign a name to a constant, for convenience and documentation. For example:

```
CR:     EQU     13
LF:     EQU     10
```

The program could now refer to ASCII carriage return and linefeed with symbols CR and LF, respectively.

2. To "parameterize" a program. I/O ports and status bits, for example, could be set by equates at the beginning of the program. Then to reassemble the program for a different I/O system would require editing only these few lines at the beginning.

3. To bypass expressions that would have two or more undefined labels or that would subtract an undefined label. See examples under OPERANDS.

```
-------------
IF      VALUE
ENDIF
-------------
```

IF allows portions of the source code to be assembled only under certain conditions. Specifically, that portion of the source code between the IF and ENDIF will be assembled only if the operand is NOT zero. This is particularly useful when producing different versions of the same program. IFs may not occur within an IF/ENDIF pair.

ORG VALUE

ORG sets the assembler's location counter, which is subsequently incremented for each byte of code produced or space allocated. The value of the location counter should always be equal to the displacement from the beginning of the segment to the next byte of code or data, since it is used to establish the value of labels. ORG may be used any number of times in a program. Any labels appearing in the expression for VALUE must have already been defined.

PUT VALUE

The assembler writes object code to the disk in Intel hex format. This format includes information which specifies the addresses at which the object file will be later loaded into memory by a hex loader.

PUT is used to specify this load address. Intially, the load address is 100H, that is, "PUT 100H" is assumed before assembly begins. Each time a PUT occurs, all subsequent code would be loaded starting at the specified address until the next PUT is encountered. This allows modules to be placed in specific areas of memory. Note that the load address is not related to the location counter (see ORG), although PUTs and ORGs will often occur together. Any labels appearing in the expression for VALUE must have already been defined.

Opcode Classifications

TWO OPERAND ALU

ADC, ADD, AND, CMP, DIV, IDIV, IMUL, MOV, MUL, OR, SBB, SBC, SUB, TEST, XCHG, XOR

Operand Forms:

REG,REG	Register to register
[ADDR],REG	Register to memory
REG,[ADDR]	Memory to register
REG,VALUE	Immediate to register
B,[ADDR],VALUE	Byte immediate to memory
W,[ADDR],VALUE	Word immediate to memory
[ADDR],VALUE	Immediate to memory defaults to word

Specific Notes:

SBC is the same as SBB.

The order of operands for TEST and XCHG is irrelevant.

XCHG may not use immediate operands.

For DIV, IDIV, MUL, and IMUL, the first operand must be AL or AX and the second operand may not be immediate.

ONE OPERAND ALU

DEC, ESC, INC, NEG, NOT, POP, PUSH

Operand Forms:

REG	Register
B,[ADDR]	Memory byte
W,[ADDR]	Memory word
[ADDR]	Default to word

Specific Notes:

POP, PUSH, and ESC only operate on words.

INPUT/OUTPUT

IN, INB, INW, OUT, OUTB, OUTW

Operand Forms:

VALUE	Input/output to fixed port
DX	Input/output to port number in DX

Specific Notes:

IN, INB, OUT, OUTB transfer bytes.

INW, OUTW transfer words.

OPCODE CLASSIFICATIONS (Continued)

SHIFT/ROTATE

RCL, RCR, ROL, ROR, SAL, SAR, SHL, SHR

Operand Forms:

REG	Shift/rotate register one bit
REG,CL	Shift/rotate register CL bits
B,[ADDR]	Shift/rotate memory byte
B,[ADDR],CL	
W,[ADDR]	Shift/rotate memory word
W,[ADDR],CL	
[ADDR]	Default to word
[ADDR],CL	

Specific Notes:

SHL and SAL are the same.

SHORT JUMPS

JA, JAE, JB, JBE, JC, JCXZ, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ, LOOP, LOOPE, LOOPNE, LOOPNZ, LOOPZ

Operand Form:

VALUE Direct jump

Specific Notes:

VALUE must be within -126 to +129 of instruction pointer, inclusive.

JP is NOT Jump on Parity. JP is the unconditional short direct jump.

JC, JNC are Jump on Carry and Jump on Not Carry, respectively.

