M68SDBUG/D

September 1997

M68SDBUG

SERIAL DEBUGGER

USER'S MANUAL

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

* P & E Microcomputer Systems, Inc.
PO Box 2044
Woburn, MA 01888-2044
(617)-353-9206

CONTENT

CHAPTER 1 GENERAL INFORMATION

CHAPTER 2 SDbug OPERATING PROCEDURE

CHAPTER 3 USING DEBUGGER COMMANDS

CHAPTER 3 USING DEBUGGER COMMANDS (continued)

CHAPTER 3 USING DEBUGGER COMMANDS (continued)

CHAPTER 4 IASM OPERATING PROCEDURE

CHAPTER 5 PROGRAMMING MCUS

APPENDIX A S-RECORD INFORMATION

APPENDIX B STATUS AND ERROR MESSAGES

FIGURES

TABLES

TABLES (continued)

CHAPTER 1
GENERAL INFORMATION

1.1 INTRODUCTION

This manual provides general information, operating instructions, and MCU programming instructions for your M68SDbug Serial Debugger (SDbug) software.

SDbug works with target devices that incorporate HCMOS microcontroller units (MCUs) of the M68HC12, M68HC16, and M68300 families. The SDbug versions are:

- **SDbug12**, which works with M68HC12 MCUs,
- **SDbug16**, which works with M68HC16 MCUs, and
- **SDbug32**, which works with M68300 MCUs.

1.2 FEATURES

SDbug features include:

- Economical means of evaluating target systems incorporating MCUs of the M68HC12, M68HC16, or M68300 families.
- Background-debug mode operation, for use with a personal computer instead of an on-board monitor.
- Integrated assembly/editing/emulation environment for easy development.
- As many as seven software breakpoints.
- Memory map of your target system.
- IASM development software (IASM12, IASM16, or IASM32), which includes an editor, cross assembler, and communications package for use with MCUs of the corresponding family.
- PROG software (PROG12S, PROG16S, or PROG32S) for programming MCUs of the corresponding family.

1.3 EQUIPMENT REQUIRED

For communication with SDbug, you need an IBM PC or compatible computer running MS-DOS. The computer must have a serial communication port. You also need a Motorola SDI™ interface.

SDbug operates in background debug mode – a backdoor method of talking to the CPU.

There are two methods of loading MCU code using SDbug:

1. Generating code via the SDbug one-line assembler/disassembler.
2. Downloading assembled code from an external source to user program RAM (pseudo ROM) via SDbug.

1.4 LOADING SOFTWARE

To load the SDbug software on your host computer:

1. Insert the software diskette into the 3.5-inch drive of your computer.
2. At the DOS prompt, type the floppy drive letter, followed by the word `install`. For example, if your 3.5-inch drive is drive B, type: **`b:install`**
3. The install program automatically loads the software. Follow the instructions that appear on the screen. This completes software loading.

Refer to the hardware user's manual for connecting your computer to the target system.

1.5 ABOUT THIS MANUAL

The rest of this manual explains SDbug functionality:

- Chapter 2 explains SDbug operating procedure.
- Chapter 3 explains SDbug commands.
- Chapter 4 explains how to use IASM development software.
- Chapter 5 explains how to program MCUs of the M68HC12, M68HC16, and M68300 families.
- Appendix A gives information about Motorola S-records.
- Appendix B explains status and error messages.

A software release guide, which comes with your SDbug, lists part numbers of SDbug components, lists all software files, and explains any restrictions or limitations.

CHAPTER 2
SDbug OPERATING PROCEDURE

2.1 INTRODUCTION

This chapter explains how to use SDbug. The explanations of this chapter cover startup, general use, the main screen, source-level debugging, counts, the variable window, macros, and tracing. Chapter 3 explains debug monitor commands.

2.1.1 Typeface and Parameter Conventions

This chapter uses two different typefaces:

1. `This typeface for examples,`
2. *This typeface for variables, and*
3. This typeface for text and explanations.

Also, note these conventions for parameters and keyboard entries:

- **add** indicates any valid, hexadecimal address or label.
- **file** indicates a DOS file name (with or without the path).
- **IP** is the instruction pointer, which points to the next instruction to be executed (M68HC16 MCUs only).
- **n** indicates any hexadecimal number: 0–0FFFF for bytes, 0–0FFFFF for words, and 0–0FFFFFFFF for longwords.
- **PC** is the program counter, which points to the next instruction to be fetched. (For an MC68HC16 MCU the PC value equals the IP value plus 6.)
- **str** indicates an ASCII character string. The maximum string length is 256 characters.
- **;** indicates that the text string following this character is a comment, not part of the instruction.
- **[ ]** indicate an optional parameter.
- label_name indicates a string of 16 or fewer characters: letters, numbers, or underscores, although the first character must be a letter.
- **<CR>** indicates the ENTER, RETURN, or carriage-return key of your keyboard.

2.1.2 SDbug Numerical Formats

Unless otherwise specified, all numbers in SDbug are hexadecimal. SDbug treats any numbers you enter as hexadecimal, unless you use the proper prefix or suffix, per Table 2-1. (Number values may have a prefix or a suffix, but must not have both.)

Table 2-1. SDbug Number Symbols

Symbol	Meaning
$	Optional prefix for hexadecimal numbers, as *$0FF*
!	Required prefix for decimal numbers, as *!255* (which equals $0FF)
%	Required prefix for binary numbers, as *%11111111* (which equals $0FF and !255,)
H	Optional hexadecimal-number suffix; an alternative to the $ prefix (0FFH = $0FF)
T	Decimal-number suffix; an alternative to the ! prefix (255T = !255)
O	Octal-number suffix
Q	Binary-number suffix; an alternative to the % prefix (11111111Q = %11111111)

2.2 STARTUP

Before starting SDbug software, read the README file (if any) on your software diskette and read the software release guide (if any). A README file or software release guide contains information not available at press time. Then make sure that the SDI™ interface and target system are connected and powered up.

To start SDbug operation, enter the startup command at the DOS prompt:

```
Sdbug# [option] [option] ...
```

where options are parameter values from Table 2-2. Note that spaces must separate multiple option-parameter values.

Table 2-2. Option Parameter Values

Value	Meaning
#	SDbug number: 12, 16 or 32
bw	Set display to black and white.
com1 ... com9	Select specified serial I/O port 1 through 9 (the default I/O port is com1).
baud *n*	Set I/O port baud rate n specifies. The rate range is 2400 to 57600; the default rate is 19200.
freq *n*	Set target frequency n (half the MPB oscillator frequency), entered with all trailing zeros. The Sdbug12 default is 2000000 (2Mhz); the default for Sdbug16 or Sdbug32 is 8000000 (8Mhz).
quiet	Start the debugger without filling the memory windows or disassembly window.
sim *n*	Set SIM type value n specifies — one of these possible values: 0 SIM 4 LIM (NOMUX) 1 SCIM 5 LIM (MUX) 2 RPSCIM 6 SLIM (MUX) 3 SCIM2 7 SLIM (NOMUX) (The Sdbug12 default is 5; the Sdbug16 and Sdbug32 default is 0.)
path	Specify DOS path to the directory and code to be debugged.
running	Start the debugger without executing a RESET (see the GOEXIT command explanation, in Chapter 3).

Example startup commands are:

`SDbug12 \iasm12\source\` Starts SDbug12, making available code in the IASM12 directory, source subdirectory.

`SDbug12 freq 4000000 sim $04`

Starts Sdbug12 for a target-system oscillator frequency of 4MHz, with a non-multiplexed light integration module (LIM).

`SDbug16 start.s19` Starts SDbug16 and performs a LOADALL command, loading the s-record and debug map.

`SDbug32 com2 baud 2400` Starts SDbug32, specifying serial communication port 2 at 2400 baud.

If a file named STARTUP.ICD is in the current directory, the debugger software runs the file as a macro at startup. (Paragraph 3.4 explains SDbug macros.) At startup, SDbug shows an opening screen that lists the version number. Press any key to advance to the main screen (Figure 2-1).

2.3 MAIN SCREEN

Figure 2-1 shows the SDbug12 main screen, which consists of the CPU window, the stack window, the code window, the program (F6) memory window, the data (F3) memory window, and the debug (F1) window.

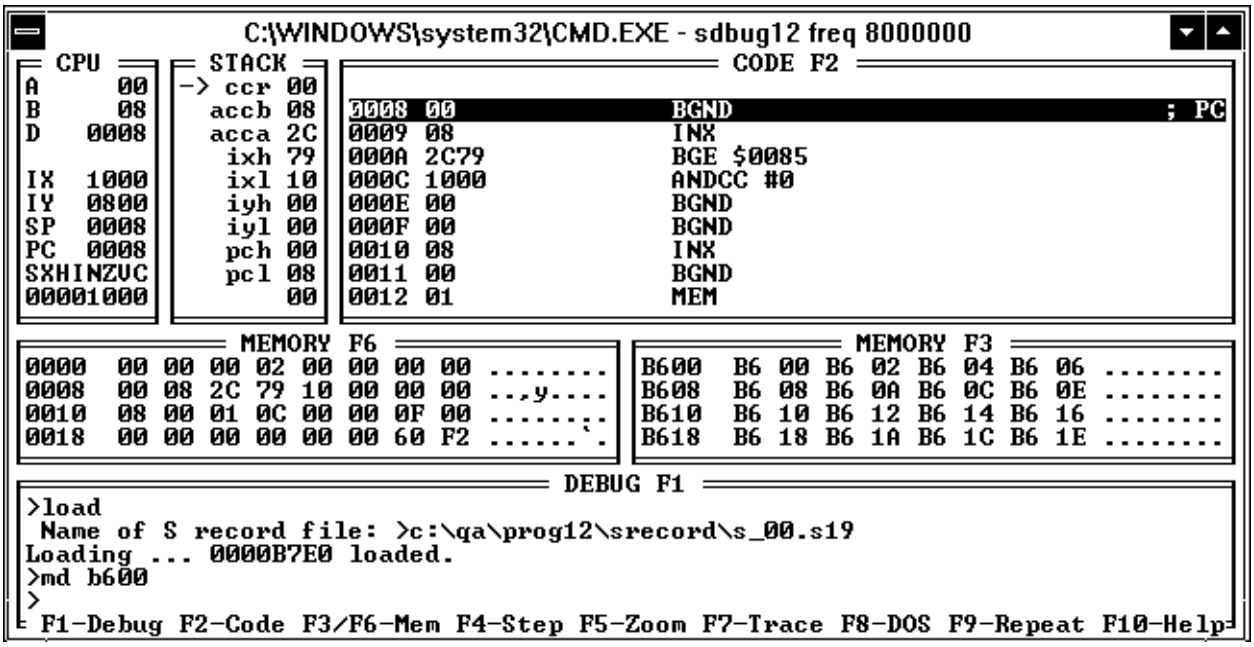


Figure 2-1. SDbug12 Main Screen

Figure 2-2 shows the SDbug16 main screen, which consists of the CPU window, the instruction pointer (IP) window, the breakpoint (BR) window, the code window, the program (F6) memory window, the data (F3) memory window, and the debug (F1) window.

