SID86UG.WS4 ("SID-86 User's Guide" 2nd Ed. for CP/M-86)

(Retyped by Emmanuel ROCHE.)

"SID-86 User's Guide"
First Edition: March 1982 (With Q, SR, and Z commands added by me.)
Second Edition: August 1982 (Code-macros are still not documented...)

(Pages i and ii missing.)

Table of Contents

Section 1: SID-86 Operation

SID-86 is a powerful symbolic debugger designed for use with the CP/M-86 and MP/M-86 operating systems. It expands on the features of the standard CP/M-86 debugger, DDT-86, allowing users to test and debug programs interactively. SID-86 includes symbolic assembly and disassembly, expressions involving hexadecimal, decimal, ASCII, and symbolic values, permanent breakpoints with pass counts, and trace without call. You should be familiar with the Intel 8086 processor, ASM-86 and the CP/M-86 or MP/M-86 operating system, as described in the "CP/M-86 Operating System System Guide" or "MP/M-86 Operating System System Guide".

1.1 Invoking SID-86

Invoke SID-86 by entering one of the following commands:

(a) SID86
(b) SID86 filename
(c) SID86 filename {,SYMfile...}
(d) SID86 *SYMfile {,SYMfile...}

Form (a) simply loads and executes SID-86. After displaying its sign-on message and prompt character ("#"), SID-86 is ready to accept your commands. Form (b) is similar to form (a), except that, after SID-86 is loaded, it loads the file specified by "filename". If the filetype is omitted from "filename", CMD is assumed. Note that SID-86 cannot load a file of type H86.

Form (c) is similar to form (b) but, once the program is loaded, SID-86 loads the other files in the command line into its symbol table. Form (d) is similar to form (c), except that no program is loaded, only symbol files.

Forms (b), (c), and (d) are analogous to the command sequences:

```
A>SID86
SID86 x.x
#Efilename

A>SID86
SID86 x.x
#Efilename {,SYMfile...}

A>SID86
SID86 x.x
#E*SYMfile {,SYMfile...}
```

At this point, the program that was loaded using form (b) or (c) is ready for execution. (See Section 3.4, "The E Command", for a complete description of file loading.)

1.2 SID-86 Command Conventions

When SID-86 is ready to accept a command, it prompts you with a pound sign ("#"). In response, you can type a command line, or a Ctrl-C to end the debugging session (see Section 1.4). A command line can have up to 64 characters, and must be terminated with a Carriage Return. While entering the command, use standard CP/M-86 line-editing functions (Ctrl-X, Ctrl-H, Ctrl-R, etc) to correct typing errors. SID-86 does not process the command line until a Carriage Return is entered.

The first character of each command line determines the command action. Table 1-1 summarizes SID-86 commands. SID-86 commands are defined individually in Section 3.

Table 1-1. SID-86 Command Summary

Command	Action
A	Enter assembly language statements
B	Compare blocks of memory
D	Display memory in hexadecimal and ASCII
E	Load program and symbols for execution
F	Fill memory block with a constant
G	Begin execution with optional breakpoints
H	Hexadecimal arithmetic
I	Set up File Control Block and command tail
L	List memory using 8086 mnemonics
M	Move memory blocks
P	Set, clear, display pass points
Q	Query I/O ports
R	Read disk file into memory
S	Set memory to new values
SR	SeaRch pattern in memory
T	Trace program execution

U Untraced program monitoring
V Show memory layout of disk file read
W Write contents of memory block to disk
X Examine or modify CPU state
Z Examine 8087 co-processor

The command character can be followed by one or more arguments, which can be symbolic expressions, filenames, or other information, depending on the command. Arguments are separated from each other by commas (",") or spaces. No spaces are allowed between the command character and the first argument. Note that, if the first character of a SID-86 command line is a semicolon (";"), the entire line is treated as a comment and ignored. Several commands (G, P, S, T, and U) can be preceded by a minus sign ("-"). The effect of the minus sign varies between commands. (Section 3, "SID-86 Commands", provides a full explanation of the effect of the minus sign on individual commands.)

1.3 Specifying a 20-Bit Address

Most SID-86 commands require one or more address as operands. Because the 8086 can address up to 1 megabyte of memory, addresses must be 20-bit values. Enter a 20-bit address as follows:

```
ssss:oooo
```

where "ssss" represents an optional 16-bit segment number and "oooo" is a 16-bit offset. SID-86 combines these values to produce a 20-bit address, as follows:

```
  ssss
+ oooo
-------
 aaaaa
```

The segment value, ssss, is optional. If you omit the segment value, SID-86 uses a default value appropriate to the command being executed, as described in Section 3.4, "The E Command".

1.4 Terminating SID-86

Terminate SID-86 by typing a Ctrl-C in response to the pound sign ("#") prompt. This returns control to the CCP. Note that CP/M-86 or MP/M-86 does not have the SAVE facility found in CP/M for 8-bit machines. Thus, if SID-86 is used to patch a file, write the file to disk using the W command before exiting SID-86.

1.5 SID-86 Operation with Interrupts

SID-86 operates in systems with interrupts enabled or disabled, and preserves

the interrupt state of the program being executed under SID-86. When SID-86 controls the CPU, either when initially invoked or when regaining control from the program being tested, the condition of the interrupt flag is the same as it was when SID-86 was invoked, except for a few critical regions where interrupts are disabled. While the program being tested has control of the CPU, the user's CPU state (which can be displayed with the X command) determines the state of the interrupt flag.

Section 2: SID-86 Expressions

An important facility of SID-86 is the ability to reference absolute machine addresses through expressions. Expressions can involve names obtained from the program under test that are included in the SYM file produced by ASM-86. Expressions can also consist of literal values in hexadecimal, decimal, or ASCII character string form. You can then combine these values with various operators, to provide access to subscripted and indirectly-addressed data or program areas. This section describes expressions, so that you can incorporate them as command parameters in the individual command forms that follow this section.

2.1 Literal Hexadecimal Numbers

SID-86 normally accepts and displays values in hexadecimal. The valid hexadecimal digits consist of the decimal digits 0 through 9 and the hexadecimal digits A, B, C, D, E, and F, corresponding to the decimal values 10 through 15, respectively.

A literal hexadecimal number in SID-86 consists of one or more contiguous hexadecimal digits. If you type four digits, then the leftmost digit is most significant, while the rightmost digit is least significant. If the number contains more than four digits, the rightmost four are taken as significant, and the remaining leftmost digits are discarded. The examples below show the corresponding hexadecimal and decimal values for the given input values.

Input Value	Hexadecimal	Decimal
1	0001	1
100	0100	256
fffe	FFFE	65534
10000	0000	0
38001	8001	32769

2.2 Literal Decimal Numbers

Enter decimal numbers by preceding the number with the # symbol. In this case, the number that follows must consist of one or more decimal digits (0 through 9) with the most significant digit on the left, and the least significant digit on the right. Decimal values are padded or truncated according to the

rules of hexadecimal numbers, as described above, by converting the decimal number to the equivalent hexadecimal value.

The input values shown to the left below produce the internal hexadecimal values shown to the right below:

Input Value	Hexadecimal
#9	0009
#10	000A
#256	0100
#65535	FFFF
#65545	0009

2.3 Literal Character Values

SID-86 accepts one or two graphic ASCII characters enclosed in apostrophes (') as literal values in expressions. Characters remain as typed within the paired apostrophes (i.e., no case translation occurs) with the leftmost character treated as the most significant, and the rightmost character treated as least significant. Character strings of length one are padded on the left with zero. Strings of length greater than zero are not allowed in expressions, except as described in the S command.

Note that the enclosing apostrophes are not included in the character string, nor are they included in the character count, with one exception: a pair of contiguous apostrophes is reduced to a single apostrophe and included in the string as a normal graphic character.

The strings shown to the left below produce the hexadecimal values shown to the right below. (For these examples, note that upper-case ASCII alphabetics begin at the encoded hexadecimal value 41; lower-case alphabetics begin at 61; a space is hexadecimal 20, and an apostrophe is encoded as hexadecimal 27.)

Input String	Hexadecimal
'A'	0041
'AB'	4142
'aA'	6141
''	0027
''''	2727
' A'	2041
'A '	4120

2.4 Register Values

You can use the contents of registers in the CPU state of the program under test in expressions. Simply use the register name wherever a number would normally be valid. For example, if you know that, at a certain point in the program, the BX register points to a data area you want to see, the command

displays the desired area of memory.

Note that, when assembling 8086 instructions using the A command, register names are treated differently than in other expressions. In particular, use of a register name in an assembly language statement entered in the A command refers to the name of a register, and not its contents.

2.5 Stack References

Elements in the stack can be included in expressions. A caret ("^") refers to the 16-bit value at the top of the stack (pointed to by the SS and SP registers in the user's CPU state). A sequence of n carets refer to the nth 16-bit value on the stack. You can use this feature to set a breakpoint on return from a subroutine, when all that is known is that the return address is on the stack, even though the actual value is not known. The commands

```
G,^
G,^^:^
```

set breakpoints on return from near and far subroutines, respectively.

2.6 Symbolic References

Given that a symbol table is present during a SID-86 debugging session, you can reference values associated with symbols through the following three forms of a symbol reference:

(a) .s
(b) @s
(c) =s

where "s" represents a sequence of one to thirty-one characters that match a symbol in the table.

Form (a) produces the 16-bit value corresponding to the symbol "s" (i.e., the value associated with the symbol in the table). Form (b) produces the 16-bit value contained in the two memory locations given by .s, while form (c) results in the 8-bit value at .s in memory. Note that forms (b) and (c) use the contents of the DS register as the segment component when fetching the 16-bit or 8-bit contents of memory. Suppose, for example, that the input symbol table contains two symbols, and appears as follows:

```
0100 GAMMA      0102 DELTA
```

Further, suppose that memory starting at 0100 in the segment referred to by the DS register contains the following byte data values:

0100: 02 3E 4D 22

Based upon this symbol table and these memory values, the symbol references shown to the left below produce the hexadecimal values shown to the right below. Recall that 16-bit 8086 memory values are stored with the least significant byte first, and thus the word values at 0100 and 0102 are 3E02 and 224D, respectively.

Symbol Reference	Hexadecimal
.GAMMA	0100
.DELTA	0102
@GAMMA	3E02
@DELTA	224D
=GAMMA	0002
=DELTA	004D

2.7 Qualified Symbols

Duplicate symbols can occur in the symbol table, due to separately assembled or compiled modules that independently use the same name for different subroutines or data areas. Further, block structured languages allow nested name definitions that are identical, but non-conflicting. Thus, SID-86 allows reference to qualified symbols that take the form:

S1/S2/.../Sn

where "S1" through "Sn" represents symbols that are present in the table during a particular session.

SID-86 always searches the symbol table from the first to last symbol, in the order the symbols appear in the symbol file. For a qualified symbol, SID-86 begins by matching the first S1 symbol, then scans for a match with symbol S2, continuing until symbol Sn is matched. If this search and match procedure is not successful, SID-86 prints the "?" response to the console. Suppose, for example, that the symbol table appears in the symbol file as follows:

0100 A 0300 B 0200 A 3E00 C 20F0 A 0102 A

with memory initialized as shown in the previous section. In the following example, the unqualified and qualified symbol references shown to the left produce the hexadecimal values shown to the right.

Symbol Reference	Hexadecimal
.A	0100
@A	3E02
.A/A	0200
.C/A/A	0102
=C/A/A	004D
.B/A/A	20F0

2.8 Operators in Expressions

Literal numbers, strings, and symbol references can be combined into symbolic expressions using unary and binary "+" and "-" operators. The entire sequence of numbers, symbols, and operators must be written without embedded blanks. Further, the sequence is evaluated from left to right, producing a four-digit hexadecimal value at each step in the evaluation. Overflow and underflow are ignored as the evaluation proceeds. The final value becomes the command parameter, whose interpretation depends upon the particular command letter that precedes it.

When placed between two operands, the "+" indicates "addition" to the previously accumulated value. The sum becomes the new accumulated value to this point in the evaluation.

The "-" symbol causes SID-86 to subtract the literal number or symbol reference from the 16-bit value accumulated thus far in the symbolic expression. If the expression begins with a minus sign ("-"), then the initial value is taken as zero. That is to say,

```
-x
```

is computed as

```
0-x
```

where "x" is any valid symbolic expression. For example, the following command:

```
DFF00-200,-#512
```

is equivalent to the simple command:

```
DFD00,FE00
```

In commands that specify a range of addresses (B, D, L, F, M, and W), the ending address of the range can be indicated as an offset from the starting address. To do this, precede the desired offset by a plus sign ("+"). For example, the command

```
D121,+7
```

displays the memory from address 121 to 128 (121 + 7). Use of the unary plus operator at other times is not allowed.