LONG JUMPS/CALLS

CALL, JMP

Operand Forms:

VALUE	Intra-segment direct
VALUE,VALUE	Inter-segment direct
REG	Intra-segment indirect through register
[ADDR]	Intra-segment indirect through memory
L,[ADDR]	Inter-segment indirect through memory ("Long")

Specific Notes:

JMP does NOT include the short direct jump. Its mnemonic is JP and is included under "Short Jumps".

RETURN

```
        RET
```

Operand Forms:

(none)	Intra-segment
L	Inter-segment ("Long")
VALUE	Intra-segment and add VALUE to SP
L,VALUE	Inter-segment and add VALUE to SP

STRING OPERATIONS

CMPB, CMPW, LODB, LODW, MOVB, MOVW, SCAB, SCAW, STOB, STOW

No operand. These mnemonics replace Intel's CMPS, LODS, SCAS, STOS. The ending "B" or "W" distinguishes between byte and word operations, respectively.

INTERRUPT

INT

Operand Form:

VALUE

ADDRESS MANIPULATION

LDS, LEA, LES

Operand Form:

REG,[ADDR] Put effective address in register

SEGMENT OVERRIDE PREFIX

SEG

Operand Form:

REG	Must be a segment register (CS, DS, ES, SS)

Specific Notes:

This opcode should appear on the line immediately preceding the line to be prefixed.

STRING REPEAT PREFIXES

REP, REPE, REPNE, REPNZ, REPZ

No operand. Conditional repeats should be read as "Repeat while . . .", e.g., REPE is Repeat While Equal. For those string operations which affect the flags, REP, REPE, REPZ, all repeat while the zero flag is set; REPNZ, REPNE repeat while the zero flag is clear. This opcode should appear on the line immediately preceding the string operation to be prefixed.

ALL OTHER OPCODES

AAA, AAD, AAM, AAS, CBW, CLC, CLD, CLI, CMC, CWD, DAA, DAS, DI, DOWN, EI, HALT, HLT, INTO, IRET, LAHF, LOCK, NOP, POPF, PUSHF, SAHF, STC, STD, STI, UP, WAIT, XLAT

No operand.

Specific Notes:

DI is the same as CLI.

EI is the same as STI.

UP is the same as CLD.

DOWN is the same as STD.

HALT is the same as HLT.

NOP is the same as XCHG AX,AX.

LOCK is treated as a separate opcode and should appear on the line immediately preceding the opcode it is to prefix.

Error List

When a non-fatal error occurs in the source code, the next line of the listing will have an error message which will include a error number in hex. The following table lists the cause associated with the given error number.

01 Register field not allowed
02 Only BP, BX, SI, DI allowed
03 Only one base register (BP, BX) allowed
04 Only one index register (SI, DI) allowed
05 Subtraction of register or undefined label not allowed
06 Only one undefined label per expression allowed
07 Illegal digit in hex number
08 Illegal digit in decimal number
0A Illegal character in label or opcode
0B Double defined label
0C Opcode not recognized
14 Invalid operand
15 "," expected
16 Register mismatch
17 Immediate not allowed here
18 "]" expected
19 Memory-to-memory not allowed
1A Immediate may not be destination
1B Register-to-register not allowed here
1C Must specify segment register
1D Load only
1E Constant must be defined
1F Value error
20 Flag must be set only once
21 Label never defined
22 "EQU" must have label on same line
23 Zero length string illegal
24 ENDIF without IF
25 One-character strings only
26 Expression may not precede multi-character string
64 Undefined label
65 Value error

Index to Opcodes

This list includes all opcodes recognized by the assembler plus those used by Intel but not used by Seattle Computer Products (SCP). Each has the page number on which it will be found in this manual, where a "+" denotes an Intel opcode NOT recognized by the cross assembler. Also listed is the page number on which a description of the operation will be found in the Intel 8086 Family User's Manual, where the "*" means Intel uses a different mnemonic for that operation. Opcodes with no entry under GROUP will be found under "All Other Opcodes".