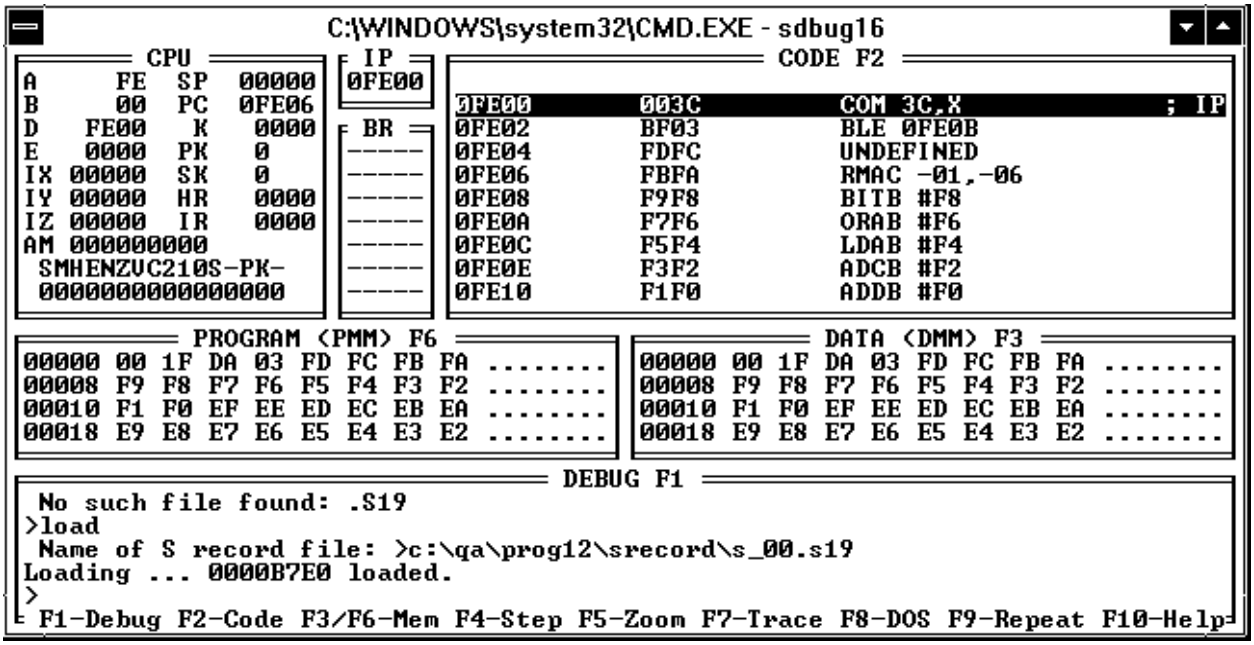


Figure 2-2. SDbug16 Main Screen

Figure 2-3 shows the SDbug32 main screen, which consists of the CPU window, the code window, the program (F6) memory window, the stack window, the data (F3) memory window, and the debug (F1) window.

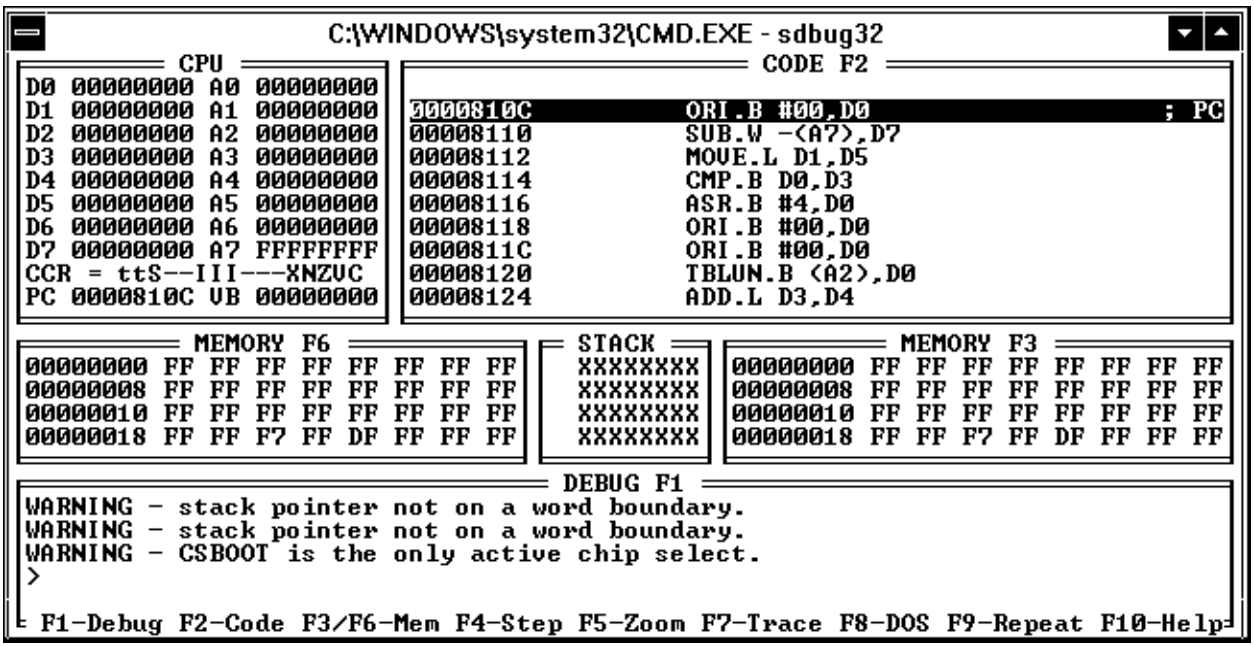
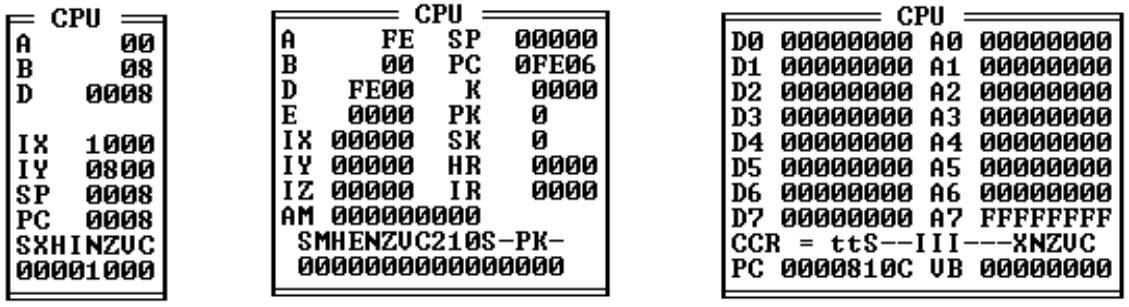


Figure 2-3. SDbug32 Main Screen

Paragraphs 2.3.1 through 2.3.6 explain these windows. Paragraph 2.3.7 explains the use of function keys with these windows.

2.3.1 CPU Window

The CPU window, at the upper left of the main screen, shows the status of CPU registers. Figure 2-4 shows the CPU windows for SDbug12, SDbug16, and SDbug32.



SDbug12 **SDbug16** **SDbug32**

Figure 2-4. CPU Windows

Use the debug window to change any register value in the CPU window: enter the register, a space, and the new value:

```
>a fe
```
Changes the value in register A to $FE.

You may change individual bits in the condition code register by specifying the bit label and entering the value (0 or 1):

```
>h 0
```
Changes the H bit to 0.

or

```
>ccr %00001000
```
Changes the CCR bits to 00001000.

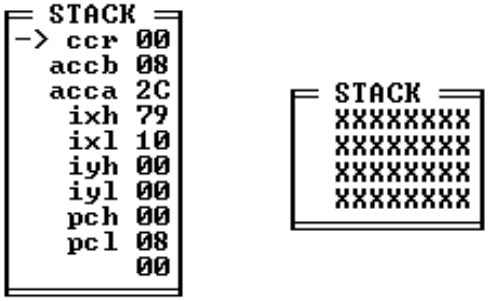
To change the value of the AM register (SDbug16 only), you must specify the high-order 20 bits (AMH) or the low-order 16 bits (AML).

The MCU type determines the registers that may appear in the CPU window:

- **For SDbug12**: A, B, CCR, D, IX, IY, PC, and SP. For more information about these registers, consult the CPU Central Processor Unit Reference Manual for M68HC12-family MCUs.
- **For SDbug16**: A, B, C, CCR, D, E, EK, H, HR, IR, IX, IY, IZ, K, N, PC, PK, S, SK, SP, V, XK, YK, Z, and ZK. (The SDbug16 also can change values of the SIGNLATCH, XMASK, and YMASK registers, but these registers do not appear in the CPU window.) For more information about these registers, consult the CPU Central Processor Unit Reference Manual for M68HC16-family MCUs.
- **For SDbug32**: D0 — D7, A0 — A7, CCR, PC, DFC, SFC, and VB. (In the CCR representation, capital letters indicate values 1, and lower-case letters indicate values 0.) For more information about these registers, consult the CPU Central Processor Unit Reference Manual for M68300-family MCUs.

2.3.2 SDbug12, SDbug32 Stack Window

The stack window displays contents of the stack. For the SDbug12, the stack window is between the CPU and code windows. The SDbug12 stack window shows the first ten values in the stack. For the SDbug32, the stack window is between the memory F6 and memory F3 windows. The SDbug32 stack window shows the contents of the first four long words on the stack. Figure 2-5 shows the SDbug12 and SDbug32 stack windows. (The SDbug16 does not have a stack window.)



SDbug12 **SDbug32**

Figure 2-5. Stack Windows

2.3.3 SDbug16 Instruction Pointer (IP) Window

The IP window, at the top center of the SDbug16 main screen, shows the value of the instruction pointer. Figure 2-6 shows the IP window.