2.9 Sample Symbolic Expressions

The formulation of SID-86 symbolic expressions is most often closely related to the program structures in the program under test. Suppose you want to debug a sorting program that contains the data items listed below:

LIST: names the base of a table of byte values to sort, assuming there are no more than 255 elements, denoted by LIST(0), LIST(1), ..., LIST(254).

N: is a byte variable that gives the actual number of items in LIST, where the value of N is less than 256. The items to sort are stored in LIST(0) through LIST(N-1).

I: is the byte subscript that indicates the next item to compare in the sorting process. LIST(I) is the next item to place in sequence, where I is in the range 0 through N-1.

Given these data areas, the command

D.LIST,+#254

displays the entire area reserved for sorting:

LIST(0), LIST(1), ..., LIST(254)

The command

D.LIST,+=i

displays the LIST vector up to and including the next item to sort:

LIST(0), LIST(1), ..., LIST(I)

The command

D.LIST+=I,+0

displays only LIST(I). Finally, the command

D.LIST,+=N-1

displays only the area of LIST that holds active items to sort:

LIST(0), LIST(1), ..., LIST(N-1)

Section 3: SID-86 Commands

This section defines SID-86 commands and their arguments. SID-86 commands give you control of program execution, and allow you to display and modify system memory and the CPU state.

3.1 The A (Assemble) Command

The A command assembles Intel 8086 mnemonics directly into memory. The form

is:

```
As
```

where "s" is the 20-bit address where assembly starts. SID-86 responds to the A command by displaying the address of the memory location where assembly begins. At this point, you enter assembly language statements as described in Section 5 on Assembly Language Syntax. When a statement is entered, SID-86 converts it to binary, places the value(s) in memory, and displays the address of the next available memory location. This process continues until you enter a blank line or a line containing only a period (".").

SID-86 responds to invalid statements by displaying a question mark ("?") and redisplaying the current assembly address.

Note that, wherever a numeric value is valid in an assembly language statement, an expression can be entered. There is one difference between expressions in assembly language statements and those appearing elsewhere in SID-86: under the A command, references to registers refer to the names of the registers, while elsewhere they refer to the contents of the registers. Under the A command, there is no way to reference the contents of a register in an expression.

The following are examples of the A command.

```
#A213                Assemble at offset 0213.
ssss:0213 mov ax,#128   Set AX register to decimal 128.
ssss:0216 push ax       Push AX register on stack.
ssss:0217 call .proc1   Call procedure whose address is the value of
                  the symbol PROC1.

ssss:021A test byte [.i/i],80
```

Test the most significant bit of the byte whose address is the value of the second occurrence of the symbol I.

```
ssss:021E jz .done      Jump (if Zero flag set) to the location whose
                  address is the value of the symbol DONE.

ssss:0220 mov al,[.array+4]
```

Move the contents of the memory byte whose address is the value of the symbol ARRAY plus 4 to the AL register.

3.2 The B (Block Compare) Command

The B command compares two blocks of memory, and displays any discrepancies at the screen. The form is:

```
Bs1,f1,s2
```

where "s1" is the 20-bit address of the start of the first block; "f1" is the

offset of the final byte of the first block; and "s2" is the 20-bit address of the start of the second block. If the segment is not specified in s2, the same value is used that was used for s1.

Any differences in the two blocks are displayed at the screen in the form:

    a1 b1 a2 b2

where the "a1" and the "a2" are the adresses in the blocks; "b1" and "b2" are the values at the indicated addresses. If no differences are displayed, the blocks are identical.

The following are examples of the B command.

    #B40:0,1FF,60:0

Compares 200h bytes starting at 40:0 with the block starting at 60:0.

    #Bes:.array1,+ff,.array2

Compares 256 bytes starting at offset ARRAY1 in the Extra Segment with ARRAY2 in the Extra Segment.

3.3 The D (Display) Command

The D command displays the contents of memory as8-bit or 16-bit hexadecimal values and in ASCII. The forms are:

    (a) D
    (b) Ds
    (c) Ds,f
    (d) DW
    (e) DWs
    (f) DWs,f

where "s" is the 20-bit address where the display starts, and "f" is the 16-bit offset within the segment specified in "s" where the display finishes.

Memory is displayed on one or more display lines. Each display line shows the values of up to 16 memory locations. For the first three forms, the display line appears as follows:

    ssss:oooo bb bb ... bb cc...c

where "ssss" is the segment being displayed, and "oooo" is the offset within segment "ssss". The bb's represent the contents of the memory locations in hexadecimal, and the c's represent the contents of memory in ASCII. A period (".") represents any non-graphic ASCII character.

In response to form (a), SID-86 displays memory from the current display address for 12 display lines. The response to form (b) is similar to form (a), except that the display address is first set to the 20-bit address "s". Form

(c) displays the memory block between locations "s" and "f".

```
#rascii.bin
  START     END
1CC1:0000 1CC1:00FF
#d0,ff
1CC1:0000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ................
1CC1:0010 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F ................
1CC1:0020 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F  !"#$%&'()*+,-./
1CC1:0030 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789:;<=>?
1CC1:0040 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHIJKLMNO
1CC1:0050 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[\]^_
1CC1:0060 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F `abcdefghijklmno
1CC1:0070 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F pqrstuvwxyz{|}~.
1CC1:0080 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F ................
1CC1:0090 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F ................
1CC1:00A0 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF ................
1CC1:00B0 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF ................
1CC1:00C0 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ................
1CC1:00D0 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF ................
1CC1:00E0 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF ................
1CC1:00F0 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF ................
```

The next three forms are analogous to the first three, except that the contents of memory are displayed as 16-bit values, rather than 8-bit values, as shown below:

```
#dw0,ff
1CC1:0000 0100 0302 0504 0706 0908 0B0A 0D0C 0F0E ................
1CC1:0010 1110 1312 1514 1716 1918 1B1A 1D1C 1F1E ................
1CC1:0020 2120 2322 2524 2726 2928 2B2A 2D2C 2F2E  !"#$%&'()*+,-./
1CC1:0030 3130 3332 3534 3736 3938 3B3A 3D3C 3F3E 0123456789:;<=>?
1CC1:0040 4140 4342 4544 4746 4948 4B4A 4D4C 4F4E @ABCDEFGHIJKLMNO
1CC1:0050 5150 5352 5554 5756 5958 5B5A 5D5C 5F5E PQRSTUVWXYZ[\]^_
1CC1:0060 6160 6362 6564 6766 6968 6B6A 6D6C 6F6E `abcdefghijklmno
1CC1:0070 7170 7372 7574 7776 7978 7B7A 7D7C 7F7E pqrstuvwxyz{|}~.
1CC1:0080 8180 8382 8584 8786 8988 8B8A 8D8C 8F8E ................
1CC1:0090 9190 9392 9594 9796 9998 9B9A 9D9C 9F9E ................
1CC1:00A0 A1A0 A3A2 A5A4 A7A6 A9A8 ABAA ADAC AFAE ................
1CC1:00B0 B1B0 B3B2 B5B4 B7B6 B9B8 BBBA BDBC BFBE ................
1CC1:00C0 C1C0 C3C2 C5C4 C7C6 C9C8 CBCA CDCC CFCE ................
1CC1:00D0 D1D0 D3D2 D5D4 D7D6 D9D8 DBDA DDDC DFDE ................
1CC1:00E0 E1E0 E3E2 E5E4 E7E6 E9E8 EBEA EDEC EFEE ................
1CC1:00F0 F1F0 F3F2 F5F4 F7F6 F9F8 FBFA FDFC FFFE ................
```

During a long display, you can abort the D command by typing any character at the console.

The following are examples of the D command.

`#DF00,F23` Display memory bytes from offset 0F00h through 0F23h in the current Data Segment.

`#D.array+=i,+#10` Display 11 bytes starting at the location of

ARRAY(I).

#DW#128,#255 Displays memory words from offset 0080h through 00FFh.

3.4 The E (Load Program, Symbols for Execution) Command

The E command loads a file into memory so that a subsequent G, T, or U command can begin program execution, and allows one or more symbol table files to be loaded. The E command takes the forms:

(a) E<filename>
(b) E<filename> <symbol filename> {,<symbol filename>...}
(c) E*<symbol filename> {,<symbol filename>...}
(d) E

Form (a) loads the file by the name <filename> using the BDOS load function. The file is assumed to be in the CMD file format, as described in the "CP/M-86 System Guide". If no filetype is specified, CMD is assumed. The contents of the CS, DS, ES, SS, and IP registers are altered according to the information in the Header Record of the CMD file loaded. When the load is complete, SID-86 displays the start and end addresses of each segment in the file loaded. Use the V command to redisplay this information at a later time.

Form (b) loads the file <filename> as described above, and then loads one or more symbol table files. If the filetype is omitted from a symbol filename, SYM is assumed. SID-86 displays the message:

SYMBOLS

when it begins loading the symbol file(s). If SID-86 detects an invalid hex digit or an invalid symbol name, it displays an error message and stops loading the symbol file. The symbols loaded up to the time the error occurred can be displayed with the H command, to determine the exact location of the error in the SYM file. A maximum of 64K bytes is available for symbol table storage.

While SID-86 allows symbol files to be loaded with separate E commands, each E command requests another memory allocation from the operating system. Therefore, if you are loading more than one symbol table file, you should load all the files with a single E command, using form (b) or (c).

Form (c) does not load a program, but simply loads the indicated symbol table file(s).

Form (d) releases all memory allocations made from SID-86, without loading any files.

When loading a program file with the E command, SID-86 releases any blocks of memory allocated by any previous E or R commands or by programs executed under SID-86. Thus, only one file at a time can be loaded for execution, and that file should be loaded before any symbol tables are read.

SID-86 issues an error message if a file does not exists or cannot be successfully loaded in the available memory.

The format of the symbol table file is that produced by ASM-86, as follows:

```
nnnn symbol1    nnnn symbol2    ...
...
...
```

where "nnnn" is a four-digit hexadecimal number, and spaces, tabs, Carriage Returns, and Line Feeds can serve as delimiters between hex values and symbol names. Symbol names can be up to thirty-one characters in length.

The following are examples of the E command.

```
#Etest                Load file TEST.CMD.

#Etest.cmd test.sym   Load file TEST.CMD and symbol table file
                      TEST.SYM.

#Etest test io format Load file TEST.CMD and symbol table files
                      TEST.SYM, IO.SYM, and FORMAT.SYM.

#E*test1              Load only the symbol table file TEST1.SYM.
```

3.5 The F (Fill) Command

The F command fills an area of memory with a byte or word constant. The forms are:

(a) Fs,f,b
(b) FWs,f,w

where "s" is a 20-bit starting address of the block to be filled, and "f" is a 16-bit offset of the final byte of the block within the segment specified in "s".

In response to form (a), SID-86 stores the 8-bit value "b" in locations "s" through "f". In form (b), the 16-bit value "w" is stored in location "s" through "f" in standard form, low 8-bits first followed by high 8-bits.

If "s" is greater than "f" or the value "b" is greater than 255, SID-86 responds with a question mark ("?"). SID-86 issues an error message if the value stored in memory cannot be read back successfully, indicating faulty or non-existant RAM at the location indicated.

The following are examples of the F command.

```
#F100,13F,0           Fill memory from 0100h to 013Fh with 00h.

#F.array,+255,ff      Fill the 256-byte block starting at ARRAY with
```

the constant 0FFh.

3.6 The G (Go) Command

The G command transfers control to the program being tested, and optionally sets one or two breakpoints. The forms are:

(a) G
(b) G,b1
(c) G,b1,b2
(d) Gs
(e) Gs,b1
(f) Gs,b1,b2
(g) -G (with forms "a" through "f")

where "s" is a 20-bit address where program execution is to start, and "b1" and "b2" are 20-bit addresses of breakpoints. If no segment value is supplied for any of these three addresses, the segment value defaults to the contents of the CS register.

In forms (a), (b), and (c), no starting address is supplied, so SID-86 derives the 20-bit address from the CS and IP registers. Form (a) transfers control to your program without setting any breakpoints. Form (b) and (c) set one and two breakpoints, respectively, before passing control to your program. Forms (d), (e), and (f) are analogous to (a), (b), and (c), except that the CS and IP registers are first set to "s".