OPCODE	GROUP	MANUAL PAGE	INTEL PAGE	REMARKS
AAA		42	2-35	
AAD		42	2-37	
AAM		42	2-37	
AAS		42	2-36	
ADC	Two Operand ALU	30	2-35	
ADD	Two Operand ALU	30	2-35	
AND	Two Operand ALU	30	2-38	
CALL	Long Jumps/Calls	35	2-43	
CBW		42	2-38	
CLC		42	2-47	
CLD		42	2-47	
CLI		42	2-48	
CMC		42	2-47	
CMP	Two Operand ALU	30	2-36	
CMPB	String Operations	37	2-42*	Intel uses CMPS
CMPS	String Operations	37+	2-42	Use CMPB, CMPW
CMPW	String Operations	37	2-42*	Intel uses CMPS
CWD		42	2-38	
DAA		42	2-36	
DAS		42	2-36	
DEC	One Operand ALU	31	2-36	
DI		42	2-48*	Intel uses CLI
DIV	Two Operand ALU	30	2-37	
DOWN		42	2-47*	Intel uses STD
EI		42	2-48*	Intel uses STI
ESC	One Operand ALU	31	2-48	
HALT		42	2-48*	Intel uses HLT
HLT		42	2-48	
IDIV	Two Operand ALU	30	2-37	
IMUL	Two Operand ALU	30	2-37	
IN	Input/Output	32	2-32	SCP/Intel different
INB	Input/Output	32	2-32	Intel uses IN
INC	One Operand ALU	31	2-35	
INT	Interrupt	38	2-46	
INTO		42	2-47	
INW	Input/Output	32	2-32*	Intel uses IN
IRET		42	2-47	
JA	Short Jumps	34	2-45	
JAE	Short Jumps	34	2-45	
JB	Short Jumps	34	2-45	

TRANS — The Z80 to 8086 Translator

TABLE OF CONTENTS

The Seattle Computer Products Z80 to 8086 Translator accepts as input a Z80 source file written using Zilog/Mostek mnemonics and converts it to an 8086 source file in a format acceptable to our 8086 Assembler.

To translate a file, simply type TRANS <filename>.<ext> . Regardless of the original extension, the ouput file will be named <filename>.ASM and will appear on the same drive as the input file.

The entire Z80 assembly language is not translated. The following opcodes will result in an "opcode error":

CPD
CPI
IM
IND
INDR
INI
INIR
LDD
LDI
OTDR
OTIR
OUTD
OUTI
RLD
RRD

Only the following pseudo-ops are allowed:

DB
DM
DS
DW
EQU
IF/ENDIF
ORG

Any others will generate an "opcode error".

Translation Notes

IX, IY, and the auxillary register set are mapped into memory locations but these locations are not defined by the translator. If a file using these registers is translated and assembled, "undefined label" errors will result. The file must be edited and the memory locations defined as follows:

```
IX:     DS      2
IY:     DS      2

BC:     DS      2       ;Auxillary register set definition
DE:     DS      2
HL:     DS      2
```

Since IX and IY are mapped into memory locations [IX] and [IY], a memory load or store of IX or IY will translate into a memory-to-memory move. LD IX,(LOC) would become MOV [IX],[LOC]. This is easily corrected by editing and using a register: MOV DI,[LOC]; MOV [IX],DI.

All references to the I (interrupt) and R (refresh) registers will generate an error when the translated file is assembled. The "I" and "R" designations are passed straight through, so that LD I,A becomes MOV I,AL, which would appear to be an attempt to move AL into an undefined immediate.

Blank spaces must not occur within operands. Blanks are equivalent to commas in separating operands.

The input file is assumed to assemble without errors with a Z80 assembler. Errors in input may cause incorrect translation without an error or warning message.

The BIT, SET, and RES instructions require the bit number to be a single digit, 0-7. Use of a label for a bit number, for example, will result in "cannot determine bit number" error.

DJNZ is translated into a decrement followed by jump-if-not-zero. DJNZ, however, does not affect the flags while the decrement does. This is flagged as a warning in the output file and may require special action in some instances.

The parity flag of the 8086 will always be set according to 8080 rules and therefore may not be correct for the Z80. Any jump on parity is flagged with this warning.

Assembly Notes

It is likely that a translated program will be flagged with some errors when assembled by our 8086 Cross Assembler. These errors are usually caused by out-of-range conditional jumps. Since all 8086 conditional jumps must be to within 128 bytes, this type of error is corrected by changing the conditional jump to a reverse-sense conditional jump around a long jump to the target.