Figure 2-6. SDbug16 IP Window

The IP value specifies the instruction to be executed next. Note that the IP value always is six bytes less than the program counter (PC) value. Use the debug window to change the IP value: enter IP, a space, and the new value.

NOTE

The IP is not a register for M68HC16-family MCUs, as it does not exist in the part.

2.3.4 SDbug16 Breakpoint (BR) Window

The BR window, below the SDbug16 IP window, lists the addresses of active breakpoints. (The same addresses appear in the debug window.) Figure 2-7 shows the BR window.

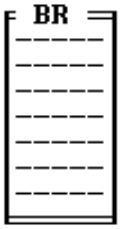


Figure 2-7. SDbug16 BR Window

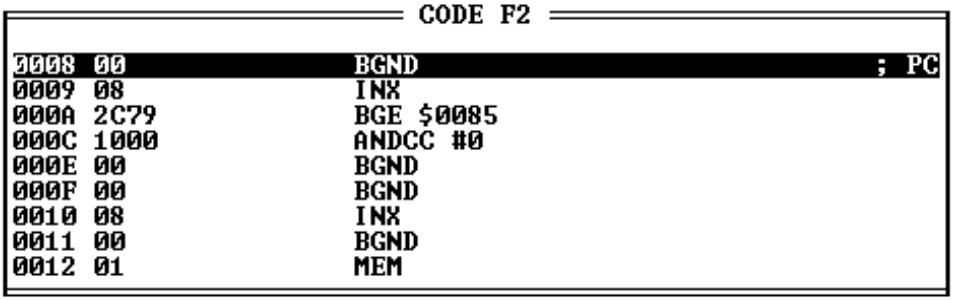
As many as seven breakpoints may be active at once; their addresses are not in any particular order in the BR window.

(The BR window pertains to SDbug16 software. To see a comparable list of breakpoints for SDbug12 or SDbug32 software, enter the BR command, without parameters; the list appears in the debug window.)

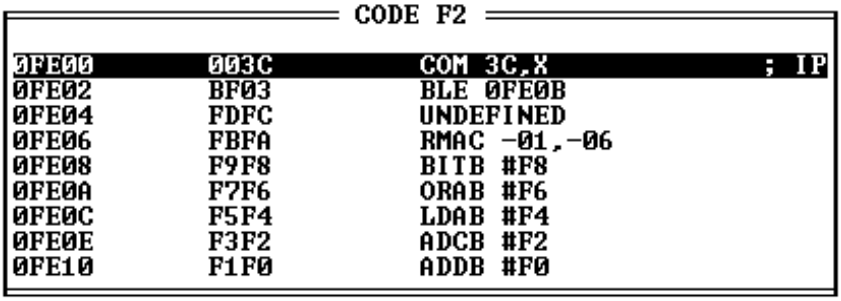
Using the GOTIL or STEPTIL command creates a temporary breakpoint that SDbug16 does not list in the BR window.

2.3.5 Code Window

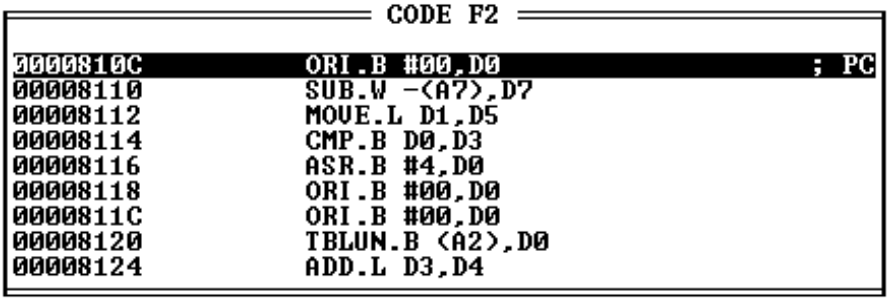
The code window, at the upper right of the main screen, displays code. Figure 2-8 shows the code windows for SDbug12, SDbug16, and SDbug32.



SDbug12



SDbug16



SDbug32

Figure 2-8. Code Windows

The code windows display code in one of three ways:

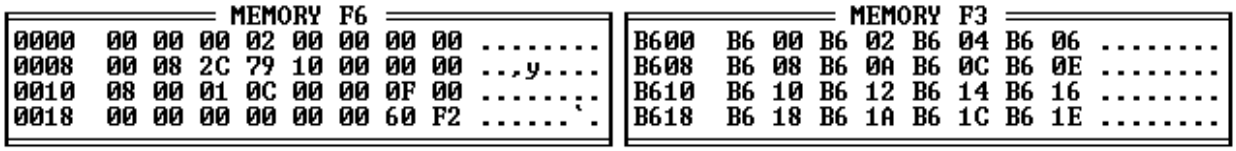
1. If you have not loaded a map file, the code window displays disassembled code. The symbol **;PC**, at the right edge of the window shows the program counter location. The symbol **;IP** (only in the SDbug16 code window) shows the instruction pointer location. The **;TR** (trace) symbol indicates the last instruction the CPU executed. The symbol **;BR** indicates active breakpoints, and the symbol **;CT** indicates counters.
2. If you have loaded a full map file, the code window can display source code. The symbol **->**, at the left of the window, indicates the instruction pointer (SDbug16) or the program counter (SDbug12, SDbug32). The symbols ° and À , at the left of the window, indicate the active breakpoints and counters, respectively.
3. Alternatively, if you have loaded a full map file, the code window can display disassembled code. Labels defined in the source code appear instead of their address values. The symbols **;PC**, **;IP**, **;TR**, **;BR**, and **;CT** have the same roles as for case 1, above.

Paragraph 3.3 explains source-level debugging via the code window.

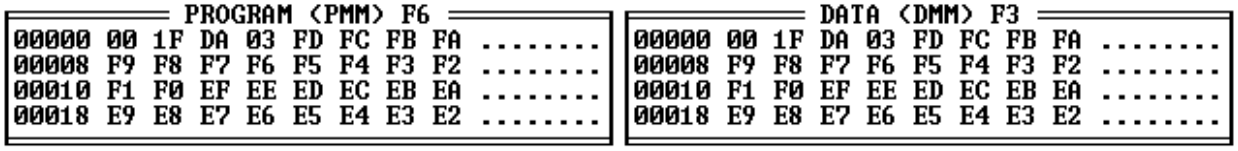
2.3.6 Memory Windows

The F6 and F3 memory windows display memory contents. Values in these windows are hexadecimal; when appropriate, values also are in seven-bit ASCII symbols. A period appears in lieu of an unprintable ASCII character. Figure 2-9 shows the memory windows for SDbug12, SDbug16, and SDbug32.

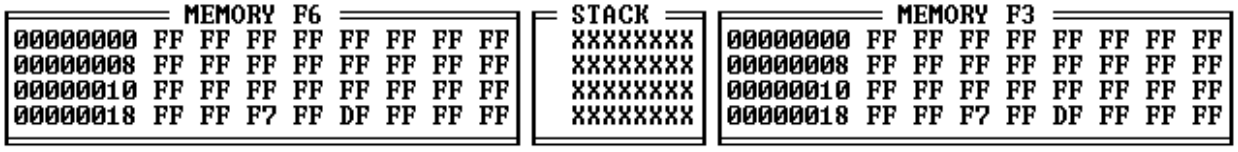
For the SDbug12 or SDbug32, either memory window displays the memory contents you specify. (Also, for the SDbug32, the two memory windows bracket the stack window.) For the SDbug16, the F6 window accesses program memory and the F3 window accesses data memory.



SDbug12



SDbug16



SDbug32

Figure 2-9. Memory Windows

2.3.7 Debug Window

The debug window is at the bottom of the main screen. Figure 2-10 shows the debug window, which is identical for all SDbug versions.

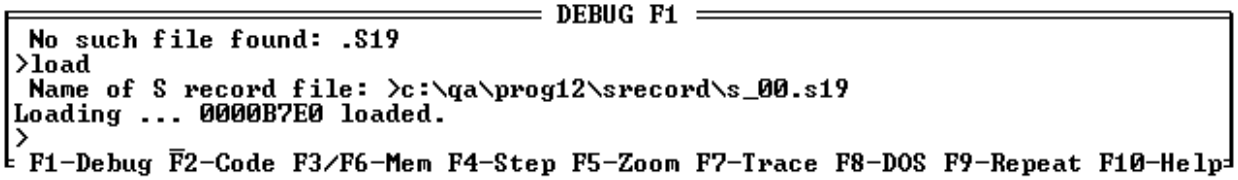


Figure 2-10. Debug Window

Use the debug window to enter commands. The prompt symbol is the > character.

2.3.8 Window Function Keys

To move between windows, and to carry out certain other actions, use the function (F) keys of the keyboard. Table 2-3 lists the functions of these keys.

Table 2-3. SDbug Special Function Keys

F Key	Function
F1	Go to the debug window
F2	Go to the code window. In this window, you may scroll to see future code.
F3	Go to the F3 memory window. In this window, you may scroll through data memory.
F4	Do a single-step trace. (This key has the same role as the STEP command.)
F5	Shrink or enlarge the code window (if it displays source code).
F6	Go to the F6 memory window or variables window. In the former, you may scroll through memory. In the latter, you may scroll through CPU variables.
F7	Go to the code window as a trace window. In this format of the code window, you may scroll through the trace buffer.
F8	Shell to DOS or execute the DOS STRING command.
F9	Repeat the last command.
F10	Activate the help window.
PgUp & PgDn	Page through as many as 30 commands previous entered at the debugger prompt.

2.3.9 Other Windows

In addition to the main-screen windows, SDbug includes temporary windows that appear in certain situations. Help pop-up windows, for example, appear if you press the F10 key or enter the HELP command. (For more information, see the explanation of the HELP command in Chapter 3.)

Another window is the variables window, which replaces the contents of the F6 memory window. The variables window shows values of any variables in the CPU data space, in byte, word, long word, or string format. To activate the variables window, enter the VAR command. To deactivate the variables window, enter the MDF6 or SHOWF6 command. (For more information, see the explanation of the VAR command in Chapter 3.)

2.4 GENERAL USE

Do most of your debugging from the debug window. You may enter all debug commands in this window. Typically, the first command a user enters is one of the load commands. If you have created both an object (.S19) file and a map (.MAP) file via IASM, enter the LOADALL command:

`>LOADALL`	Load code and symbols into SDbug.
`Filename:`	Prompt if no filename is given with the LOADALL command.