The forms in (g) are analogous to forms (a) through (f), except that intermediate pass points displays are suppressed.

Once control has been transferred to the program under test, it executes in real-time until a breakpoint is encountered. At this point, SID-86 regains control, clears the breakpoints set by the G command, and indicates the address at which execution of the program under test was interrupted as follows:

*ssss:oooo .symbol

"ssss" corresponds to the CS, "oooo" corresponds to the IP where the break occurred, and ".symbol" is the symbol whose value is equal to "oooo", if such a symbol exists. When a breakpoint returns control to SID-86, the instruction at the breakpoint address has not yet been executed.

The following are examples of the G command.

#G Begin program execution at address given by CS and IP registers, with no breakpoints set.

#G.start,.error Begin program execution at label START in the code segment, setting a breakpoint at label ERROR.

#G,.error,^ Continue program execution at address given by CS and

IP registers, with breakpoints at label ERROR and at the address at the top of the stack (the return address of the procedure being executed).

#-G,34F Begin execution with a breakpoint at 034Fh, suppressing intermediate pass point display.

3.7 The H (Hexadecimal Math) Command

The H command provides several useful arithmetic functions. The forms are:

(a) Ha,b
(b) Ha
(c) H

Form (a) computes the sum (ssss), difference (ddd), product (pppppppp), and quotient (qqqq) with the remainder (rrrr) of two 16-bit values. The results are displayed in hexadecimal as follows:

+ ssss - dddd * pppppppp / qqqq (rrrr)

Underflow and overflow are ignored in addition and subtraction.

Form (b) displays the value of the expression "a" in hexadecimal, decimal, ASCII (if the value has a graphic ASCII equivalent), and symbolic (if a symbol exists with a value equal to the value of the expression) form as shown:

hhhh #ddddd 'c' .s

Form (c) displays the symbols currently loaded in the SID-86 symbol table. Each symbol is displayed in the form:

nnnn <symbol name>

You can abort the display by pressing any key at the console.

The following are examples of the H command.

#H List all symbols and values loaded with E command(s).

#H.open Show the value of the symbol OPEN in hex and decimal.

#H@index Show the word contents of the memory location at INDEX in hex and decimal.

#H5C28,80 Show sum, difference, product, and quotient of 5C28h and 0080h.

3.8 The I (Input Command Tail) Command

The I command prepares a file control block (FCB) and DMA buffer in SID-86's Base Page, and copies this information into the Base Page of the last file loaded with the E command. The command takes the following form:

I<command tail>

where <command tail> is a character string that usually contains one or more filenames. The first filename is parsed into the default file control block at 005Ch. The optional second filename is parsed into the second part of the default file control block beginning at 006Ch. The characters in <command tail> are also copied into the default DMA buffer at 0080h. The length of <command tail> is stored at 0080h, followed by the character string terminated with a binary zero.

If a file has been loaded with the E command, SID-86 copies the file control block and DMA buffer from the Base Page of SID-86 to the Base Page of the program loaded. The location of SID-86's Base Page can be obtained from the 16-bit value at location 0:6. The location of the Base Page of a program loaded with the E command is the value displayed for DS upon completion of the program load.

```
A>sid
SID86 1.01   1/4/83
#edumpasc
    START      END
CS 1CC1:0000 1CC1:013F
DS 1CD5:0000 1CD5:015F
#idumpasc.cmd
#dw0:6,7
0000:0006 1850 P.
#d1850:0
1850:0000 0F 47 00 50 18 01 0F 47 00 50 18 01 00 00 00 00 .G.P...G.P......
1850:0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:0050 00 00 00 00 00 00 00 00 00 00 00 00 00 44 55 4D .............DUM
1850:0060 50 41 53 43 20 43 4D 44 00 00 00 00 00 20 20 20 PASC CMD.....
1850:0070 20 20 20 20 20 20 20 20 00 00 00 00 00 00 00 00         ........
1850:0080 0B 44 55 4D 50 41 53 43 2E 43 4D 44 00 00 00 00 .DUMPASC.CMD....
1850:0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
#d
1850:00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1850:0100 E9 BA 01 E9 F5 03 E9 B1 03 E9 C2 03 E9 FE 12 E9 ................
1850:0110 F6 0B E9 DD 11 E9 3C 12 E9 71 0E CD E0 20 43 4F ......<..q... CO
1850:0120 50 59 52 49 47 48 54 20 28 43 29 20 31 39 38 32 PYRIGHT (C) 1982
1850:0130 2C 20 44 49 47 49 54 41 4C 20 52 45 53 45 41 52 , DIGITAL RESEAR
1850:0140 43 48 20 00 00 00 00 00 00 53 49 44 38 36 20 31 CH ......SID86 1
1850:0150 2E 30 31 20 20 20 31 2F 34 2F 38 33 A0 41 D8 42 .01   1/4/83.A.B
```

```
1850:0160 D8 43 D8 44 D8 53 D0 42 D0 53 C9 44 C9 43 D3 44 .C.D.S.B.S.D.C.D
1850:0170 D3 53 D3 45 D3 49 D0 4F 44 49 54 53 5A 41 50 43 .S.E.I.ODITSZAPC
#v
    START     END
CS 1CC1:0000 1CC1:013F
DS 1CD5:0000 1CD5:015F
#dds:0
1CD5:0000 3F 01 00 C1 1C 00 5F 01 00 D5 1C 00 00 00 00 00 ?.....?.....
1CD5:0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:0050 00 00 00 00 00 00 00 00 00 00 00 00 00 44 55 4D .............DUM
1CD5:0060 50 41 53 43 20 43 4D 44 00 00 00 00 00 20 20 20 PASC CMD.....
1CD5:0070 20 20 20 20 20 20 20 20 00 00 00 00 00 00 00 00         ........
1CD5:0080 0B 44 55 4D 50 41 53 43 2E 43 4D 44 00 00 00 00 .DUMPASC.CMD....
1CD5:0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
#d
1CD5:00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:0100 44 55 4D 50 41 53 43 2D 38 36 20 62 79 20 45 6D DUMPASC-86 by Em
1CD5:0110 6D 61 6E 75 65 6C 20 52 4F 43 48 45 0D 0A 07 46 manuel ROCHE...F
1CD5:0120 69 6C 65 20 6E 6F 74 20 66 6F 75 6E 64 2E 24 0D ile not found.$.
1CD5:0130 0A 07 4E 6F 20 72 65 63 6F 72 64 73 20 65 78 69 ..No records exi
1CD5:0140 73 74 20 28 7A 65 72 6F 2D 6C 65 6E 67 74 68 20 st (zero-length
1CD5:0150 66 69 6C 65 29 2E 24 00 00 00 00 00 00 00 00 00 file).$.........
1CD5:0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CD5:0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

The following are examples of the I command.

```
#Ifile1.cmd
```

Set up a file control block at 005Ch for the file FILE1.CMD, and put the string "FILE1.CMD" in the DMA buffer at 0080h (in the Data Segment of the last file loaded with an E command).

```
#Ia:file1 b:file2 c:file3 $px
```

Set up file control blocks at 005Ch and 006Ch for the files A:FILE1 and B:FILE2, and copy the string following the "I" into the DMA buffer at 0080h.

3.9 The L (List) Command

The L command lists the contents of memory in assembly language. The forms are:

(a) L

(b) Ls
(c) Ls,f
(d) -L
(e) -Ls
(f) -Ls,f

where "s" is a 20-bit address where the list starts, and "f" is a 16-bit offset within the segment specified in "s" where the list finishes.

Each disassembled instruction is in the following form:

label:
ssss:oooo <prefixes> opcode <operands> <.symbol> <=memory value>

where "label" is the symbol whose value is equal to the offset "oooo", if such a symbol exists; <prefixes> are segment override, lock and repeat prefixes; "opcode" is the mnemonic for the instruction, <operands> field contains 0, 1, or 2 operands, as required by the instruction, and <symbol> is the symbol whose value is equal to the numeric operand, if there is one and such a symbol exists. If the instruction references a memory location, the contents of the location are displayed in the <memory value> field as a byte, word, or double word, as indicated by the instruction.

Form (a) lists twelve disassembled instructions from the current list address. Form (b) sets the list address to "s" and then lists twelve instructions. Form (c) lists disassembled code from "s" through "f". The last three forms are analogous to the first three, except that no symbolic information is displayed (the labels and <symbol> fields are omitted).

In all cases, the list address is set to the next unlisted location, in preparation for a subsequent L command. When SID-86 regains control from a program being tested (see G, T, and U commands), the list address is set to the current value of the CS and IP registers.

You can abort long displays by typing any key during the list process. Or, enter Ctrl-S to halt the display temporarily.

The syntax of the assembly language statements produced by the L command is described in Section 5, "Assembly Language Syntax for A and L Commands".

If the memory location being disassembled is not a valid 8086 instruction, SID-86 displays it in the form:

??= nn

where "nn" is the hexadecimal value of the contents of the memory location.

The following are examples of the L command.

#L243C,244E Disassemble instructions from 243Ch through 244Eh.

#L.find,+20 Disassemble 20h bytes from the label FIND.

#L.err+3 Disassemble 12 lines of code from the label ERR plus 3.

3.10 The M (Move) Command

The M command moves a block of data values from one area of memory to another. The form is:

```
Ms,f,d
```

where "s" is the 20-bit starting address of the block to be moved; "f" is the offset of the final byte to be moved within the segment described by "s", and "d" is the 20-bit address of the first byte of the area to receive the data. If the segment is not specified in "d", the same value is used that was used for "s".

Note that, if "d" is between "s" and "f", part of the block being moved is overwritten before it is moved, because data is transferred starting from location "s".

The following are examples of the M command.

```
#M20:2400,+9,30:100     Move 10 bytes from 20:2400 to 30:100.

#M.array,+#63,.array2   Move 64 bytes from ARRAY to ARRAY2.
```

3.11 The P (Pass Point) Command

The P command sets, clears, and displays pass points. The forms are:

(a) Pa,n
(b) Pa
(c) -Pa
(d) -P
(e) P

A pass point is a permanent breakpoint that remains in effect until it is explicitly removed, as opposed to breakpoints set with the G command, that must be re-entered with each G command. Pass points have associated pass counts, ranging from 1 to 0FFFFh. The pass count indicates how many times the instruction at the pass point executes before the control returns to the console. Up to sixteen pass points can be set at a time.

An important distinction between breakpoints and pass points is that, when execution stops at a breakpoint, the instruction at the breakpoint has not yet been executed. When execution stops due to a pass point whose pass count has reached 1, the instruction at the pass point has been executed. This makes it simple to proceed from a pass point with a G command, without immediately encountering the same pass point.

Forms (a) and (b) are used to set pass points. Form (a) sets a pass point at address "a" with a pass count of "n", where "a" is the 20-bit address of the

pass point, and "n" is the pass count, in the range 1 to 0FFFFh. If a pass point is already active at "a", the pass count is simply changed to "n". SID-86 responds with a question mark if there are already 16 active pass points.

Form (b) sets a pass point at "a" with a pass count of 1. If a pass point is already active at "a", the pass count is simply changed to 1. SID-86 responds with a question mark if there are already 16 active pass points.

Forms (c) and (d) are used to clear pass points. Form (c) clears the pass point at location "a". SID-86 responds with a question mark if there is no pass point set at "a". Form (d) clears all the pass points.

Form (e) displays all the active pass points in the form:

nnnn ssss:oooo .symbol

where "nnnn" is the current pass count for the pass point, "ssss:oooo" is the segment and offset of the pass point location, and ".symbol" is the symbolic name of the offset of the pass point, if such a symbol exists.

When a pass point is encountered, SID-86 displays the pass point information in the form:

nnnn PASS ssss:oooo .symbol

where "nnnn", "ssss:oooo", and ".symbol" are as described above. Next, SID-86 displays the CPU state before the instruction at the pass point is executed. SID-86 then executes the instruction at the pass point. If the pass point is greater than 1, SID-86 decrements the pass count and transfers control back to the program under test.

When the pass count reaches 1, SID-86 displays the break address (that of the next instruction to be excuted) in the following form:

*ssss:oooo .symbol

Once the pass count reaches 1, it remains at 1 until the pass point is cleared or the pass count is changed with another P command.

Use pass points in conjunction with the G, T, and U commands. When you use the G or U commands, you can suppress the intermediate pass point display with the -G or -U forms (see Section 3.6, "The G Command", and Section 3.17, "The U Command"). In this case, only the final pass points (when the pass count = 1) are displayed. You can interrupt the G or U command before its normal termination by pressing any key at the console. SID-86 aborts the G or U command when the next pass point is encountered, and prompts for the next command.

You can also use pass points in conjunction with breakpoints set with the G command. If a pass point and a breakpoint are set at the same address, the breakpoint is encountered first. Otherwise, pass points behave in the usual manner, decrementing the pass count until it reaches 1, and then returning control to SID-86.

Normally, the segment registers are not displayed at pass points. You can use the S/-S command to enable/disable the segment register display (See Section 3.14, "The S Command").

The following are examples of the P command.

#P Display active pass points.

#P.error Set pass point at label ERROR.

#P.print,17 Set pass point at label PRINT with count of 17h.

#-P Clear all pass points.

#-P.error Clear pass point at label ERROR

3.12 The QI, QO (Query I/O) Command

The QI and QO commands allow access to any of the 65,536 Input/Output ports of the Intel 8086 processor. The QI command reads data from a port; the QO command writes data to a port.

The forms of the QI command are:

(a) QIn
(b) QIWn

where "n" is the 16-bit port number. In the case of form (a), SID-86 displays the 8-bit value read from port "n". In the case of form (b), SID-86 displays a 16-bit value from port "n".

The forms of the QO command are:

(a) QOn,v
(b) QOWn,v

where "n" is the 16-bit port number, and "v" is the value to output. In the case of form (a), the 8-bit value "v" is written to port "n". If "v" is greater than 255, SID-86 responds with a question mark. In the case of form (b), the 16-bit value "v" is written to port "n".

3.13 The R (Read) Command

The R command reads a file into a contiguous block of memory. The forms are:

(a) R<filespec>
(b) R<filespec>,s

where <filespec> is the name and type of the file to be read, and "s" is the location to which the file is read. Form (a) lets SID-86 determine the memory

location into which the file is read. Form (b) causes SID-86 to read the file into the memory segment beginning at "s". This address can have the standard form (ssss:oooo) as in the following example:

```
#Rcpm.sys,1000:0
```

The low-order four bits of "s" are assumed to be zero, so SID-86 reads files on a paragraph boundary. If the memory at "s" is not available, SID-86 issues the message:

```
MEMORY REQUEST DENIED
```

SID-86 reads the file into memory, computes, allocates, and displays the start and end addresses of the block of memory occupied by the file. A V command can redisplay this information at a later time. The default display pointer (for subsequent D commands) is set to the start of the block occupied by the file.

The R command does not free any memory previously allocated by another R or E command. Thus, a number of files can be read into memory without overlapping. The number of files that can be loaded is limited to seven under CP/M-86, which is the number of memory allocations allowed by the BDOS, minus one for SID-86 itself.

SID-86 issues an error message if the file does not exist or there is not enough memory to load the file.

The following are examples of the R command.

```
#Rsid86.cmd     Read file SID86.CMD into memory.

#Rtest          Read file TEST into memory.

#Rtest,1000:0   Read file TEST into memory, starting at location
                1000:0.
```

3.14 The S (Set) Command

The S command changes the contents of bytes or words of memory. The forms are:

(a) Ss
(b) SWs
(c) S
(d) -S

where "s" is the 20-bit address where the change occurs.

SID-86 displays the memory address and its current contents on the following line. In response to form (a), the display is:

```
ssss:oooo bb
```

and, in response to form (b); the display is:

ssss:oooo wwww

where "bb" and "wwww" are the contents of memory in byte and word formats, respectively.

In response to one of the above displays, you can choose to alter the memory location or to leave it unchanged. If you enter a valid expression, the contents of the byte (or word) in memory is replaced with the value of the expression. If you do not enter a value, the contents of memory are unaffected and the contents of the next address are displayed. In either case, SID-86 continues to display successive memory addresses and values until you enter a period (".") on a line by itself or until SID-86 detects an invalid expression.

In response to form (a), you can enter a string of ASCII characters, beginning with a quotation mark (') and ending with a Carriage Return. The characters between the quotation mark and the Carriage Return are placed in memory starting at the address displayed. No case conversion takes place. The next address displayed is the address following the character string.

SID-86 issues an error message if the value stored in memory cannot be read back successfully, indicating faulty or non-existant RAM at the location indicated.

The forms (c) and (d) control the display of the segment registers when the CPU state is displayed with the trace command and at pass points. Form (c) turns ON the segment register display, and form (d) turns it OFF. It is often convenient to turn OFF the segment register display while debugging, to allow the CPU state display to fit on one line.

The following are examples of the S command.

```
#S.array+3          Begin set at ARRAY(3).
nnnn:1234  55  0       Set byte to 0.
nnnn:1235  55  'abc'   Set 3 bytes to 'a', 'b', 'c'.
nnnn:1238  55  #75     Set byte to decimal 75.
nnnn:1239  55  .       Terminate set command.

#S                Enable segment register display in CPU state
                  display.

#-S               Disable segment register display in CPU state
                  display.
```

3.15 The SR (Search) Command

The SR (SeaRch) command searches a block of memory for a given pattern of numeric or ASCII values, and lists the addresses where the pattern occurs. The form is:

```
SRs,f,pattern {,pattern...}
```

where "s" is the 20-bit starting address of the block to be searched, "f" is the offset of the final address of the block, and "pattern" is a list of one or more hexadecimal values and/or ASCII strings, separated by commas (","). ASCII strings are enclosed in double quotes (") and can be any length. For example,

```
SR200,300,"The form",0D,0A
```

For each occurrence of "pattern", SID-86 displays the 20-bit address of the first byte of the pattern, in the form:

```
ssss:oooo
```

If no addresses are listed, "pattern" was not found.

3.16 The T (Trace) Command

The T command traces program execution for 1 to 0FFFFh program steps, displaying the CPU state before each step. The forms are:

(a) T
(b) Tn
(c) TW
(d) TWn
(e) -T (with forms "a" through "d")

where "n" is the number of program steps to execute before returning control to the console. If "n" is omitted, a single program step is executed.

A program step is generally a single instruction, with the following exceptions:

1) If a BDOS interrupt instruction is traced, the entire BDOS function is treated as one program step, and executes in real-time. This is because SID-86 itself makes BDOS calls, and the BDOS is not re-entrant.

2) If the traced instruction is a MOV or POP whose destination is a segment register, the CPU executes the next instruction immediately. This is due to a feature of the 8086 that disables interrupts (including the Trace interrupt) for one instruction after a MOV or POP loads a segment register. This allows a sequence such as:

```
MOV SS,StackSegment
MOV SP,StackOffset
```

to be executed with no chance of an interrupt occurring between the two instructions, at which time the stack is undefined. A sequence of such MOV or POP instructions, plus one instruction after the sequence, is considered a one program step.

3) If any of the TW forms are used and the traced instruction is a CALL, CALLF, or INT, the entire called subroutine or interrupt handler (and any subroutines called therein) is treated as a one program step and executes in real-time.

Before each program step is executed, SID-86 displays the CPU state, the disassembled instruction to be executed, the symbolic name of the instruction operand (if any), and the contents of the memory location(s) referenced by the instruction (if appropriate). (See Section 3.20, "The X Command", for a detailed description of the CPU state display.) If there is a symbol whose value is equal to the IP, the symbol name followed by a colon (":") is displayed on the line preceding the CPU state display. The segment registers are not normally displayed with the T command, which allows the entire CPU state to be displayed on one line. To enable the segment register display, use the S command (see Section 3.14, "The S Command"). With the segment register display enabled, the display of the CPU state is identical to that of the X command.

In all of the forms, control transfers to the program under test at the address indicated by the CS and IP registers. If "n" is not specified, as in forms (a), one program step is executed. Otherwise, SID-86 executes "n" program steps and displays the CPU state before each step, as in form (b). You can abort a long trace before "n" steps have been executed by typing any character at the console.

When "n" steps have been executed, SID-86 displays the address of the next instruction to be executed, along with the symbolic value of the IP, if there is such a symbol, in the form:

*ssss:oooo .symbol

Forms (c) and (d) are analogous to forms (a) and (b), except in the way subroutine calls are treated. In the TW forms, the entire subroutine called from the program level being traced is treated as a single program step, and executes in real-time. This allows tracing at a high-level of the program, ignoring subroutines that have already been debugged, or for other reasons are not currently of interest.

If the command is preceded by a minus sign ("-"), as in form (e), symbolic labels and symbolic operands are omitted from the CPU state display. This can speed up the display when large symbol tables are loaded, by skipping the symbol table lookup.

When a single instruction is being traced, interrupts are disabled for the duration of the instruction. This prevents SID-86 from tracing through interrupt handlers when debugging on systems in which interrupts occur frequently.

After a T command, the list address used in the L command is set to the address of the next instruction to be executed, and the default segment values are set to the CS and DS register values.

The following are examples of the T command.

#T Trace one program step.

#Tffff Trace 65535 steps.

#-T#500 Trace 500 program steps with symbolic lookup disabled.

3.17 The U (Untrace) Command

The U command is similar to the T command, except that the CPU state is displayed only before the first instruction is executed, rather than before every step. The forms are:

(a) U
(b) Un
(c) UW
(d) UWn
(e) -U (with forms "a" through "d")

where "n" is the number of instructions to execute before returning control to the console. You can abort the U command before "n" steps have been executed by striking any key at the console.

Form (e) differs from the analogous T command in that SID-86 disables the display of intermediate pass points (while the pass count is greater than 1). In this case, only when the pass count reaches 1 is the pass information displayed (see Section 3.11, "The P Command").

The following are examples of the U command.

#U200 Trace without display 200h steps.

#-U200 Trace without display 200h steps, suppressing the intermediate pass point display.

3.18 The V (Value) Command

The V command displays information about the last file loaded with the E or R commands, excluding symbol tables loaded with the E command. The form is:

V

If the last file was loaded with the E command, the V command displays the start and end addresses of each of the segments contained in the file. If the last file was read with the R command, the V command displays the start and end addresses of the block of memory where the file was read. SID-86 responds to the V command with a question mark ("?") if neither the R or E commands have been used.

3.19 The W (Write) Command

The W command writes the contents of a contiguous block of memory to disk. The forms are:

(a) W<filespec>
(b) W<filespec>,s,f

where <filespec> is the filename and filetype of the disk file to receive the data, and "s" and "f" are the 20-bit first and last addresses of the block to be written. If the segment is not specified in "f", SID-86 uses the same value that was used for "s".

With form (a), SID-86 assumes the "s" and "f" values from the last file read with a R command. If no file was read with an R command, SID-86 responds with a question mark ("?"). This form is useful for writing out files after patches have been installed, assuming the overall length of the file is unchanged.

With form (b), where "s" and "f" are specified as 20-bit addresses, the low four bits of "s" are assumed to be 0. Thus, the block being written must always start on a paragraph boundary.

If a file with the name specified in the W command already exists, SID-86 deletes it before writing a new file.

The following are examples of the W command.

#Wtest.cmd Write to the file TEST.CMD the contents of the memory block read into by the most recent R command.

#Wb:test.cmd,40:0,3FF Write the contents of the memory block 40:0 through 40:3FF to the file TEST.CMD on drive B.

3.20 The X (Examine CPU State) Command

The X command allows you to examine and alter the CPU state of the program under test. The forms are:

(a) X
(b) Xr
(c) Xf

where "r" is the name of one of the 8086 CPU registers and "f" is the abbreviation of one of the CPU flags. Form (a) displays the CPU state in the form:

```
        AX   BX   CX  ...  SS   ES   IP
--------- xxxx xxxx xxxx ... xxxx xxxx xxxx
 <instruction>  <symbol name>   <memory value>
```

The nine hyphens at the beginning of the line indicate the state of the nine CPU flags. Each position can be either a hyphen, indicating that the corresponding flag is not set (0), or a 1-character abbreviation of the flag name, indicating that the flag is set (1). The abbreviation of the flag names are shown in Table 3-1.

Table 3-1. Flag Name Abbreviations

Character	Name
O	Overflow
D	Direction
I	Interrupt Enable
T	Trap
S	Sign
Z	Zero
A	Auxiliary Carry
P	Parity
C	Carry

<instruction> is the disassembled instruction at the next location to be executed, which is indicated by the CS and IP registers. If the symbol table contains a symbol whose value is equal to one of the operands in <instruction>, the symbol name is displayed in the <symbol name> field, preceded by a period ("."). If <instruction> references memory, the contents of the referenced location(s) are displayed in the <memory value> field, preceded by an equal sign ("="). Either a byte, word, or double word value is shown, depending on the instruction. In addition to displaying the machine state, the first form changes the values of the default segments back to the CS and DS register values, and the default offset for the L command to the IP register value.

Form (b) allows you to alter the registers in the CPU state of the program being tested. The "r" following the X is the name of one of the 16-bit CPU registers. SID-86 responds by displaying the name of the register, followed by its current value. If you type a Carriage Return, the value of the register is not changed. If you type a valid expression, the contents of the register are changed to the value of the expression. In either case, the next register is then displayed. This process continues until you enter a period (".") or an invalid expression, or the last register is displayed.

Form (c) allows you to alter one of the flags in the CPU state of the program being tested. SID-86 responds by displaying the name of the flag, followed by its current state. If you type a Carriage Return, the state of the flag is not changed. If you type a valid value, the state of the flag is changed to that value. Only one flag can be examined or altered with each Xf command. Set or reset flags by entering a value of 1 or 0.

The following are examples of the X command.