For example:

```
        JZ      FARAWAY
```

becomes

```
        JNZ     SKIP1
        JMP     FARAWAY
SKIP1:
```

Other assembly errors may occur because the cross assembler does not have all the features found in some Z80 assemblers, particularly in expression handling, where the only operations are + and -. These errors can only be corrected by finding a way not use the missing feature.

DEBUG — The Resident Debugger

TABLE OF CONTENTS

Introduction

Debug is executed with a command of the form DEBUG <FILENAME>. Debug will load the specified file at 0100 hex in the lowest available segment, and CX will be set to the number of records loaded (see Register command). Currently, Debug will not perform conversion of HEX files; these must have been already converted by HEX2BIN.

Debug commands are available to display, alter and search memory; to do inputs and outputs; to read and write disk files or physical sectors; and to aid in debugging 8086 programs. The debugging commands allow the user to execute a program in a controlled manner, observing its behavior. This controlled execution may be done either by single-stepping or through execution with breakpoints.

Single-stepping is done with Debug's Trace command. By using 8086 hardware trace mode, a single instruction can be executed, and the resulting effects on the registers or memory displayed. Even ROM may be traced, and every instruction is traced correctly (unlike 8080 or Z80 debuggers).

Execution with breakpoints (Go command) allows the user to quickly execute previously tested program portions but stops program execution if a breakpoint is reached. Breakpoints require more care than single-stepping since they can only be used in RAM at the address of the first byte of an 8086 opcode.

Both methods of "controlled execution" allow the user to modify or examine CPU registers. A "register save area" is maintained in memory: just before execution, all registers are set with values from this area; and when control is returned to the monitor, all registers are saved back in this area. The Register command allows this area to be displayed or modified.

Execution of any command may be aborted by typing Control-C. Typing Control-S during output will cause the display to pause so it may be read before scrolling away; any key (except Control-C) may be typed to continue.

Parameters

All commands of Debug accept one or more parameters on the line following the command letter. These parameters MAY be separated from each other and the command letter by spaces or commas, but one these delimiters is REQUIRED only to separate consecutive hex values. Most parameters are one of the following types:

<DRIVE>, <BYTE>, <RECORD>, <HEX4>, <ADDRESS> - A hexadecimal number with no more than 1, 2, 3, 4, or 5 digits, respectively. If too many digits are entered or a non-hex character is typed, the error arrow will point to the mistake. Hex A-F must be in upper case.

<RANGE> - A <RANGE> is either <ADDRESS> <ADDRESS> or <ADDRESS> L <HEX4>. The first form specifies the first and last addresses affected by the command. The second form specifies a starting address and a length. For either form, the maximum length (first address - last address + 1) cannot exceed 10000H, and this limit may be as low as 0FFF1H due to limitations of working within a segment. (Specifically, [starting address modulo 16] + length must be <= 10000H.) An "RG Error" results if the length is too large. To specifiy a length of 10000H with only four digits, use a length of zero. Note that the "L" in this form must be upper case.

<LIST> - This is always the last parameter on a line and may extend to the end of the input buffer. It is actually a series of one or more parameters, each of which is either a <BYTE> or a <STRING>.

A <STRING> is any number of characters enclosed by either single (') or double (") quotes. Since the opening and closing quotes must be the same, the other type may appear in the string freely. If the same quote as opened the string needs to appear within it, it must be given as two adjacent quotes. The ASCII values of the characters in the string are used as a list of bytes.

Commands

A command is executed by typing the first letter of its name (upper case only) followed by any parameters. If a syntax error occurs in a command line, then an arrow followed by the word "Error" will appear under the first bad character. Note that all editing functions listed for the Command Interpreter apply to Debug as well. Commands are listed below in alphabetical order, with the forms of all parameters shown.

D <ADDRESS>
D <RANGE>

Dump - Displays memory contents in hex and ASCII. If only a starting address is specified, 80H bytes are dumped; otherwise the specified range is displayed. To help pinpoint addresses, each line (except possibly the first) begins on a 16-byte boundary, and each 8-byte boundary is marked with a "-". Non-printing characters are shown as a "." in the ASCII dump.