In response to the filename prompt, enter the name of the object file. Next, if you have loaded appropriate reset vectors for the code, enter a reset command; this resets the hardware and initializes the IP and PC correctly.

`>RESET`	Reset the hardware.

The code window shows your code, as either disassembled code or actual source code. At this point you may start debugging. To have the F3 (or F6) memory window display code starting at a useful address, enter the SHOWF3 (or SHOWF6) command:

`>SHOWF3 myarray`	Set location **myarray** as the start of the F3 window display.

Now you may begin debugging and testing your code, by setting breakpoints or single-stepping For example, to single-step through 80 instructions, enter this command:

`>STEP 50`	Single-step through $50 (!80) instructions.

NOTE

For many M68HC16 and M68300 MCUs, the watchdog timer is active after a reset. Using the WATCHDOG command disables the timeout, preventing watchdog resets from occurring during your debug session.

CHAPTER 3
USING DEBUGGER COMMANDS

3.1 INTRODUCTION

This chapter explains SDbug debugger commands, and covers related information on source-level debugging, macros, and the trace buffer.

3.2 DEBUG WINDOW COMMANDS

Table 3-1 lists SDbug debug commands, noting the CPU types that each command supports. Paragraphs 3.2.1 through 3.2.70 are standard-format explanations of the individual commands.

Table 3-1. Debug Window Commands

Syntax	Meaning	SDbug Type
ASCIIF3	Toggle F3 window display	32
ASCIIF6	Toggle F6 window display	32
ASM add	Assemble into pseudo ROM	12/16/32
BAUD baud	Set baud rate	12/16/32
BF[.X] add add n	Block fill memory	12/16/32
BR [add [n]]	Set or remove breakpoint	12/16/32
BW	Change display to black-and-white mode	12/16/32
CAPTURE file	Save data to a log file	12/16/32
CAPTUREOFF	Stop logging data to log file	12/16/32
CLEARMAP	Remove source-level debug information	12/16/32
CLEARSYMBOL	Remove temporary symbols	12/16/32
CODE add	Show disassembled code	12/16/32
COUNT [add] [add]	Counts execution	12/16/32
COUNTER [add]	Add or remove counter location	12/16/32
DEBUGGER_DFC [n]	Set debugger destination function code value	32

Table 3-1. Debug Window Commands (continued)

Syntax	Meaning	SDbug Type
DEBUGGER_SFC [n]	Set debugger source function code value	32
DFC [n]	Set MCU DFC Value	32
DMM[.X] add [n]...[n]	Modify RAM data	16
DOS	Shell to DOS or executes the DOS_STRING	12/16/32
DOS_STRING str	Specify DOS Command	12/16/32
DUMP[.X] add add [n]	Load memory to debug window	12/16/32
DUMP_TRACE	Load trace buffer to debug window	12/16/32
EVAL n [op] [n]	Evaluate expression	12/16/32
EXIT	Exit the program	12/16/32
GO [add] [add]	Execute program	12/16/32
GOEXIT [add]	Exit debugger with target running	12/16/32
GONEXT	Go from IP/PC to Next Instruction	12/16/32
GOTIL add	Go from IP/PC to address	12/16/32
GOTILROM add	Single-step fast to address	12/16/32
HELP	Display help system	12/16/32
LOAD file	Load S-record file	12/16/32
LOADALL file	Load map and S-record files	12/16/32
LOADMAP file	Load IASM debug file	12/16/32
LOADV file	Load and verify file	12/16/32
LOAD_BIN file add	Load binary file	12/16/32
LOADV_BIN file add	Load and verify binary file	12/16/32
MAC	Display MAC unit contents	16
MACRO [file]	Execute macro file	12/16/32
MACROEND	End macro file	12/16/32
MACROSTART [file]	Start macro file	12/16/32
MACS	Show macro list and directory structure	12/16/32

Table 3-1. Debug Window Commands (continued)

Syntax	Meaning	SDbug Type
MDF3 add	Set F3 window memory display	12/16/32
MDF6 add	Set F6 window memory display	12/16/32
MM[.X] add [n]...[n]	Modify memory	12/16/32
NOBR	Remove all breakpoints	12/16/32
PMM[.X] add [n]...[n]	Modify memory in program space	16
QUIET	Toggle window refresh	12/16/32
REM	Write remark	12/16/32
RESET	Execute hardware reset	12/16/32
SERIAL port baud par dbit sbit	Set serial-port parameters	12/16/32
SERIALOFF	Deactivate serial port	12/16/32
SERIALON	Activate serial port	12/16/32
SFC [n]	Set MCU SFC value	32
SNAPSHOT	Screen capture	12/16/32
SOURCE	Toggle code display	12/16/32
SOURCEPATH [path]	Code search	12/16/32
SS	Source step	12/16/32
STATUS	Show processor status	12/16/32
STEP [n]	Step trace	12/16/32
STEPFOR	Step to breakpoint	12/16/32
STEPTIL add	Step to address	12/16/32
SYMBOL chars n	Add symbol to map file	12/16/32
TRACE [add] [add]	Execute trace	12/16/32

Table 3-1. Debug Window Commands (continued)

Syntax	Meaning	SDbug Type
UPLOAD_SREC add add	Display S-record files	12/16/32
VAR[.X] [add] [n]	Show variables window	12/16/32
VERIFY [file]	Compare file to memory	12/16/32
VERSION	Display the current software version	12/16/32
WATCHDOG	Disable watchdog timer	16/32
WHEREIS symbol	Show symbol value	12/16/32

ASCIIF3 Toggle F3 Window Display

3.2.1 Toggle F3 Window Display

```
ASCIIF3
```

This command toggles the F3 window display between hexadecimal bytes and ASCII characters. (The display shows periods in lieu of non-printable characters.)

NOTE

This command is only available in SDbug32.

Example:

`>ASCIIF3`	Change F3 window display from hexadecimal bytes to ASCII characters (or vice versa).

ASCIIF6 Toggle F6 Window Display

3.2.2 Toggle F6 Window Display

```
ASCIIF6
```

This command toggles the F6 window display between hexadecimal bytes and ASCII characters. (The display shows periods in lieu of non-printable characters.)

NOTE

This command is only available in SDbug32.

Example:

`>ASCIIF6`	Change F6 window display from hexadecimal bytes to ASCII characters (or vice versa).

ASM Assemble into Pseudo ROM

3.2.3 Assemble into Pseudo ROM

```
ASM add
```

where:

add Starting address or label for assembly.

This command invokes the one-line assembler, starting at the specified address.

This assembler assembles user code, including labels, provided that the labels are in a previously loaded map file. If no map file is loaded, you may not use labels. The ASM command does not define labels. Your entry must start with an opcode.

During assembly, SDbug shows the current instruction in the debug window. To modify this instruction, type in a new one. If the code window shows disassembled code, you will see the change immediately.

To advance to the next location without changing the present location, press <CR>. To end the assembly session, type a period (.) at the prompt.

You may press the F10 (help) key to see the format of assembly-language instructions.

Examples:

`>ASM 10500`	Start assembly at location $10500.
`10500 274c >nop`	Shows disassembly; prompts for new opcode.

BAUD Set Baud Rate

3.2.4 Set Baud Rate

```
BAUD baud
```

where:

baud Serial communication speed between 2400 to 57600.

Use this command to set the serial communication speed.

Examples:

```
>BAUD 57600
```
Set the communication baud rate at 57600.

BF Block Fill Memory

3.2.5 Block Fill Memory

```
BF[.X] add add n
```

where:

.X	Program space units: .B or .b = bytes, .W or .w = words, .L or .l = long words.
add	First parameter: fill operation lower limit;
	Second parameter: fill operation upper limit.
n	Fill pattern of the size set by the .X option.

This command repeats the value n specifies throughout the specified user memory range. An invalid address leads to an error message. You may use this command at the beginning of a debug session to initialize an area of memory or an array. The alias of the BF command is FILL.

Examples:

`>BF C000 C030 FF`	Assign value $FF to each byte, $C000–$C030.
`>BF C000 C000 0`	Assign value 0 to location $C000.

BR Set or Remove Breakpoint

3.2.6 Set or Remove Breakpoint

```
BR [add [n]]
```

where:

add	Address or label of breakpoint to be set or removed.
n	Delayed occurrence number (default = 1).

This command sets or removes a breakpoint at the specified address (by modifying the breakpoint address table). The debug window shows breakpoint addresses; as many as seven breakpoints may be active at any time. The SDbug16 BR window also shows breakpoint addresses. The order of breakpoints in the window has no effect on their operation.

An n parameter value in the command delays the occurrence of the breakpoint. Program execution passes through the breakpoint-address instruction n-1 times. (Each pass decrements the value of n.) The nth time execution arrives at the instruction, the breakpoint occurs (and n regains the value you specified in the BR command).

Enter the BR command without parameter values to see a list of active breakpoints, with the current values of their n parameters. To clear a breakpoint you must re-enter the breakpoint address. You cannot redefine a breakpoint with a different occurrence number. You must clear the breakpoint and then set a new breakpoint..

For a temporary, additional breakpoint in RAM or pseudo ROM, use the GOTIL or STEPTIL command. For such a breakpoint in ROM, use the GOTILROM command.

Examples:

`>BR 200 3`	Set breakpoint at address $200, to occur the third time execution arrives at the address.
`>BR start1`	Set breakpoint at label start1.

BW Change Display to Black-and-White Mode

3.2.7 Change Display to Black-and-White Mode

```
BW
```

This command puts the display in black-and-white mode. Use this command if the default color display is hard to read (for example, if you use a lap-top computer). Note also that BW is an option for the SDbug startup command. You may change to black-and-white mode only once, following a hardware reset; there is no command to return to color mode.

Example:

`>BW`	Put display in black-and-white mode.

CAPTURE Save Data to a Log File

3.2.8 Save Data to a Log File

```
CAPTURE file
```

where:

file Name of log file.

This command logs debug window outputs to the specified file. A prompt asks whether to append log entries to the specified file or to overwrite previous file contents. If the specified file does not yet exist, SDbug creates the file.

(This logging stops when you enter the CAPTUREOFF command or end your debugging session.)

Example:

`>capture capfile1.cap` Log debug window data to file `capfile1.cap`.