```
#Xbp          Change registers, starting with BP.
BP=1000 2B64    Change BP to hex 2B64.
SI=2000 #12345  Change SI to decimal 12345.
DI=0020 .var+6  Change DI to value of symbol VAR plus 6.
```

CS=0040 . Terminate X command.

3.21 The Z (Display 8087 Math Co-processor Registers) Command

(Not documented... Due to Richard Plinston.)

```
 CW   SW   TW      IP        OP
037F 0000 FFFF 0000 0000 0000 0000

0  0000 0000 0000 0000 0000
1  0000 0000 0000 0000 0000
2  0000 0000 0000 0000 0000
3  0000 0000 0000 0000 0000
4  0000 0000 0000 0000 0000
5  0000 0000 0000 0000 0000
6  0000 0000 0000 0000 0000
7  0000 0000 0000 0000 0000
```

Section 4: Default Segment Values

SID-86 has an internal mechanism that keeps track of the current segment value, making segment specification an optional part of a SID-86 command. SID-86 divides the command set into two types of commands, according to the segment a command defaults to, if you do not specify a segment value in the command line.

The first type of command, pertaining to the Code Segment, includes the A (Assemble), L (List Mnemonics), P (Pass Points), and W (Write) commands. These commands use the internal type-1 segment value if no segment value is specified in the command.

When invoked, SID-86 sets the type-1 segment value to 0, and changes it when one of the following actions is taken:

* When an E command loads a file, SID-86 sets the type-1 segment value to the value of the CS register.

* When an R command reads a file, SID-86 sets the type-1 segment value to the base segment where the file was read.

* When an X command changes the value of the CS register, SID-86 changes the type-1 segment value to the new value of the CS register.

* When SID-86 regains control from a user program after a G, T, or U command, it sets the type-1 segment value to the value of the CS register.

* When an A or L command explicitly specifies a segment value, SID-86 sets the type-1 segment value to the segment value specified.

The second type of command, pertaining to the Data Segment, includes the D (Display), F (Fill), M (Move), and S (Set) commands. These commands use the internal type-2 segment value if no segment value is specified in the command.

When invoked, SID-86 sets the type-2 segment value to 0, and changes it when one of the following actions is taken:

* When an E command loads a file, SID-86 sets the type-2 segment value to the value of the DS register.

* When an R command reads a file, SID-86 sets the type-2 segment value to the base segment where the file was read.

* When an X command changes the value of the DS register, SID-86 changes the type-2 segment value to the new value of the DS register.

* When SID-86 regains control from a user program after a G, T, or U command, it sets the type-2 segment value to the value of the DS register.

* When a D, F, M, or S command explicitly specifies a segment value, SID-86 sets the type-2 segment value to the segment value specified.

When evaluating programs that have identical values in the CS and DS registers, all SID-86 commands defaults to the same segment value, unless explicitly overridden.

Table 4-1 summarizes SID-86's default segment values. Note that the G (Go) command does not fall into either group, because it defaults to the CS register.

Table 4-1. SID-86 Default Segment Values

Command	Type-1	Type-2
A	*	
B		*
D		
E	@	@
F		*
G	@	@
H		
I		
L	*	
M		*
P	%	
Q		
R	*	@
S		*
SR		
T	@	@
U	@	@
V		
W	*	

```
X     @     @
Z
```

Abbreviations Used:
* = Use this segment default if none specified;
change default if specified explicitly.
@ = Change this segment default.
% = Use this segment default if none specified.

Section 5: Assembly Language Syntax for A and L commands

In general, the syntax of the assembly language statements in the A and L commands is standard Intel 8086 assembly language. Several minor exceptions are listed below.

* Up to three prefixes (LOCK, repeat, segment override) can appear in one statement, but they all must precede the opcode of the statement. Alternately, a prefix can be entered on a line by itself.

* The distinction between byte and word string instructions is made as follows:

Byte	Word
LODSB	LODSW
STOSB	STOSW
SCASB	SCASW
MOVSB	MOVSW
CMPSB	CMPSW

* The mnemonics for near and far control transfer instructions are as follows:

Short	Normal	Far
JMPS	JMP	JMPF
	CALL	CALLF
	RET	RETF

* If the operand of a CALLF or JMPF instruction is a 20-bit absolute address, it is entered in the form:

```
ssss:oooo
```

where "ssss" is the segment and "oooo" is the offset of the address.

* Operands that can refer to either a byte or word are ambiguous, and must be preceded either by the prefix BYTE or WORD. These prefixes can be abbreviated to BY and WO. For example:

```
INC BYTE[BP]
NOT WORD[1234]
```

Failure to supply a prefix when needed results in an error message.

* Operands that address memory directly are enclosed in square brackets, to distinguish them from immediate values. For example:

```
ADD AX,5       ; Add 5 to register AX
ADD AX,[5]     ; Add the contents of location 5 to AX
```

* The forms of register indirect memory operands are:

```
[pointer register]
[index register]
[pointer register + index register]
```

where the pointer registers are BX and BP, and the index registers are SI and DI. Any of these forms can be preceded by a numeric offset. For example:

```
ADD BX,[BP+SI]
ADD BX,3[BP+SI]
ADD BX,1D47[BP+SI]
```

Section 6: SID-86 Sample Session

In the following sample session, the user interactively debugs a simple program. User input is shown in Italic print; remarks in square brackets explain the steps involved.

[Type source file of program to test.]

```
A>type typ.a86
; Display the contents of an
; ASCII file at the console.
;-------------------------------
; Base Page locations used.
;
fcb     EQU     005Ch          ; File Control Block
;
; ASCII characters used.
;
eof     EQU     1Ah            ; CP/M End-Of-File character
;
; Interrupts used.
;
bdosi   EQU     224            ; BDOS Interrupt Number
;
; BDOS functions used.
;
ConOutC EQU      2             ; Console Output
Pstring EQU      9             ; Print string
OpenC   EQU     15             ; Open File
ReadC   EQU     20             ; Read File
SetDMAc EQU     26             ; Set DMA Address
```

```
;
;-------------------------------
; Main.
;
start:  MOV     DX,fcb          ; Points to FCB
        CALL    open            ; Open file
loop:   CALL    GetChr          ; Get one char
        CMP     AL,eof          ; Is it the EOF char?
        JZ      done            ; Yes: the end
        CALL    ConOut          ; No: display it
        JMPS    loop            ;     and loop.
;
; The end. Back to CP/M.
;
done:   MOV     DL,0            ; = ?
        MOV     CL,0            ; = Reset function
        JMP     bdos            ; Ditto
;
;-------------------------------
; Get a char from DMA buffer.
;
GetChr: CMP     bPtr,bSize      ; See if we need a new buffer fill
        JC      Get1            ;
        CALL    FilBuf          ; Yes: Refill buffer from file
Get1:   MOV     BX,OFFSET buffer  ;
        MOV     AL,buffer[BX]   ; Get next character from buffer
        INC     bPtr            ; Increment buffer pointer
        RET                     ;
;
;-------------------------------
; Fill DMA buffer from file.
;
FilBuf: MOV     DX,OFFSET buffer  ;
        CALL    SetDMA          ;
        MOV     DX,fcb          ;
        CALL    read            ;
        MOV     bPtr,00h        ;
        RET                     ;
;
;-------------------------------
; Open a file.
;
Open:   MOV     CL,openc        ;
        CALL    bdos            ; Do it the old fashioned way
        CMP     AL,0FFh         ;
        JNZ     err             ;
        RET                     ;
;
;-------------------------------
; Set DMA Address.
;
SetDMA: MOV     CL,SetDMAc      ;
        JMPS    bdos            ;
;
```

```
;-------------------------------
; Read one record from file.
;
Read:   MOV     CL,ReadC        ;
        CALL    bdos            ;
        CMP     AL,00h          ;
        JNZ     err             ;
        RET                     ;
;
;-------------------------------
; Console Output.
;
ConOut: MOV     CL,ConOutC      ;
        JMPS    bdos            ;
;
;-------------------------------
; Print a string.
;
PrintM: MOV     CL,Pstring      ;
        JMPS    bdos            ;
;
;-------------------------------
; Provides Old fashioned BDOS call.
;
bdos:   INT     bdosi           ; IBM PC
        RET                     ;
;
;-------------------------------
; Print error message and abort.
;
Err:    MOV     DX,OFFSET ErrorM  ;
        CALL    printm          ;
        JMP     done            ;
;
;-------------------------------
        DSEG
;-------------------------------
        ORG     100h            ; Reserve Base Page
;
ErrorM  DB      'ERROR',0Dh,0Ah,'$'
bSize   EQU     80h             ; Buffer Size = 128 characters
buffer  RS      bSize           ; Reserve them
bPtr    DB      bSize           ; Buffer Pointer = 1st char

;
;-------------------------------
;
        END
```

[Assemble TYP.A86 file.]

```
A>asm typ

CP/M 8086 ASSEMBLER VER 1.1
```

```
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY.  NUMBER OF ERRORS:   0.  USE FACTOR:  0%

[Type list file created by ASM-86.]

A>type typ.lst

CP/M ASM86 1.1  SOURCE: TYP.A86                                    PAGE   1


                ; Display the contents of an
                ; ASCII file at the console.
                ;-------------------------------
                ; Base Page locations used.
                ;
  005C            fcb     EQU     005Ch          ; File Control Block
                ;
                ; ASCII characters used.
                ;
  001A            eof     EQU     1Ah            ; CP/M End-Of-File character
                ;
                ; Interrupts used.
                ;
  00E0            bdosi   EQU     224            ; BDOS Interrupt Number
                ;
                ; BDOS functions used.
                ;
  0002            ConOutC EQU     2              ; Console Output
  0009            Pstring EQU     9             ; Print string
  000F            OpenC   EQU     15             ; Open File
  0014            ReadC   EQU     20            ; Read File
  001A            SetDMAc EQU     26             ; Set DMA Address
                ;
                ;-------------------------------
                ; Main.
                ;
0000 BA5C00           start:  MOV     DX,fcb         ; Points to FCB
0003 E83B00       0041        CALL    open           ; Open file
0006 E81000       0019 loop:  CALL    GetChr         ; Get one char
0009 3C1A                     CMP     AL,eof         ; Is it the EOF char?
000B 7405         0012        JZ      done           ; Yes: the end
000D E84900       0059        CALL    ConOut         ; No: display it
0010 EBF4         0006        JMPS    loop           ;     and loop.
                ;
                ; The end. Back to CP/M.
                ;
0012 B200             done:   MOV     DL,0           ; = ?
0014 B100                     MOV     CL,0           ; = Reset function
0016 E94800       0061        JMP     bdos           ; Ditto
                ;
                ;-------------------------------
                ; Get a char from DMA buffer.
                ;
```

```
0019 803E880180       GetChr: CMP     bPtr,bSize      ; See if we need a new buffer fill

001E 7203       0023        JC      Get1            ;
0020 E80C00      002F        CALL    FilBuf          ; Yes: Refill buffer from file
0023 BB0801           Get1:   MOV     BX,OFFSET buffer  ;
0026 8A870801                 MOV     AL,buffer[BX]   ; Get next character from buffer
002A FE068801                 INC     bPtr            ; Increment buffer pointer
002E C3                   RET                 ;
                ;
                ;-------------------------------
                ; Fill DMA buffer from file.