E <ADDRESS> <LIST>
E <ADDRESS>

Enter - In the first form, the list of bytes is entered at the specified address, with the command being executed and completed upon hitting <carriage return>. If an error occurs, NO locations are changed.

The second form puts Debug into "Enter Mode", starting at the specified address. After hitting <carriage return>, the address and its current contents will be displayed. The user now has several options:

1) Replace the displayed value with a new value. Simply type in the new value in hex, using <backspace> or <delete> to correct mistakes. If an illegal hex digit is typed or more than two digits are typed, the bell will sound and the character will not be echoed. After entering the new value, type either <space>, "-", or <carriage return>, as defined below.

2) Type <space> to display and possibly replace the next memory location. Every 8-byte boundary will start a new line with the current address.

3) Type "-" to backup to the preceding memory location. This will always start a new line with the address.

4) Type <carriage return> to terminate the command.

F <RANGE> <LIST>

Fill - The specified range is filled with the values in the list. If the list is larger than the range, not all values will be used; if the range is larger, the list will be repeated as many times as necessary to fill it. All memory in <RANGE> must be valid for this command to work properly. If bad or non-existent memory is encountered, the error will be propagated into all succeeding locations.

G

G <ADDRESS> . . . <ADDRESS>

Go - Sets all registers from the register save area. Since this includes the Code Segment and Instruction Pointer, this implies a jump to the program under test.

This command allows setting up ten breakpoints. Attempting to set more than ten will cause a "BP Error". Breakpoints may be set only at an address containing the first byte of an 8086 opcode. A breakpoint is set by placing an interrupt opcode (0CCH) at the specified address. When that opcode is executed, all registers are saved and displayed, and all breakpoints locations are restored to their original value. If control is not returned to Debug by a breakpoint, the breakpoints will not be cleared.

The user stack pointer must be valid and have 6 bytes available for this command to work. The jump to the user program is made with an IRET instruction with the user stack pointer set and user Flags, Code Segment register, and Instruction Pointer on the user stack. Thus if the user stack is not valid, the system will "crash".

H <ADDRESS> <ADDRESS>

Hex - Performs hexadecimal arithmetic on the two parameters. Two results are returned: the sum of the parameters, and their difference (the first minus the second).

I <HEX4>

Input - Inputs a byte from the specified port and displays it. A 16-bit port address is allowed.

L
L <ADDRESS> <DRIVE> <RECORD> <RECORD>

Load - The first form loads a file into memory. The name of the file must appear in the normal format of a file control block at CS:005C. This requirement is met by a file name typed as the first parameter of the command that started Debug, or the Name command will format a file name properly. The file will be loaded at CS:0100, and the CX will be set to the number of records read.

The second form performs a read of any physical disk sector into any memory area. The first <RECORD> parameter is the logical record number of the first sector to be read. Logical record numbers start at zero and increase sequentially for each sector on the disk, regardless of track boundaries. For example, with standard IBM format single-density disks, logical record zero corresponds to track 0, sector 1, while logical record 37 hex corresponds to track 2, sector 4. The last parameter is the number of records to read.

M <RANGE> <ADDRESS>

Move - Moves the block of memory specified by <RANGE> to <ADDRESS>. Overlapping moves are always performed without loss of data, i.e., data is moved before it is overwritten. To do this, all moves from higher addresses to lower ones are done front-to-back, while moves from lower addresses to higher ones are done back-to-front.

N <FILENAME>

Name - The file name is set up in the proper format of an unopened file control block at CX:005C. An optional second file name is similarly set up at CS:006C. In addition, all characters typed after the command letter "N" are copied into CS:0081, and CS:0080 is set to the number of characters copied. Thus this command may be used to set up a file name for use with the Load or Write commands, but it also formats all standard parameter areas as if the parameters had been part of a command line that executed a COM program. For example, if file PROG.COM is being debugged, it may be desirable to test it as though it had been executed with the command

PROG FILE1 FILE2 OTHERSTUFF

To do this start with

DEBUG PROG.COM

which will execute the debugger and load PROG.COM in to memory. Then use

N FILE1 FILE2 OTHERSTUFF

and the parameters for PROG.COM will be formatted correctly.