CAPTUREOFF Stop Logging Data to Log File

3.2.9 Stop Logging Data to Log File

```
CAPTUREOFF
```

This command stops logging of debug window outputs. Ending your debugging session is another way to stop such logging.

(Enter the CAPTURE command to begin logging debug window outputs.)

CLEARMAP Remove Source-Level Debug Information

3.2.10 Remove Source-Level Debug Information

```
CLEARMAP
```

This command clears the previously loaded map file from SDbug software.

Clearing the map file eliminates symbols and source code debugging. The code window defaults to simple disassembly.

Example:

`>CLEARMAP`	Delete the current map information.

CLEARSYMBOL Remove Temporary Symbols

3.2.11 Remove Temporary Symbols

```
CLEARSYMBOL
```

This command clears the SDbug symbol table.

Example:

`>CLEARSYMBOL`	Delete the current information in the symbol table.

CODE Show Disassembled Code

3.2.12 Show Disassembled Code

```
CODE add
```

where:

add Starting address or label for disassembled code.

This command is an alternative to scrolling in the code window. After execution of any instruction, the code window reverts to showing IP and PC values.

Examples:

`>CODE 10300`	Show code starting at address $10300.
`>CODE sub1`	Show code starting at label **sub1**.

COUNT Counts Execution

3.2.13 Counts Execution

```
COUNT [add] [add]
```

where:

add First parameter: execution starting address or label;

Second parameter: breakpoint address or label.

This command tells how many times a piece of code executes; this can be useful information for optimizing code. This command starts and stops processor execution of instructions according to the specified address parameters. When execution stops, count values appear in the code window. Count values pertain to addresses listed in the internal counter table; use the COUNTER command to add an address to (or remove an address from) the internal table. Parameter possibilities are:

- If the command has two parameter values, the system sets a new breakpoint at the second address or label, then executes code from the first address or label. Execution continues until it arrives at a breakpoint (which could be the one just set) or until you press a key. Note that the new breakpoint is permanent: it remains set until you remove it via the BR or NOBR command.

NOTE

Remember that the maximum number of breakpoints is seven. If seven breakpoints already exist, but you include a second add parameter value in the COUNT command, the software will ignore that second add value and issue an error message.

- If the command has one parameter value, the system executes code from that address or label until it arrives at an existing breakpoint, or until you press a key.
- If the command has no parameter values, the system executes code from the IP/PC value until it arrives at an existing breakpoint, or until you press a key.

When COUNT stops the results are displayed in a temporary window occupying the source window space. You must press the escape key (Esc) to exit the window.

The processor runs at full execution speed during a COUNT command. However, there is a brief pause each time execution arrives at a counter address; the screen does not show this pause.

COUNT Counts Execution

Examples:

`>COUNT start time1`	Start code execution at label start, stop at label time1, then show count values of included counter addresses.
`>COUNT 1050`	Start code execution at address $1050; at stop, show count values of included counter addresses.
`>COUNT start`	Start code execution at label start; at stop, show count values of included counter addresses.
`>COUNT`	Start code execution at IP/PC value; at stop, show count values of included counter addresses.

COUNTER Add or Remove Counter Location

3.2.14 Add or Remove Counter Location

```
COUNTER [add]
```

where:

add Address or label.

This command adds or removes an address from the SDbug internal counter table. For each address, the system counts code execution. For example, if the counter table contains three addresses included in a block of code that you execute twice, the system increments the counts for these addresses by 2. If you enter this command without a parameter value, the system shows a list of all current counters. The maximum number of addresses in the internal counter table is 50. (The internal counter table is the source for count values displayed via the COUNT command.)

Examples:

`>COUNTER 10300`	Add address 10300 to (or remove the address from) the internal counter table.
`>COUNTER`	Show list of current counters.

DEBUGGER_DFC Set Debugger DFC Value

3.2.15 Set Debugger Destination Function Code Value

```
DEBUGGER_DFC [n]
```

where:

n Destination function code value.

This command sets the debugger destination function code (DFC) to the specified n value. (The default n value is 5). This new value is valid only when the processor is not executing. SDbug uses only the three least significant bits of the n value.

If you enter this command without an n value, the system displays the current DEBUGGER_DFC value.

NOTE

This command is only available in SDbug32.

Examples:

`>DEBUGGER_DFC 4`	Set DFC value to $4.
`>DEBUGGER_DFC`	Show current DFC value.

DEBUGGER_SFC Set Debugger SFC Value

3.2.16 Set Debugger Source Function Code Value

```
DEBUGGER_SFC [n]
```

where:

n Source function code value.

This command sets the debugger source function code (SFC) to the specified n value. (The default n value is 5). This new value is valid only when the processor is not executing. SDbug uses only the three least significant bits of the n value.

If you enter this command without an n value, the system displays the current DEBUGGER_SFC value.

NOTE

This command is only available in SDbug32.

Examples:

`>DEBUGGER_SFC 4`	Set SFC value to $4.
`>DEBUGGER_SFC`	Show current SFC value.

DFC Set MCU DFC Value

3.2.17 Set MCU DFC Value

```
DFC [n]
```

where:

n Destination function code value.

This command sets the destination function code (DFC) to the specified n value. This new value is valid only while the processor is executing. If you are in background debug mode, use the DEBUGGER_DFC command, not the DFC command, to set the DFC. SDbug uses only the three least significant bits of the n value.

If you enter this command without an n value, the system displays the current DFC value.

NOTE

This command is only available in SDbug32.

Examples:

`>DFC 4`	Set DFC value to $4.
`>DFC`	Show current DFC value.

DMM Modify RAM Data

3.2.18 Modify RAM Data

```
DMM[.X] add [n] ... [n]
```

where:

.X	RAM units: .B or .b = bytes, .W or .w = words, .L or .l = long words.
add	RAM address or label to receive data value.
n	Data to be entered. The size is defined by the .X option.

This command writes the specified data into RAM at the specified address. Consecutive data values (separated by spaces) go into consecutive memory units the .X parameter specifies (the default is bytes.)

If the command line does not specify data, the software prompts for data, one memory unit at a time. Such prompts include the memory location and current value. To change the value, enter the new value. To advance to the next location without changing the present location, press <CR>. To exit this command, type a period (.) at the prompt.

NOTE

This command is only available in SDbug16.

Examples:

`>DMM 100 1 2 3 4`	Put values 1–4 into locations $100–$103.
`>DMM.B 200`	Start interactive memory modification, in bytes.
`200 = 41 >`	Shows current value, prompts for new one.

DOS Shell to DOS

3.2.19 Shell to DOS

```
DOS
```

This command suspends the debugger, exiting to the DOS prompt. If you have defined a DOS command via the DOS_STRING command, the system immediately executes that DOS command and returns to SDbug.

To return to the debugger, enter EXIT at the DOS prompt.

Example:

`>DOS`	Suspend SDbug and shell to DOS.
C:\>	DOS prompt.

DOS_STRING Specify DOS Command

3.2.20 Specify DOS Command

```
DOS_STRING str
```

where:

str Command to be executed by DOS

The DOS_STRING command specifies a DOS command to be executed in background. If a value is defined for DOS_STRING and you shell to DOS, the string is then executed in background. Enter the DOS_STRING command with no parameters to clear any value defined using the DOS_STRING command. You can then shell to DOS via the DOS command.

Example:

`>DOS_STRING run.exe`	Define the routine to be executed the next time you execute a DOS command.
`>DOS`	Execute the run.exe command in background.
`>DOS_STRING`	Clears the DOS_STRING.
`>DOS`	Shell to DOS.

DUMP Load Memory to Debug Window

3.2.21 Load Memory to Debug Window

```
DUMP[.X] add add [n]
```

where:

.X	RAM units: .B or .b = bytes, .W or .w = words, .L or .l = long words.
add	First parameter: memory range start address;
	Second parameter: memory range end address.
n	Number of units to be displayed on each line. The size is defined by the .X option.

This command uploads data from memory for display in the debug window.

The two add parameter values specify the memory range to be uploaded. The .X parameter value specifies the uploading units (the default unit is bytes). If the .X parameter specifies words or long words, the first add parameter must specify an address on an even boundary. The n parameter specifies the number of memory units (bytes, words, or long words) to be written on one line.

If the capture feature is active, SDbug also logs the lines of dumped data to the capture file.

Examples:

```
>DUMP.B 0 8 3
Dumping memory data, Press any key to abort.
00000000 4E 71 4E
00000003 71 4E 71
00000006 4E 71 4A
done.

>DUMP.W 0 6 2
Dumping memory data, Press any key to abort.
00000000 4E71 4E71
00000004 4E71 4E71
done.

>DUMP.L 0 12 4
Dumping memory data, Press any key to abort.
00000000 4E71FFFF 4E71FFFF 4E71FFFF 4E71FFFF
done.
```

DUMP_TRACE Load Trace Buffer to Debug Window

3.2.22 Load Trace Buffer to Debug Window

```
DUMP_TRACE
```

This command displays the current trace-buffer contents in the debug window. If the capture feature is active, SDbug also logs the lines of displayed data to the capture file.

Example:

```
>DUMP_TRACE             Dump trace buffer.
00000004          NOP                          ; PC
00000002          NOP                          ;-0001
00000000          NOP                          ;-0002
```

EVAL Evaluate Expression

3.2.23 Evaluate Expression

```
EVAL n [op] [n]
```

where:

n	First parameter: Expression or first term to be evaluated;
	Third parameter: second term to be evaluated.
op	Operator.

This command evaluates an arithmetic expression, giving the result in hexadecimal, decimal, octal, and binary values. This command uses 32-bit, signed integer arithmetic for its evaluations. The expression can contain the operators for addition, subtraction, multiplication, and division (+, –, *, and /). Single spaces must separate parameter values.

If this command had only one parameter value, it echoes that value in four bases.

Example:

```
>EVAL 102T + 54                                  Evaluate !102 plus $54.
000BAH 186T 000002720 00000000000010111010Q     Answer in four bases.
```

EXIT Exit Debugger

3.2.24 Exit Debugger

```
EXIT
```

This command exits the program, returning to DOS. (Alternatives to the EXIT command are entering the QUIT command or pressing ALT-X.)

Example:

```
>EXIT                    Return to DOS.
```

GO Execute Program

3.2.25 Execute Program

```
GO [add] [add]
```

where:

add First parameter: execution starting address or label;

Second parameter: breakpoint address or label.