CP/M ASM86 1.1  SOURCE: TYP.A86                                                 PAGE  2


                ;
002F BA0801           FilBuf: MOV     DX,OFFSET buffer  ;
0032 E81600      004B        CALL    SetDMA          ;
0035 BA5C00                 MOV     DX,fcb          ;
0038 E81400      004F        CALL    read            ;
003B C606880100             MOV     bPtr,00h        ;
0040 C3                   RET                 ;
                ;
                ;-------------------------------
                ; Open a file.
                ;
0041 B10F             Open:   MOV     CL,openc        ;
0043 E81B00      0061        CALL    bdos            ; Do it the old fashioned way
0046 3CFF                 CMP     AL,0FFh         ;
0048 751A       0064        JNZ     err             ;
004A C3                   RET                 ;
                ;
                ;-------------------------------
                ; Set DMA Address.
                ;
004B B11A             SetDMA: MOV     CL,SetDMAc      ;
004D EB12       0061        JMPS    bdos            ;
                ;
                ;-------------------------------
                ; Read one record from file.
                ;
004F B114             Read:   MOV     CL,ReadC        ;
0051 E80D00      0061        CALL    bdos            ;
0054 3C00                 CMP     AL,00h          ;
0056 750C       0064        JNZ     err             ;
0058 C3                   RET                 ;
                ;
                ;-------------------------------
                ; Console Output.
                ;
0059 B102             ConOut: MOV     CL,ConOutC      ;
005B EB04       0061        JMPS    bdos            ;
                ;
                ;-------------------------------
```

```
                ; Print a string.
                ;
005D B109           PrintM: MOV     CL,Pstring      ;
005F EB00       0061        JMPS    bdos            ;
                ;
                ;--------------------------------
                ; Provides Old fashioned BDOS call.
                ;
0061 CDE0           bdos:   INT     bdosi           ; IBM PC
0063 C3                 RET                     ;
                ;
                ;--------------------------------
                ; Print error message and abort.

CP/M ASM86 1.1  SOURCE: TYP.A86                                         PAGE   3


                ;
0064 BA0001         Err:    MOV     DX,OFFSET ErrorM  ;
0067 E8F3FF     005D        CALL    printm          ;
006A E9A5FF     0012        JMP     done            ;
                ;

                ;--------------------------------
                        DSEG
                ;--------------------------------
                        ORG     100h            ; Reserve Base Page
                ;
0100 4552524F520D       ErrorM  DB      'ERROR',0Dh,0Ah,'$'
    0A24
 0080               bSize   EQU     80h             ; Buffer Size = 128 characters
0108                buffer  RS      bSize           ; Reserve them
0188 80              bPtr   DB      bSize           ; Buffer Pointer = 1st char
                ;
                ;--------------------------------
                ;
                        END


END OF ASSEMBLY.  NUMBER OF ERRORS:   0.  USE FACTOR:  0%

[Type hex file produced by ASM-86.]

A>type typ.h86
:0400000300000000F9
:1B000081BA5C00E83B00E810003C1A7405E84900EBF4B200B100E94800803E02
:1B001B818801807203E80C00BB08018A870801FE068801C3BA0801E81600BA2E
:1B0036815C00E81400C606880100C3B10FE81B003CFF751AC3B11AEB12B114E1
:1B005181E80D003C00750CC3B102EB04B109EB00CDE0C3BA0001E8F3FFE9A5C4
:01006C81FF13
:080100824552524F520D0A24B0
:010188828074
:00000001FF
```

[Type symbol table file produced by ASM-86.]

A>type typ.sym

0000 VARIABLES
0188 BPTR 0108 BUFFER 0100 ERRORM

0000 NUMBERS
00E0 BDOSI 0080 BSIZE 0002 CONOUTC 001A EOF 005C FCB
000F OPENC 0014 READC 001A SETDMAC

0000 LABELS
0061 BDOS 0059 CONOUT 0012 DONE 0064 ERR 002F FILBUF
0023 GET1 0019 GETCHR 0006 LOOP 0041 OPEN 005D PRINTM
004F READ 004B SETDMA 0000 START

[Create command file from hex file.]

A>gencmd typ

BYTES READ 007A
RECORDS WRITTEN 05

[Try executing the program with the file TYP.A86 as data.]

A>typ typ.a86
ERROR

[The program did not work correctly, so load the command file and symbol table file to find out why.]

A>sid typ.cmd typ.sym
SID86 1.01 1/4/83

[SID-86 shows the start and end addresses of each segment from the file.]

  START END
CS 1CC1:0000 1CC1:006F
DS 1CC8:0000 1CC8:018F
SYMBOLS

[Display all the symbols that SID-86 loaded.]

#h
0000 VARIABLES
0188 BPTR
0108 BUFFER
0100 ERRORM
0000 NUMBERS
00E0 BDOSI
0080 BSIZE
0002 CONOUTC
001A EOF
005C FCB

```
000F OPENC
0014 READC
001A SETDMAC
0000 LABELS
0061 BDOS
0059 CONOUT
0012 DONE
0064 ERR
002F FILBUF
0023 GET1
0019 GETCHR
0006 LOOP
0041 OPEN
005D PRINTM
004F READ
004B SETDMA
0000 START
```

[Disassemble the beginning of the code segment.]

```
#l
START:
  1CC1:0000 MOV    DX,005C .FCB
  1CC1:0003 CALL   0041 .OPEN
LOOP:
  1CC1:0006 CALL   0019 .GETCHR
  1CC1:0009 CMP    AL,1A .EOF
  1CC1:000B JZ     0012 .DONE
  1CC1:000D CALL   0059 .CONOUT
  1CC1:0010 JMPS   0006 .LOOP
DONE:
  1CC1:0012 MOV    DL,00 .VARIABLES
  1CC1:0014 MOV    CL,00 .VARIABLES
  1CC1:0016 JMP    0061 .BDOS
GETCHR:
  1CC1:0019 CMP    BYTE [0188],80 .BPTR
  1CC1:001E JB     0023 .GET1
```

[Set up the default file control block at 005Ch with the name of the file to process.]

```
#ityp.a86
```

[Trace the first two instructions of the program.]

```
#t2
        AX   BX   CX   DX   SP   BP   SI   DI   IP
--I------ 0000 0000 0000 0000 015C 0000 0000 0000 0000 MOV    DX,005C .FCB
--I------ 0000 0000 0000 005C 015C 0000 0000 0000 0003 CALL   0041 .OPEN
```

[SID-86 stops execution after two instructions, at the label OPEN.]

```
*1CC1:0041 .OPEN
```

[Display the contents of the default FCB, to make sure it is set up right.]

```
#d.fcb,+#35
1CC8:005C 00 54 59 50 20 20 20 20 20 41 38 36 00 00 00 00 .TYP     A86....
1CC8:006C 00 20 20 20 20 20 20 20 20 20 20 20 00 00 00 00 .           ....
1CC8:007C 00 00 00 00 ....
```

[The FCB looks Ok. Disassemble the next few instructions.]

```
#l
OPEN:
  1CC1:0041 MOV    CL,0F .OPENC
  1CC1:0043 CALL   0061 .BDOS
  1CC1:0046 CMP    AL,FF
  1CC1:0048 JNZ    0064 .ERR
  1CC1:004A RET
SETDMA:
  1CC1:004B MOV    CL,1A .EOF
  1CC1:004D JMPS   0061 .BDOS
READ:
  1CC1:004F MOV    CL,14 .READC
  1CC1:0051 CALL   0061 .BDOS
  1CC1:0054 CMP    AL,00 .VARIABLES
  1CC1:0056 JNZ    0064 .ERR
  1CC1:0058 RET
```

[Continue program execution with a break point after the open function.]

```
#g,46
*1CC1:0046
```

[Display the CPU registers.]

```
#x
        AX   BX   CX   DX   SP   BP   SI   DI   CS   DS   SS   ES   IP
--I------ 0000 0000 0000 0000 015A 0000 0000 0000 1CC1 1CC8 0AFB 1CC8 0046
CMP    AL,FF
```

[Registers look Ok; trace a few more instructions.]

```
#t2
        AX   BX   CX   DX   SP   BP   SI   DI   IP
--I------ 0000 0000 0000 0000 015A 0000 0000 0000 0046 CMP    AL,FF
--I---A-C 0000 0000 0000 0000 015A 0000 0000 0000 0048 JNZ    0064 .ERR
*1CC1:0064 .ERR
```

[Shouldn't be getting to the ERR label -- the jump instruction seems to be of the wrong flavor. It should be a JZ. Rather than editing the source, let's install a patch. Since the new instruction is the same length as the old one, it will be easy. Read the file into memory (including the header record).]

```
#rtyp.cmd
  START      END
1CEF:0000 1CEF:027F
```

[The file was read into memory starting at paragraph 1CEF. That is where the Header Record is -- the code will start 8 paragraphs later.]

```
#h1CEF,8
+ 1CF7   - 1CE7   * 0000E778   / 039D (0007)
```

[The code starts in paragraph 1CF7. If we use that as the base for the L command, all the symbol values will be correct.]

```
#l1CF7:0
START:
  1CF7:0000 MOV    DX,005C .FCB
  1CF7:0003 CALL   0041 .OPEN
LOOP:
  1CF7:0006 CALL   0019 .GETCHR
  1CF7:0009 CMP    AL,1A .EOF
  1CF7:000B JZ     0012 .DONE
  1CF7:000D CALL   0059 .CONOUT
  1CF7:0010 JMPS   0006 .LOOP
DONE:
  1CF7:0012 MOV    DL,00 .VARIABLES
  1CF7:0014 MOV    CL,00 .VARIABLES
  1CF7:0016 JMP    0061 .BDOS
GETCHR:
  1CF7:0019 CMP    BYTE [0188],80 .BPTR
  1CF7:001E JB     0023 .GET1
```

[Disassemble the OPEN routine.]

```
#l.open,.setdma-1
OPEN:
  1CF7:0041 MOV    CL,0F .OPENC
  1CF7:0043 CALL   0061 .BDOS
  1CF7:0046 CMP    AL,FF
  1CF7:0048 JNZ    0064 .ERR
  1CF7:004A RET
```

[Assemble patch instruction.]

```
#a48
```

[Note that symbolic values may be used in the A command.]

```
1CF7:0048 jz .err
1CF7:004A .
```

[Write the patched file back to disk. The start and end addresses need not be included in the W command, since the overall length of the file did not change.]

```
#wtyp.cmd
```

[Reload the patched file and symbols. This is needed, since the R command

doesn't set up registers.]

```
#etyp.cmd typ.sym
    START     END
CS 1CC1:0000 1CC1:006F
DS 1CC8:0000 1CC8:018F
SYMBOLS
#ityp.a86
```

[Execute the program with a break point at DONE.]

```
#g,.done
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffff
```

[This doesn't look quite right. Invoke SID-86 again, leaving off the file types, since SID-86 uses the appropriate defaults.]

```
A>sid typ typ
SID86 1.01   1/4/83
    START     END
CS 1CC1:0000 1CC1:006F
DS 1CC8:0000 1CC8:018F
SYMBOLS
```

[Set up default file control block.]

```
#ityp.a86
```

[Disassemble start of code segment.]

```
#l
START:
  1CC1:0000 MOV    DX,005C .FCB
  1CC1:0003 CALL   0041 .OPEN
LOOP:
  1CC1:0006 CALL   0019 .GETCHR
  1CC1:0009 CMP    AL,1A .EOF
  1CC1:000B JZ     0012 .DONE
  1CC1:000D CALL   0059 .CONOUT
  1CC1:0010 JMPS   0006 .LOOP
DONE:
  1CC1:0012 MOV    DL,00 .VARIABLES
  1CC1:0014 MOV    CL,00 .VARIABLES
  1CC1:0016 JMP    0061 .BDOS
GETCHR:
  1CC1:0019 CMP    BYTE [0188],80 .BPTR
  1CC1:001E JB     0023 .GET1
```

[Trace without call, so SID-86 doesn't trace the OPEN routine, which should be fixed.]

```
#tw2
```

```
        AX   BX   CX   DX   SP   BP   SI   DI   IP
--I------ 0000 0000 0000 0000 015C 0000 0000 0000 0000 MOV    DX,005C .FCB
--I------ 0000 0000 0000 005C 015C 0000 0000 0000 0003 CALL   0041 .OPEN
*1CC1:0006 .LOOP
```

[Disassemble next few instructions.]