O <HEX4> <BYTE>

Output - <BYTE> is sent to the specified output port. A 16-bit port address is allowed.

Q

Quit - Terminate the debugger.

R
R <REGISTER NAME>

Register - with no parameters, this command dumps the register save area.

Giving a register name as a parameter allows that register to be displayed and modified. The register name may be AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, IP, PC, or F (upper case only); anything else will result in an "BR Error". IP and PC both refer to the Instruction Pointer and F refers to the Flag register. For all exept the Flag register, the current 16-bit value will be printed in hex, then a colon will appear as a prompt for the replacement value. Typing <carriage return> leaves the register unchanged; otherwise type a <HEX4> to replace.

The Flag register uses a system of two-letter mnemonics for each flag, as shown below:

FLAG	SET		CLEAR	
Overflow	OV	Overflow	NV	No Overflow
Direction	DN	Down (Decrementing)	UP	Up (Incrementing)
Interrupt	EI	Enabled Interrupts	DI	Disabled Interrupts
Sign	NG	Negative	PL	Plus
Zero	ZR	Zero	NZ	Not Zero
Auxillary Carry	AC	Auxillary Carry	NA	No Auxillary Carry
Parity	PE	Parity Even	PO	Parity Odd
Carry	CY	Carry	NC	No Carry

Whenever the Flag register is displayed, all flags are displayed in this order. When the F register is specified with the R command, the flags are displayed and then Debug waits for any replacements to be made. Any number of two-letter flag codes may be typed, and only those flags entered will be modified. If a flag has more than one code in the list, a "DF Error" (Double Flag) will result. If any code is not recognized, a "BF Error" (Bad Flag) will occur. In either case, those flags up to the error have been changed, and those after the error have not.

On start-up, all registers are set to zero except the segment registers, which are set to the bottom of free memory, the Instruction Pointer, which is set to 0100H, and the Stack Pointer, which is set to 003EH. Flags are all cleared.

S <RANGE> <LIST>

Search - The range is searched for a byte or string of bytes specified by <LIST>. For each occurence the first address of the match is displayed.

T
T <HEX4>

Trace - The number of instructions specified (default 1) are traced. After each instruction, the complete contents of the registers and flags are displayed. (For the meaning of the flag symbols, see Register command.) Since this command uses the hardware trace mode of the 8086, even ROM may be traced.

W
W <ADDRESS> <DRIVE> <RECORD> <RECORD>

Write - The first form writes a portion of memory to a disk file. The name of the file must appear in the normal format of a file control block at CS:005C. This requirement is met by a file name typed as the first parameter of the command that started Debug, or the Name command will format a file name properly. CX must be set to the number of records to be written, and the file will be saved starting at address CS:0100. 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. If the Trace or Go commands are used, CX may be modified by the executing program and then must be reset before using the Write command.

The second form performs a write to any physical disk sector from any memory area. The first <RECORD> parameter is the logical record number of the first sector to be written. Logical record numbers start at zero and increase sequentially for each sector on the disk, regardless of track boundaries. For example, with standard IBM format single-density disks, logical record zero corresponds to track 0, sector 1, while logical record 37 hex corresponds to track 2, sector 4. The last parameter is the number of records to write.

ERROR SUMMARY

BF - Bad Flag

BP - Too many BreakPoints

BR - Bad Register

DF - Double Flag (Flag occurs twice)

RG - RanGe too large

Hard Disk Errors

Should a hard disk error occur, one of the messages "Disk read error" or "Disk write error" will appear on the console. The system then waits for one of the following responses to typed on the console:

"A" - Abort. Terminate the program requesting the disk transfer.

"C" - Continue. If the bad sector is in the File Allocation Table, then the transfer is attempted on a spare allocation table. If no spare allocation tables can be read, the message "All FATs on drive are bad" appears. If the sector is not in the File Allocation Table, then this is similar to Ignore, below.

"I" - Ignore. Pretend the error did not occur.

"R" - Retry.

Another type of error may or may not be related faulty disk transfer. The message "Bad FAT" means the copy in memory of one of the allocation tables has pointers to non-existent disk blocks. This may be caused by using a disk which has not been CLEARed.