Use the GO command to execute user code (G is the alias of the GO command). This command starts and stops processor execution of instructions according to the specified address parameters:

- If the command has two parameter values, the system sets a new breakpoint at the second address or label, then executes code from the first address or label. Execution continues until it arrives at a breakpoint (which could be the one just set) or until you press a key. Note that the new breakpoint is permanent: it remains set until you remove it via the BR or NOBR command.

NOTE

Remember that the maximum number of breakpoints is seven. If seven breakpoints already exist, but you include a second add parameter value in the GO command, the software will ignore that second add value and issue an error message.

- If the command has one parameter value, the system executes code from that address or label until it arrives at an existing breakpoint, or until you press a key.
- If the command has no parameter values, the system executes code from the IP/PC value until it arrives at an existing breakpoint, or until you press a key.

The processor runs at full execution speed during a GO command.

NOTE

You may use the computer serial port for processor execution of instructions, converting the F1 window to a dumb terminal. To arrange this functionality, use the commands SERIAL, SERIALON, and SERIALOFF, as well as the GO command. For more information, see the explanations of these other commands.

GO Execute Program

Examples:

`>GO start time1`	Start code execution at label start; break at label time1.
`>GO 1050`	Start code execution at address $1050.
`>G start`	Start code execution at label start.
`>GO`	Start code execution at IP/PC value.

GOEXIT Exit Debugger with Target Running

3.2.26 Exit Debugger with Target Running

```
GOEXIT [add]
```

This command starts target-processor execution of instructions from the specified address parameter, then exits SDbug while the target system continues executing instructions. The target system runs without breakpoints. If you enter this command without an add parameter value, instruction execution begins from the current IP/PC value.

To restart the debugger without interrupting the target system, you must include the **running** option in the SDbug startup command. This restarts the debugger without executing a reset.

NOTE

If you use the GOEXIT command to start instruction execution and exit the debugger, then restart the debugger without the running option, the system halts instruction execution.

GONEXT Go from IP/PC to Next Instruction

3.2.27 Go from IP/PC to Next Instruction

```
GONEXT
```

This command inserts a temporary breakpoint at the first instruction after the current IP (SDbug16) or PC (SDbug12, SDbug32) value. Code execution stops when it reaches this temporary breakpoint. This command works only with program code in RAM or pseudo ROM. (To debug code in ROM, use the GOTILROM command.)

NOTE

This command is similar to the STEP command, but does not stop inside a subroutine or interrupt. If the current IP or PC points to a subroutine or interrupt, GONEXT stops code execution at the first instruction *after* the subroutine or interrupt.

The processor runs at full execution speed during a GONEXT command.

Example:

`>GONEXT`	Execute code from IP/PC value to next instruction.

GOTIL Go from IP/PC to Address

3.2.28 Go from IP/PC to Address

```
GOTIL add
```

where:

add Execution stop address or label.

This command inserts a temporary breakpoint at the specified address or label, then starts execution of code at the IP (SDbug16) or PC (SDbug12, SDbug 32) value. Code execution stops when it reaches the temporary breakpoint. This command works only with program code in RAM or pseudo ROM. (To debug code in ROM, use the GOTILROM command.)

The processor runs at full execution speed during a GOTIL command.

Examples:

`>GOTIL sub1`	Execute code from IP/PC to label sub1.
`>GOTIL 1055`	Execute code from IP/PC to address $1055.

GOTILROM Single-Step Fast to Address

3.2.29 Single-Step Fast to Address

```
GOTILROM add
```

where:

add Destination stop address or label, in ROM.

This command rapidly single-steps through code in ROM, from the IP (SDbug16) or PC (SDbug12, SDbug32) value to the specified address or label. The processor does not run at full execution speed during a GOTILROM command.

Examples:

`>GOTILROM sub1`	Single-step through code from IP/PC to label sub1.
`>GOTILROM 1055`	Execute code from IP/PC to address $1055.

HELP Display Help System

3.2.30 Display Help System

```
HELP
```

This command, an alternative to F10, activates the help system. The help system works via temporary pop-up windows. Use these keys to maneuver in help windows:

arrow keys	move within a menu
<CR>	go to a chosen menu
ESC	go to a previous menu or exit
PgDn	go back one menu page
PgUp	go forward one menu page

Example:

`>HELP`	Activate the help system.

LOAD Load S-Record File

3.2.31 Load S-Record File

```
LOAD [file]
```

where:

file Name of file that contains object code, in S-record format.

This command loads object code of the specified file into your target hardware. The code must be in S-record format. If the file is not in the current directory, enter the entire DOS path. If you do not specify a file extension, the system assumes the extension **.S19**.

Note that the LOAD command only loads a file; it does not verify loading, do a reset, or affect any CPU resources. (To load and verify, use the LOADV command.)

Example:

`>LOAD myfile1` Load object code file `myfile1.s19`.

LOADALL Load Map and S-Record Files

3.2.32 Load Map and S-Record Files

```
LOADALL [file]
```

where:

file — Name of file that contains object code, in S-record format, and of corresponding debug map file.

This command loads a specified object-code file and a debug map file at the same time. The object code must be in S-record format. If the files are not in the current directory, enter the entire DOS path. The system assumes the file extensions **.S19** and **.MAP**. (Use IASM to create map files, per Chapter 4.)

Note that the LOADALL command only loads files; it does not verify loading, do a reset, or affect any CPU resources. (To load and verify, use the LOADV command.)

Example:

`>LOADALL myfile2`	Load object code file `myfile2.s19` and map file `myfile2.map`.

LOADMAP Load IASM Debug File

3.2.33 Load IASM Debug File

```
LOADMAP [file]
```

where:

file Name of debug map file.

This command loads the specified debug map file into SDbug. If the file is not in the current directory, enter the entire DOS path. If you do not specify a file extension, the system assumes the extension **.MAP**. (Use IASM to create map files, per Chapter 4.)

Example:

```
>LOADMAP myfile3
```
Load map file `myfile3.map`.

LOADV Load and Verify File

3.2.34 Load and Verify File

```
LOADV [file]
```

where:

file Name of file that contains object code, in S-record format.

This command loads object code of the specified file into your target hardware, then begins verification of the file. The code must be in S-record format. If the file is not in the current directory, enter the entire DOS path. If you do not specify a file extension, the system assumes the extension **.S19**.

Verification is comparison of the file contents with the contents of program memory. Verification ends the first time the system finds memory locations whose values do not match. In such a case, the address of the discrepancy appears, then the system returns to the > prompt. If memory contents match perfectly, a confirmation message appears before the system returns to the > prompt.

This command is the same as the successive commands LOAD and VERIFY.

Example:

`>LOADV myfile1` Load and verify file `myfile1.s19`.

LOAD_BIN Load Binary File

3.2.35 Load Binary File

```
LOAD_BIN file add
```

where:

file	Name of file that contains binary data code.
add	Loading starting address.

This command loads binary code of the specified file into your target hardware, starting at the add parameter value. If the file is not in the current directory, enter the entire DOS path.

Note that the LOAD_BIN command only loads a binary file; it does not verify loading, do a reset, or affect any CPU resources. (To load and verify, use the LOADV_BIN command.)

Example:

`>LOAD_BIN myfile1.bin $100` Load binary file `myfile1.bin` at address $100.

LOADV_BIN Load and Verify Binary File

3.2.36 Load and Verify Binary File

```
LOADV_BIN file add
```

where:

file	Name of file that contains binary data code.
add	Loading starting address.

This command loads binary code of the specified file into your target hardware, starting at the add parameter value, then begins verification of the file. If the file is not in the current directory, enter the entire DOS path.

Verification is comparison of the file contents with the contents of program memory. Verification ends the first time the system finds memory locations whose values do not match. In such a case, the address of the discrepancy appears, then the system returns to the > prompt. If memory contents match perfectly, a confirmation message appears before the system returns to the > prompt.

This command is the same as the successive commands LOAD_BIN and VERIFY.

Example:

`>LOADV_BIN myfile1.bin $100` Load binary file `myfile1.bin` at address $100.

MAC Display MAC Unit Contents

3.2.37 Display MAC Unit Contents

```
MAC
```

This command displays the contents of the multiply and accumulate unit. This unit includes the XMASK, YMASK, and SIGNLATCH.

NOTE

This command is only available in SDbug16.

Example:

```
>MAC                    Display MAC unit contents.
```

MACRO Execute Macro File

3.2.38 Execute Macro File

```
MACRO [file]
```

where:

file Macro file name.

This command starts execution of the specified macro file. If you do not supply a file extension, the system assumes the extension **.ICD**. You may not nest SDbug macros. (Paragraph 3.4 gives more information about macros.)

The MACRO command does not define a macro file, it executes a macro file already defined. Use the MACROSTART and MACROEND commands to define a macro file.

Example:

`>MACRO divby3` Execute macro file `divby3.icd`.

MACROEND End Macro File

3.2.39 End Macro File

```
MACROEND
```

This command ends the definition of a macro file and stores the file.

Use the MACROSTART command to begin the definition of a macro file; use the MACRO command to execute a macro file.

Example:

```
>MACROEND          End current macro definition.
```

MACROSTART Start Macro File

3.2.40 Start Macro File

```
MACROSTART [file]
```

where:

file Name for the new macro file.

This command starts definition of a new macro file. If you do not supply a file extension, the system assumes the extension **.ICD**. Subsequent commands you enter become part of the new macro.

A macro file is a simple ASCII file that contains one command per line. You may not nest SDbug macros; that is, you may not use the name of one macro in the definition of another.

To end the macro definition, enter the MACROEND command. Subsequently, to execute the macro, enter the MACRO command. (Paragraph 3.4 explains more about macros.)

Example:

`>MACROSTART double` Start definition of macro file `double.icd`.

MACS Show Macro List

3.2.41 Show Macro List

```
MACS
```

This command shows a list of all macro files (that is, all files with the extension **.ICD**) and directory structure, in a temporary window. Use the arrow keys to highlight a macro name. To begin execution of the highlighted macro, press <CR>. To close the temporary window without executing a macro, press <ESC>.

Example:

`>MACS`	Show list of macros.

MDF3 Set F3 Window Memory Display

3.2.42 Set F3 Window Memory Display

```
MDF3 add
```

where:

add Starting address or label for code in the window

This command resets the F3 memory window, so that the window show code starting at the specified address. The alias of MDF3 is SHOWF3.