```
#l
LOOP:
  1CC1:0006 CALL   0019 .GETCHR
  1CC1:0009 CMP    AL,1A .EOF
  1CC1:000B JZ     0012 .DONE
  1CC1:000D CALL   0059 .CONOUT
  1CC1:0010 JMPS   0006 .LOOP
DONE:
  1CC1:0012 MOV    DL,00 .VARIABLES
  1CC1:0014 MOV    CL,00 .VARIABLES
  1CC1:0016 JMP    0061 .BDOS
GETCHR:
  1CC1:0019 CMP    BYTE [0188],80 .BPTR
  1CC1:001E JB     0023 .GET1
  1CC1:0020 CALL   002F .FILBUF
GET1:
  1CC1:0023 MOV    BX,0108 .BUFFER
```

[Trace without call next three instructions, to see if this sequence is working.]

```
#tw3
        AX   BX   CX   DX   SP   BP   SI   DI   IP
--I---A-C 0000 0000 0000 0000 015C 0000 0000 0000 0006 CALL   0019 .GETCHR
--I------ 0059 0108 0000 0000 015C 0000 0000 0000 0009 CMP    AL,1A .EOF
--I---AP- 0059 0108 0000 0000 015C 0000 0000 0000 000B JZ     0012 .DONE
*1CC1:000D
```

[GETCHR is returning a 59h -- something must be wrong there. Use the E command to bring in a fresh copy of the program.]

```
#etyp typ
    START      END
CS 1CC1:0000 1CC1:006F
DS 1CC8:0000 1CC8:018F
SYMBOLS
```

[Disassemble code segment.]

```
#l
START:
  1CC1:0000 MOV    DX,005C .FCB
  1CC1:0003 CALL   0041 .OPEN
LOOP:
  1CC1:0006 CALL   0019 .GETCHR
  1CC1:0009 CMP    AL,1A .EOF
  1CC1:000B JZ     0012 .DONE
```

```
 1CC1:000D CALL   0059 .CONOUT
 1CC1:0010 JMPS   0006 .LOOP
DONE:
 1CC1:0012 MOV    DL,00 .VARIABLES
 1CC1:0014 MOV    CL,00 .VARIABLES
 1CC1:0016 JMP    0061 .BDOS
GETCHR:
 1CC1:0019 CMP    BYTE [0188],80 .BPTR
 1CC1:001E JB     0023 .GET1
```

[Trace without call, since we know the open works.]

```
#tw2
        AX   BX   CX   DX   SP   BP   SI   DI   IP
--I---AP- 0059 0108 0000 0000 015C 0000 0000 0000 0000 MOV    DX,005C .FCB
--I---AP- 0059 0108 0000 005C 015C 0000 0000 0000 0003 CALL   0041 .OPENERROR
```

[Oops! Forgot to set up the file control block... Try again.]

```
A>sid typ typ
SID86 1.01   1/4/83
    START      END
CS 1CC1:0000 1CC1:006F
DS 1CC8:0000 1CC8:018F
SYMBOLS
```

[This should work better.]

```
#ityp.a86
#g,.getchr
*1CC1:0019 .GETCHR
```

[Disassemble GETCHR routine.]

```
#l
GETCHR:
 1CC1:0019 CMP    BYTE [0188],80 .BPTR
 1CC1:001E JB     0023 .GET1
 1CC1:0020 CALL   002F .FILBUF
GET1:
 1CC1:0023 MOV    BX,0108 .BUFFER
 1CC1:0026 MOV    AL,0108[BX] .BUFFER
 1CC1:002A INC    BYTE [0188] .BPTR
 1CC1:002E RET
FILBUF:
 1CC1:002F MOV    DX,0108 .BUFFER
 1CC1:0032 CALL   004B .SETDMA
 1CC1:0035 MOV    DX,005C .FCB
 1CC1:0038 CALL   004F .READ
 1CC1:003B MOV    BYTE [0188],00 .BPTR
```

[Trace first few instructions. Note that SID-86 shows the contents of BPTR as it is being compared.]

```
#t2
       AX   BX   CX   DX   SP   BP   SI   DI   IP
--I---A-C 0000 0000 0000 0000 015A 0000 0000 0000 0019 CMP    BYTE [0188],80 .BPTR =80
--I--Z-P- 0000 0000 0000 0000 015A 0000 0000 0000 001E JB     0023 .GET1
*1CC1:0020
```

[The compare worked Ok; keep going.]

```
#t
       AX   BX   CX   DX   SP   BP   SI   DI   IP
--I--Z-P- 0000 0000 0000 0000 015A 0000 0000 0000 0020 CALL   002F .FILBUF
*1CC1:002F .FILBUF
```

[See what FILBUF looks like.]

```
#l
FILBUF:
  1CC1:002F MOV    DX,0108 .BUFFER
  1CC1:0032 CALL   004B .SETDMA
  1CC1:0035 MOV    DX,005C .FCB
  1CC1:0038 CALL   004F .READ
  1CC1:003B MOV    BYTE [0188],00 .BPTR
  1CC1:0040 RET
OPEN:
  1CC1:0041 MOV    CL,0F .OPENC
  1CC1:0043 CALL   0061 .BDOS
  1CC1:0046 CMP    AL,FF
  1CC1:0048 JZ     0064 .ERR
  1CC1:004A RET
SETDMA:
  1CC1:004B MOV    CL,1A .EOF
```

[See what is in the buffer before the disk read takes place.]

```
#d.buffer,+2F
1CC8:0108 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CC8:0118 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1CC8:0128 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

[Trace the FILBUF routine.]

```
#tw6
       AX   BX   CX   DX   SP   BP   SI   DI   IP
--I--Z-P- 0000 0000 0000 0000 0158 0000 0000 0000 002F MOV    DX,0108 .BUFFER
--I--Z-P- 0000 0000 0000 0108 0158 0000 0000 0000 0032 CALL   004B .SETDMA
--I--Z-P- 00C3 00C3 0000 0108 0158 0000 0000 0000 0035 MOV    DX,005C .FCB
--I--Z-P- 00C3 00C3 0000 005C 0158 0000 0000 0000 0038 CALL   004F .READ
--I--Z-P- 0000 0000 0000 0000 0158 0000 0000 0000 003B MOV    BYTE [0188],00 .BPTR =80
--I--Z-P- 0000 0000 0000 0000 0158 0000 0000 0000 0040 RET
*1CC1:0023 .GET1
```

[See what is in the buffer after the read.]

```
#d.buffer,+4F
```

```
1CC8:0108 3B 20 44 69 73 70 6C 61 79 20 74 68 65 20 63 6F ; Display the co
1CC8:0118 6E 74 65 6E 74 73 20 6F 66 20 61 6E 0D 0A 3B 20 ntents of an..;
1CC8:0128 41 53 43 49 49 20 66 69 6C 65 20 61 74 20 74 68 ASCII file at th
1CC8:0138 65 20 63 6F 6E 73 6F 6C 65 2E 0D 0A 3B 2D 2D 2D e console...;---
1CC8:0148 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ----------------
```

[Looks like good data in the buffer. See what's next.]

```
#l
GET1:
  1CC1:0023 MOV    BX,0108 .BUFFER
  1CC1:0026 MOV    AL,0108[BX] .BUFFER
  1CC1:002A INC    BYTE [0188] .BPTR
  1CC1:002E RET
FILBUF:
  1CC1:002F MOV    DX,0108 .BUFFER
  1CC1:0032 CALL   004B .SETDMA
  1CC1:0035 MOV    DX,005C .FCB
  1CC1:0038 CALL   004F .READ
  1CC1:003B MOV    BYTE [0188],00 .BPTR
  1CC1:0040 RET
OPEN:
  1CC1:0041 MOV    CL,0F .OPENC
  1CC1:0043 CALL   0061 .BDOS
```

[Trace the code getting the next character from the buffer.]

```
#t4
        AX   BX   CX   DX   SP   BP   SI   DI   IP
--I--Z-P- 0000 0000 0000 0000 015A 0000 0000 0000 0023 MOV    BX,0108 .BUFFER
--I--Z-P- 0000 0108 0000 0000 015A 0000 0000 0000 0026 MOV    AL,0108[BX] .BUFFER =59
--I--Z-P- 0059 0108 0000 0000 015A 0000 0000 0000 002A INC    BYTE [0188] .BPTR =00
--I------ 0059 0108 0000 0000 015A 0000 0000 0000 002E RET
*1CC1:0009
```

[It is getting the wrong data, because BX should have the contents of BPTR in it, rather than the address of the buffer. Need to edit, and also install fix that we patched earlier.]

```
#^C
A>ed typ.a86
    : *#afOpen:
  65: *
  66:         CALL    bdos          ; Do it the old fashioned way
  66: *
  67:         CMP     AL,0FFh       ;
  67: *
  68:         JNZ     err           ;
  68: *sJNZ^ZJZ^Z0lt
  68:         JZ      err           ;
  68: *bfGet1:
  46: *0lt
  46:  Get1:   MOV     BX,OFFSET buffer ;
  46: *sOFFSET buffer^ZbPtr^Z0lt
```

```
  46:  Get1:   MOV     BX,bPtr         ;
  46: *sBX^ZBL^Z0lt
  46:  Get1:   MOV     BL,bPtr         ;
  46: *
  47:          MOV     AL,buffer[BX]   ; Get next character from buffer
  47: *i
  47:        MOV     BH,00h          ;
  48: ^Z
  48: *e

A>asm typ

CP/M 8086 ASSEMBLER VER 1.1
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY.  NUMBER OF ERRORS:   0.  USE FACTOR:  0%

A>gencmd typ

BYTES READ    007D
RECORDS WRITTEN 05
```

[Try it again.]

```
A>typ typ.a86
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffff
```

[Still no good. I thought this was supposed to be a simple project!]

```
A>sid typ typ
SID86 1.01   1/4/83
    START      END
CS 1CC1:0000 1CC1:006F
DS 1CC8:0000 1CC8:018F
SYMBOLS
#ityp.a86
#l
START:
  1CC1:0000 MOV    DX,005C .FCB
  1CC1:0003 CALL   0044 .OPEN
LOOP:
  1CC1:0006 CALL   0019 .GETCHR
  1CC1:0009 CMP    AL,1A .EOF
  1CC1:000B JZ     0012 .DONE
  1CC1:000D CALL   005C .CONOUT
  1CC1:0010 JMPS   0006 .LOOP
DONE:
  1CC1:0012 MOV    DL,00 .VARIABLES
  1CC1:0014 MOV    CL,00 .VARIABLES
  1CC1:0016 JMP    0064 .BDOS
GETCHR:
  1CC1:0019 CMP    BYTE [0188],80 .BPTR
```

```
 1CC1:001E JB     0023 .GET1
```

[Trace at the top level.]

```
#tw5
        AX   BX   CX   DX   SP   BP   SI   DI   IP
--I------ 0000 0000 0000 0000 015C 0000 0000 0000 0000 MOV    DX,005C .FCB
--I------ 0000 0000 0000 005C 015C 0000 0000 0000 0003 CALL   0044 .OPEN
LOOP:
--I---A-C 0000 0000 0000 0000 015C 0000 0000 0000 0006 CALL   0019 .GETCHR
--I------ 003B 0000 0000 0000 015C 0000 0000 0000 0009 CMP    AL,1A .EOF
--I----P- 003B 0000 0000 0000 015C 0000 0000 0000 000B JZ     0012 .DONE
*1CC1:000D
```