Examples:

`>MDF3 200`	Start F3 window display with address $200.
`>MDF3 myarray`	Start F3 window display at label myarray.

MDF6 Set F6 Window Memory Display

3.2.43 Set F6 Window Memory Display

```
MDF6 add
```

where:

add Starting address or label for code in the window

This command resets the F6 memory window, so that the window shows code starting at the specified address. In particular, this command deactivates the variables window, restoring the F6 memory window to the main screen. The alias of MDF6 is SHOWF6.

Examples:

`>MDF6 8000`	Start F6 window display with address $8000.
`>MDF6 table`	Start F6 window display at label table.

MM Modify Memory

3.2.44 Modify Memory

```
MM[.X] add [n] ... [n]
```

where:

.X	RAM units: .B or .b = bytes, .W or .w = words, .L or .l = long words.
add	RAM address or label to receive data value.
n	Data to be entered. The size is defined by the .X option.

For the SDbug12, this command writes the specified data to the specified memory address.

For the SDbug16, this command is an alternate form of the DMM command, writing the specified data into RAM at the specified address.

For the SDbug32, this command does the same thing, except that it writes the data to space that the DEBUGGER_DFC or DEBUGGER_SFC command specifies.

Examples:

`>MM 100 1 2 3 4`	Put values 1–4 into locations $100–$103.
`>MM.B 200`	Start interactive memory modification, in bytes.
`200 = 41 >`	Shows current value, prompts for new one.

NOBR Remove All Breakpoints

3.2.45 Remove All Breakpoints

```
NOBR
```

This command removes all the addresses from the breakpoint address table. (The SDbug16 BR window shows this table).

Example:

>NOBR	Remove all breakpoints.

PMM Modify Memory in Program Space

3.2.46 Modify Memory in Program Space

```
PMM[.X] add [n] ... [n]
```

where:

.X	Program space units: .B or .b = bytes, .W or .w = words, .L or .l = long words.
add	Program space address or label to receive data value.
n	Data to be entered. The size is defined by the .X option.

This command writes the specified data into program space at the specified address. Consecutive data values (separated by spaces) go into consecutive memory units the .X parameter specifies (the default is bytes.)

If the command line does not specify data, the software prompts for data, one memory unit at a time. Such prompts include the memory location and current value. To change the value, enter the new value. To advance to the next location without changing the present location, press <CR>. To exit this command, type a period (**.**) at the prompt.

NOTE

This command is only available in SDbug16.

Examples:

`>PMM 100 1 2 3 4`	Put values 1–4 into locations $100–$103.
`>PMM 200`	Start interactive memory modification.
`200 = 41 >`	Shows current value, prompts for new one.

QUIET Toggle Window Refresh

3.2.47 Toggle Window Refresh

```
QUIET
```

This command turns on the memory-based window refresh, or turns refresh off. (The default is ON).

NOTE

This command turns off the code window, the F3 and F6 memory windows, and (for the SDbug12 and SDbug32) the stack window, reducing single-stepping execution time to the minimum. The QUIET command also helps avoid memory-bus errors when the windows point to non-existent memory. (Otherwise, such errors are possible upon a processor reset.)

REM Write Remark

3.2.48 Write Remark

```
REM
```

This command lets you write a remark in the debug window. Such a remark can be helpful, for your documentation or listing, during execution of macro files or during a capture of the debug window.

Example:

```
REM your comment here
```

RESET Execute Hardware Reset

3.2.49 Execute Hardware Reset

RESET

This command does a hardware reset to the reset vector address in the MCU, then enters background debug mode.

Example:

>RESET	Reset hardware.

SERIAL Set Serial-Port Parameters

3.2.50 Set Serial-Port Parameters

```
SERIAL port baud par dbit sbit
```

where:

port	1 or 2: port com1 or port com2.
baud	Baud rate: 9600, 4800, 2400, 1200, 600, 300, 150, or 110.
par	Parity: N, E, or O (for none, even, or odd).
dbit	7 or 8: the number of data bits.
sbit	1 or 2: the number of stop bits.

This command sets serial-port parameter values, if you want to use the SDbug debug window as a dumb terminal. You must use the SERIAL command to set these before you enter the SERIALON command.

The SERIAL command does not activate serial-port functionality, but the command is a preliminary requirement for this functionality.

(After you enter the SERIAL command, you can enter the SERIALON command to start the serial-port functionality. This means that you can enter GO commands to execute instructions via the computer serial port. To end serial-port functionality, enter the SERIALOFF command.)

NOTE

The SERIAL command is for using a second serial port to directly manipulate the MCU. This does not control the SDI interface.

Example:

`>SERIAL 1 9600 N 8 1` Set serial port com1 to 9600 baud, no parity, eight data bits, and one stop bit.

SERIALOFF Deactivate Serial Port

3.2.51 Deactivate Serial Port

```
SERIALOFF
```

This command deactivates serial-port functionality. The debug window returns to its normal SDbug role; program execution no longer involves the computer serial port.

(The SERIALON command activated serial-port functionality. To reactivate this functionality, enter the SERIALON command again.)

Example:

```
>SERIALOFF          End serial-port functionality; restore debug window.
```

SERIALON Activate Serial Port

3.2.52 Activate Serial Port

```
SERIALON
```

This command activates SDbug serial-port functionality. That is, entering this command makes the debug window act as a dumb terminal, so that you can enter GO commands to execute instructions via the computer serial port.

To end serial-port functionality, enter the SERIALOFF command.

NOTES

Before using the SERIALON command, you must set serial-port parameters via the SERIAL command.

While serial-port functionality is activated, pressing the F1 key terminates program execution.

The SERIAL command is for using a second serial port to directly manipulate the MCU. This does not control the SDI interface.

Example:

`>SERIALON`	Start serial-port functionality: make debug window a dumb terminal.

SFC Set MCU SFC Value

3.2.53 Set MCU SFC Value

```
SFC [n]
```

where:

n Source function code value.

This command sets the source function code (SFC) to the specified n value. This new value is valid only while the processor is executing. If you are in background debug mode, use the DEBUGGER_SFC command, not the SFC command, to set the SFC. SDbug uses only the three least significant bits of the n value.

If you enter this command without an n value, the system displays the current DFC value.

Examples:

`>SFC 4`	Set SFC value to $4.
`>SFC`	Show current SFC value.

SNAPSHOT Screen Capture

3.2.54 Screen Capture

SNAPSHOT

This command captures a screen, sending the screen to an open capture file.

SOURCE Toggle Code Display

3.2.55 Toggle Code Display

```
SOURCE
```

This command toggles the debug-window display between source code and disassembled code. A valid map file must be loaded for this command to work.

NOTE

Your program counter must be set to the correct value before the source command will show the source code.

Example:

`>LOADMAP myfile3`	Load map file `myfile3.map` into the SDbug.
`>SOURCE`	Toggle debug window display to source code (or to disassembled code).

SOURCEPATH Code Search

3.2.56 Code Search

```
SOURCEPATH [path]
```

where:

path Sets the default search directory.

This command starts a search for source code not in the current directory. The system prompts for the DOS path.

NOTE

The SOURCEPATH command pertains only to source-code debugging; do not use it for such actions as downloading S-records.

Example:

```
>SOURCEPATH          Search for code in another directory.
```

SS Source Step

3.2.57 Source Step

SS

This command traces one step of source code from the IP (SDbug16) or PC (SDbug12, SDbug32) value. If your source code is in C or another high-order language, this command traces through the steps of object or executable code that correspond to that one high-order-language step. The processor does not run at full execution speed during an SS command.

Use the LST2MAP executable to convert absolute listing files to P&E debug map files.

Example:

>SS	Trace one step of the source code.

STATUS Show Processor Status

3.2.58 Show Processor Status

STATUS

This command dumps the current status of the processor to the debug window. A common use for this command is for logging such information to a capture file.

Example:

>STATUS	Show processor status in the debug window.

STEP Step Trace

3.2.59 Step Trace

```
STEP [n]
```

where:

n The number of steps to trace.

This command traces n steps of code, starting at the IP (SDbug16) or PC (SDbug12, SDbug32) value. The default n value is 1. This single-step tracing does *not* stop at breakpoints. The processor does not run at full execution speed during a STEP command.

NOTE

The STEP command traces n steps of code, even if that means halting execution in a subroutine or interrupt. If the current IP or PC points to a subroutine or interrupt, you can use the GONEXT command, which halts code execution at the first instruction *after* the subroutine or interrupt.

The two aliases of the STEP command are ST and T.

Examples:

`>STEP 10`	Trace through $10 (!16) steps.
`>ST`	Trace through 1 step.

STEPFOR Step to Breakpoint

3.2.60 Step to Breakpoint

```
STEPFOR
```

This command begins continuous step tracing from the IP (SDbug16) or PC (SDbug12, SDbug32) value. Stepping continues until it arrives at a breakpoint, until it encounters an error, or until you press a key. The processor does not run at full execution speed during a STEPFOR command.

Example:

```
>STEPFOR          Step trace from IP/PC to breakpoint.
```

STEPTIL Step to Address

3.2.61 Step to Address

```
STEPTIL add
```

where:

add Trace stop address or label.

This command starts continuous step tracing from the IP (SDbug16) or PC (SDbug12, SDbug32) value to the specified address or label. This step tracing ignores any breakpoints it may encounter before arriving at the specified stop address, but you may stop this tracing immediately by pressing any key.

Example:

```
>STEPTIL sub1          Step trace from IP/PC to label sub1.
```

SYMBOL Add Symbol to Map File

3.2.62 Add Symbol to Map File

```
SYMBOL label_name n
```

where:

label_name ASCII characters of the new label.

n The new symbol value.

This command adds the specified label to the symbol table, giving it the specified value. The symbol table contains as many as 30 user-defined labels. Creating a label with the same value as a previously defined label erases the first label.

To verify a new symbol, move to the code window, then check the disassembled code.

NOTE

The SYMBOL command does not verify the values you define. Do not use reserved labels for you assembler.

For SDbug12, do not use these reserved words as labels: **A**, **B**, **CCR**, **D**, **PC, SP**, **TMP2**, **TMP3**, **X**, or **Y**.

For SDbug16, do not use these reserved words as labels: **E**, **PC, X**, **Y**, or **Z**.

For SDbug32, do not use these reserved words as labels: **A0**, **A1**, ..., **A7**; **CCR**; **DFC**; **D0**, **D1**, ..., **D7**; **PC**; **SFC**; **SR**; or **VBR**.