[Got the right data from GETCHR.]

```
#l
 1CC1:000D CALL   005C .CONOUT
 1CC1:0010 JMPS   0006 .LOOP
DONE:
 1CC1:0012 MOV    DL,00 .VARIABLES
 1CC1:0014 MOV    CL,00 .VARIABLES
 1CC1:0016 JMP    0064 .BDOS
GETCHR:
 1CC1:0019 CMP    BYTE [0188],80 .BPTR
 1CC1:001E JB     0023 .GET1
 1CC1:0020 CALL   0032 .FILBUF
GET1:
 1CC1:0023 MOV    BL,[0188] .BPTR
 1CC1:0027 MOV    BH,00 .VARIABLES
 1CC1:0029 MOV    AL,0108[BX] .BUFFER
 1CC1:002D INC    BYTE [0188] .BPTR
#t
        AX   BX   CX   DX   SP   BP   SI   DI   IP
--I----P- 003B 0000 0000 0000 015C 0000 0000 0000 000D CALL   005C .CONOUT
*1CC1:005C .CONOUT
#l
CONOUT:
 1CC1:005C MOV    CL,02 .CONOUTC
 1CC1:005E JMPS   0064 .BDOS
PRINTM:
 1CC1:0060 MOV    CL,09
 1CC1:0062 JMPS   0064 .BDOS
BDOS:
 1CC1:0064 INT    E0 .BDOSI
 1CC1:0066 RET
ERR:
 1CC1:0067 MOV    DX,0100 .ERRORM
 1CC1:006A CALL   0060 .PRINTM
 1CC1:006D JMP    0012 .DONE
 1CC1:0070 ??=    6F
 1CC1:0071 ADD    [BX+SI],AL
 1CC1:0073 ??=    C1
```

[At this point, the data to output to the console should be in DL, but it is not... Another edit.]

```
#^C
A>ed typ.a86
    : *#afConOut
  18: *0lt
  18:  ConOutC EQU      2             ; Console Output
  18: *1fConOut
  31: *0lt
  31:         CALL    ConOut         ; No: display it
  31: *i
  31:        MOV     DL,AL          ; Prepare for ConOut
  32: ^Z
  32: *e

A>asm typ

CP/M 8086 ASSEMBLER VER 1.1
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY.  NUMBER OF ERRORS:   0.  USE FACTOR:  0%

A>gencmd typ

BYTES READ    007F
RECORDS WRITTEN 06
```

[One more time!]

```
A>typ typ.a86
; Display the contents of an
; ASCII file at the console.
;-------------------------------
; Base Page locations used.
;
fcb     EQU     005Ch          ; File Control Block
;
; ASCII characters used.
;
eof     EQU     1Ah            ; CP/M End-Of-File character
;
; Interrupts used.
;
bdosi   EQU     224            ; BDOS Interrupt Number
;
; BDOS functions used.
;
ConOutC EQU      2             ; Console Output
Pstring EQU      9             ; Print String
OpenC   EQU     15             ; Open File
ReadC   EQU     20             ; Read File
SetDMAc EQU     26             ; Set DMA Address
;
```

```
;-------------------------------
; Main.
;
start:  MOV     DX,fcb          ; Points to FCB
        CALL    open            ; Open file
loop:   CALL    GetChr          ; Get one char
        CMP     AL,eof          ; Is it the EOF char?
        JZ      done            ; Yes: the end
        MOV     DL,AL           ; Prepare for ConOut
        CALL    ConOut          ; No: display it
        JMPS    loop            ;     and loop.
;
; The end. Back to CP/M.
;
done:   MOV     DL,0            ; = ?
        MOV     CL,0            ; = Reset function
        JMP     bdos            ; Ditto
;
;-------------------------------
; Get a char from DMA buffer.
;
GetChr: CMP     bPtr,bSize      ; See if we need a new buffer fill
        JC      Get1            ;
        CALL    FilBuf          ; Yes: Refill buffer from file
Get1:   MOV     BL,bPtr         ;
        MOV     BH,00h          ;
        MOV     AL,buffer[BX]   ; Get next character from buffer
        INC     bPtr            ; Increment buffer pointer
        RET                     ;
;
;-------------------------------
; Fill DMA buffer from file.
;
FilBuf: MOV     DX,OFFSET buffer  ;
        CALL    SetDMA          ;
        MOV     DX,fcb          ;
        CALL    read            ;
        MOV     bPtr,0          ;
        RET                     ;
;
;-------------------------------
; Open a file.
;
Open:   MOV     CL,openc        ;
        CALL    bdos            ; Do it the old fashioned way
        CMP     AL,0FFh         ;
        JZ      err             ;
        RET                     ;
;
;-------------------------------
; Set DMA Address.
;
SetDMA: MOV     CL,SetDMAc      ;
        JMPS    bdos            ;
```

```
;
;-------------------------------
; Read one record from file.
;
Read:   MOV     CL,ReadC        ;
        CALL    bdos            ;
        CMP     AL,00h          ;
        JNZ     err             ;
        RET                     ;
;
;-------------------------------
; Console Output.
;
ConOut: MOV     CL,ConOutC      ;
        JMPS    bdos            ;
;
;-------------------------------
; Print a string.
;
PrintM: MOV     CL,Pstring      ;
        JMPS    bdos            ;
;
;-------------------------------
; Provides Old fashioned BDOS call.
;
bdos:   INT     bdosi           ; IBM PC
        RET                     ;
;
;-------------------------------
; Print error message and abort.
;
Err:    MOV     DX,OFFSET ErrorM  ;
        CALL    printm          ;
        JMP     done            ;
;
;-------------------------------
        DSEG
;-------------------------------
        ORG     100h            ; Reserve Base Page
;
ErrorM  DB      'ERROR',0Dh,0Ah,'$'
bSize   EQU     80h             ; Buffer Size = 128 characters
buffer  RS      bSize           ; Reserve them
bPtr    DB      bSize           ; Buffer Pointer = 1st char
;
;-------------------------------
;
        END
```

[Done at last. Just out of curiosity, let's find out how many characters are in the file by setting a pass point with a high count at CONOUT.]

```
A>sid typ typ
SID86 1.01   1/4/83
```

```
    START     END
CS 1CC1:0000 1CC1:007F
DS 1CC9:0000 1CC9:018F
SYMBOLS
#ityp.a86
```

[Hopefully, there are fewer than 65535 characters!]

```
#p.conout,ffff
```

[Begin execution, with a break point at DONE.]

```
#g,.done
FFFF PASS 1CC1:005E .CONOUT
--I----P- 003B 0000 0000 003B 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTC;
FFFE PASS 1CC1:005E .CONOUT
--I---AP- 0220 0001 0000 1820 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTC
FFFD PASS 1CC1:005E .CONOUT
--I---A-- 0244 0002 0000 1844 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTCD
FFFC PASS 1CC1:005E .CONOUT
--I---A-- 0269 0003 0000 1869 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTCi
FFFB PASS 1CC1:005E .CONOUT
--I---AP- 0273 0004 0000 1873 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTCs
FFFA PASS 1CC1:005E .CONOUT
--I---AP- 0270 0005 0000 1870 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTCp
FFF9 PASS 1CC1:005E .CONOUT
--I------ 026C 0006 0000 186C 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTCl
FFF8 PASS 1CC1:005E .CONOUT
--I---AP- 0261 0007 0000 1861 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTCa
FFF7 PASS 1CC1:005E .CONOUT
--I---AP- 0279 0008 0000 1879 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTCy
FFF6 PASS 1CC1:005E .CONOUT
--I---AP- 0220 0009 0000 1820 015A 0000 0000 0000 005E MOV    CL,02 .CONOUTC
FFF5 PASS 1CC1:005E .CONOUT
```

[This is too messy. Using the -G command will suppress the display of the CPU registers until the pass count is 1.]

```
#-g,.done
the contents of an
; ASCII file at the console.
;-------------------------------
; Base Page locations used.
;
fcb     EQU     005Ch           ; File Control Block
;
; ASCII characters used.
;
eof     EQU     1Ah             ; CP/M End-Of-File character
;
; Interrupts used.
;
bdosi   EQU     224             ; BDOS Interrupt Number
;
```

```
; BDOS functions used.
;
ConOutC EQU     2           ; Console Output
Pstring EQU     9           ; Print String
OpenC   EQU     15          ; Open File
ReadC   EQU     20          ; Read File
SetDMAc EQU     26          ; Set DMA Address
;
;-------------------------------
; Main.
;
start:  MOV     DX,fcb      ; Points to FCB
        CALL    open        ; Open file
loop:   CALL    GetChr      ; Get one char
        CMP     AL,eof      ; Is it the EOF char?
        JZ      done        ; Yes: the end
        MOV     DL,AL       ; Prepare for ConOut
        CALL    ConOut      ; No: display it
        JMPS    loop        ;     and loop.
;
; The end. Back to CP/M.
;
done:   MOV     DL,0        ; = ?
        MOV     CL,0        ; = Reset function
        JMP     bdos        ; Ditto
;
;-------------------------------
; Get a char from DMA buffer.
;
GetChr: CMP     bPtr,bSize  ; See if we need a new buffer fill
        JC      Get1        ;
        CALL    FilBuf      ; Yes: Refill buffer from file
Get1:   MOV     BL,bPtr     ;
        MOV     BH,00h      ;
        MOV     AL,buffer[BX]   ; Get next character from buffer
        INC     bPtr        ; Increment buffer pointer
        RET                 ;
;
;-------------------------------
; Fill DMA buffer from file.
;
FilBuf: MOV     DX,OFFSET buffer  ;
        CALL    SetDMA      ;
        MOV     DX,fcb      ;
        CALL    read        ;
        MOV     bPtr,00h    ;
        RET                 ;
;
;-------------------------------
; Open a file.
;
Open:   MOV     CL,openc    ;
        CALL    bdos        ; Do it the old fashioned way
        CMP     AL,0FFh     ;
```

```
        JZ      err             ;
        RET                     ;
;
;-------------------------------
; Set DMA Address.
;
SetDMA: MOV     CL,SetDMAc      ;
        JMPS    bdos            ;
;
;-------------------------------
; Read one record from file.
;
Read:   MOV     CL,ReadC        ;
        CALL    bdos            ;
        CMP     AL,0            ;
        JNZ     err             ;
        RET                     ;
;
;-------------------------------
; Console Output.
;
ConOut: MOV     CL,ConOutC      ;
        JMPS    bdos            ;
;
;-------------------------------
; Print a string.
;
PrintM: MOV     CL,Pstring      ;
        JMPS    bdos            ;
;
;-------------------------------
; Provides Old fashioned BDOS call.
;
bdos:   INT     bdosi           ; IBM PC
        RET                     ;
;
;-------------------------------
; Print error message and abort.
;
Err:    MOV     DX,OFFSET ErrorM ;
        CALL    printm          ;
        JMP     done            ;
;
;-------------------------------
        DSEG
;-------------------------------
        ORG     100h            ; Reserve Base Page
;
ErrorM  DB      'ERROR',0Dh,0Ah,'$'
bSize   EQU     80h             ; Buffer Size = 128 characters
buffer  RS      bSize           ; Reserve them
bPtr    DB      bSize           ; Buffer Pointer = 1st char
;
;-------------------------------
```

```
;
        END

[Reached the break point at DONE.]

*1CC1:0014 .DONE

[Display the currently active pass points.]

#p
F229 1CC1:005E .CONOUT

[The  pass  count went from FFFF to F229, so the difference is the  number  of
times  CONOUT  was  called, or the number of characters in  the  file.  The  H
command will perform the subtraction, and display the result in decimal.]

#hffff-F229
0DD6 #3542

[3542 characters in the file.]

#^C
A>That's all, folks!
```

Appendix A: SID-86 Error Messages

Table A-1. SID-86 Error Messages

Format: Error Message
Meaning

AMBIGUOUS OPERAND
An attempt was made to assemble a command with an ambiguous operand. Precede the operand with the prefix "BYTE" or "WORD".

BAD FILE NAME
A filename in an E, R, or W command is incorrectly specified.

BAD HEX DIGIT
A SYM file being loaded with an E command has an invalid hexadecimal digit.

CANNOT CLOSE
The disk file written by a W command cannot be closed.

DISK READ ERROR
The disk file specified in an R command could not be read properly.

DISK WRITE ERROR
A disk write operation could not be successfully performed during a W command, probably due to a full disk.

INSUFFICIENT MEMORY

There is not enough memory to load the file specified in an R or E command.

MEMORY REQUEST DENIED
A request for memory during an R command could not be fulfilled either because the maximum number of memory allocations has already been made, or the memory at the specified address is not available. Up to eight blocks of memory can be allocated at a given time under CP/M-86.

NO FILE
The file specified in an R or E command could not be found on the disk.

NO SPACE
There is no space in the directory for the file being written by a W command.

SYMBOL LENGTH ERROR
A symbol in a SYM file being loaded with an E command has more than thirty-one characters.

SYMBOL TABLE FULL
There is no more space in SID-86's symbol table.

VERIFY ERROR AT seg:off
The value placed in memory by a Fill, Set, Move, or Assemble command could not be read back correctly, indicating bad RAM, or attempting to write to ROM or non-existent memory at the indicated location.

Index

(To be done...)

EOF