Example:

```
>SYMBOL start 200
```
Define the label start = $200.

TRACE Execute Trace

3.2.63 Execute Trace

```
TRACE [add] [add]
```

where:

add First parameter: trace starting address or label;
Second parameter: trace stop address or label.

This command starts and stops tracing according to the specified add parameters:

- If the command has two parameter values, the system sets a temporary breakpoint at the second address or label, then traces code from the first address or label. Tracing continues until it arrives at the temporary breakpoint just set, until it arrives at an existing (permanent) breakpoint, or until you press a key.
- If the command has one parameter value, the system traces code from that address or label. Tracing continues until it arrives at an existing breakpoint, or until you press a key.
- If the command has no parameter values, the system traces code from the IP (SDbug16) or PC (SDbug12, SDbug32) value. Tracing continues until it arrives at an existing breakpoint, or until you press a key.

Tracing yields a log of CPU execution. The system stores this log in the trace buffer, so does not execute in real time. Paragraph 3.5 gives more information about the trace buffer.

When tracing begins, the system blanks all windows. At the end of tracing, press the F7 key to convert the code window to the trace window. The trace window shows the contents of the trace buffer, allowing you to scroll through the executed code. (To convert the trace window back to the code window, press the F2 key.)

Examples:

`>TRACE 200 1050`	Start code trace at address $200; break at address $1050.
`>TRACE 1050`	Start code trace at address $1050.
`>TRACE start`	Start code trace at label start.
`>TRACE`	Start code trace at IP/PC value.

UPLOAD_SREC Display S-Record Files

3.2.64 Display S-Record Files

```
UPLOAD_SREC add add
```

where:

add First parameter: starting address of memory range;
Second parameter: ending address of memory range.

This command displays the contents of the specified memory range, in S-record format. If the capture feature is active, SDbug also saves the data to the capture file.

Example:

```
>UPLOAD_SREC 0 5
Uploading S records, Press any key to abort.
S10900004E714E714E71B9
done.
```

VAR Show Variables Window

3.2.65 Show Variables Window

```
VAR[.X] add [n]
```

where:

.X	Value formats: .B or .b = byte (the default), .W or .w = word, .L or .l = long word, .S or .s = string.
add	Program space address or label that contains a data value.
n	Length of a string value (default is 18).

This command activates the variables window, which shows any variables of the CPU data space. The variables window replaces the F6 memory window; just as with the F6 memory window, you can scroll through variable-window values. If you move to a different window of the main screen, pressing the F6 key moves you back to the variables window.

Each time you enter a new add parameter value (via a new VAR command), you add the corresponding value to the variables window. The system automatically updates these values each time you enter an execution command (such as COUNT, GO, or STEP). To delete a variable from the window, scroll to the value, then press the delete key.

The variables window remains part of the main screen until you enter the MDF6 or SHOWF6 command. Either of these commands reactivates the F6 memory window.

The contents and format of the variables window remain as set by the most recent VAR command. For example, entering the VAR command with an .X parameter value activates the variables window with values in the specified format. (If the variables window already is activated, the new command changes the format of the values.) Entering the VAR command without an .X parameter value also activates the variables window, but with the value format unchanged.

Examples:

`>VAR.W totalval`	Show two-byte quality at location `totalval`.
`>VAR.S mystring 8`	Show eight-character string at location `mystring`.

VERIFY Compare File to Memory

3.2.66 Compare File to Memory

```
VERIFY [file]
```

where:

file The name of the file to verify against memory contents.

This command compares contents of an S-record file with contents of program memory. When you enter the VERIFY command, the system prompts for a file name. If the file is not in the current directory, enter the entire DOS path.

Verification ends the first time the system finds memory locations whose values do not match. In such a case, the address of the discrepancy appears, then the system returns to the > prompt. If memory contents match perfectly, a confirmation message appears before the system returns to the > prompt.

Example:

```
>VERIFY          Verify contents of file and memory.
```

VERSION Display the Current Software Version

3.2.67 Display the Current Software Version

```
VERSION
```

This command displays the current software version of SDbug and SDI firmware in the command window.

Example:

`>VERSION`	Display the current software version of SDbug and the SDI firmaware.

WATCHDOG Disable Watchdog Timer

3.2.68 Disable Watchdog Timer

```
WATCHDOG
```

This command disables the watchdog timer. You may take this action only once, following a hardware reset. (For many processors, a reset enables the watchdog timer.)

NOTE

This command is not available in SDbug12.

Example:

```
>WATCHDOG          Disable watchdog timer.
```

WHEREIS Show Symbol Value

3.2.69 Show Symbol Value

```
WHEREIS label_name
```

where:

label_name Label or address.

This command echoes a specified label and displays its address. Alternatively, this command echoes a specified address and displays the label at that address.

Examples:

`>WHEREIS start`	Show address of label start.
`start = 200`	System echoes label and shows address.
`>WHEREIS 200`	Show label at address $200.
`start = 200`	System echoes address and shows label.

3.3 SOURCE-LEVEL DEBUGGING

Entering the LOADALL or LOADMAP command loads a map file into the SDbug, enabling source-level debugging. If the IP (SDbug16) or PC (SDbug12, SDbug32) points to a code location, the code that appears in the code window is your actual source code. (If no map file is loaded, or if the IP/PC points outside source code, the code window shows disassembled code.)

Create valid map files via IASM or for other high-level absolute listings (such as C code) use LST2MAP, per the information in Chapter 4

The code window does not permit editing, but does let you set and remove breakpoints. The PC and IP (SDbug16) or PC (SDbug12, SDbug32) values are highlighted in the code window. To maneuver within the code window, use the keys or key combinations of Table 3-2. (Hold down the ALT key to see a list of code-window commands at the bottom of the screen. When you release the ALT key, the bottom line reverts to its normal display.)

Note that you may set breakpoints in any module. (In this context, module means your main IASM source code or any include file.)

Even if the module in the code window has no breakpoints, the breakpoints remain active in other modules. If a break occurs in another such module, the module at breakpoint replaces the current module in the display.

NOTES

When appropriate, source-code debugging automatically reverts to symbolic disassembly. Typically, this is because the code displayed is not in the map file. Code in pseudo ROM from another source is an example.

If you use ASM, PMM (SDbug16), or another command to modify code memory, the code window does not reflect your changes in source mode. The code window does show your changes in disassembly mode.

Table 3-2. Source Window Commands

Keys(1)	Meaning
Alt–B	Sets a breakpoint at the highlighted line or removes a breakpoint from the highlighted line (like the BR command)
Alt–C	Sets a counter at the highlighted line or removes a counter from the highlighted line (like the COUNTER command)
Alt–F	Prompts for and finds a search string
Alt–G	Executes code from the IP/PC to the highlighted line (like the GOTIL command)
Alt–I/Alt–P	Sets the IP/PC to the location of the highlighted line
Alt–L	Finds the next occurrence after Alt–F
Alt–M	Lists available source-code modules (press <CR> or ESC to select one)
Alt–X	Exits SDbug, returning to DOS.
arrow keys	Scrolls through code window or maneuvers cursor in code window
<CR>	Selects a source-code module (from list displayed via the Alt–M keys)
<ESC>	Cancels a request for a source-file module
(1) Hold down the ALT key to see a list of code-window commands at the bottom of the screen. When you release the ALT key, the bottom line reverts to its normal display.	

3.4 SDBUG MACROS

SDbug macros can automate the debugging process. A macro is a file of SDbug commands, all executed by using the MACRO command. To define a macro, enter the MACROSTART command, specifying a name. (The default system extension is **.ICD**.) Then, enter the commands of the macro. Any command you can enter in the debug window can be in a macro. When you complete your macro, enter the command MACROEND; this command ends the macro definition and stores the macro file. (Alternatively, you can use any general-purpose text editor to define a macro. The macro must be a plain ASCII file with one command per line.)

To call a macro, enter the MACRO command, specifying the macro filename in the command. Macro execution begins immediately.

To see a list of all macro files, enter the MAC command; the list appears, in a temporary window. To call a macro directly from this list, scroll to highlight the macro name, then press <CR>. To close the temporary window without executing a macro, press <ESC>.

3.5 TRACE BUFFER

The trace buffer is a circular, 1024K buffer that supports the SDbug trace functionality. When you enter the TRACE command, the system begins execution of the specified instructions, logging in the trace buffer each instruction the CPU executes.

During tracing, the system disables screen refreshes, so tracing happens as rapidly as possible. The only information requested from the CPU between steps is the IP (SDbug16) or the PC (SDbug12, SDbug32) value.

To view the competed trace, press the **F7** key. The code-window display changes, to show a disassembly of the addresses in the trace buffer. Use the arrow keys to scroll through the buffer.

NOTES

The trace buffer shows disassembled code, not source code, using labels whenever possible.

Tracing through self-modifying code does not work properly, as the trace shows disassembly of the current memory contents.

CHAPTER 4
IASM OPERATING PROCEDURE

4.1 INTRODUCTION

IASM development software is an editor, cross-assembler, and communications package from P&E Microcomputer Systems. IASM12 is for M68HC12 MCUs, IASM16 is for M68HC16 MCUs, and IASM32 is for MC68300 MCUs. The three applications of each package are blended into a single environment for writing, assembling, and debugging source code. Via SDbug software, you can correct source code syntactical errors without leaving the environment. The IASM integrated editor is a standard source file editor. The integrated cross assembler is optimized for the IASM environment. The built-in communication environment lets you work with any development system attached to your personal computer. You may download and test assembled files during the editing and assembly of original source code.

The symbol **<CR>** indicates the ENTER, RETURN, or carriage-return key of your keyboard.

4.1.1 System Requirements

IASM runs under MS-DOS on an IBM PC compatible computer. There should be at least 512 kilobytes of system memory. Communications use ports COM1 or COM2. To print from within IASM, use the standard DOS printer port.

4.1.2 System Overview

IASM lets you generate standard assembly-language source code or read source code in from a disk file. You may generate three kinds of output files:

- Object files – machine language for the target processor.
- Listing files – copies of input text with machine-code, cycle-timing, and other such annotations.
- Map files – files used by other P&E Microcomputer software, such as simulators, user interfaces, and the SDbug.

The options you choose determine which types of files IASM generates. If the system finds an error during assembly of a file, the system highlights the line that contains the error, places the cursor on this line, and alerts you.

4.1.3 Getting Started

Before starting IASM software, read the README file (if any) of your software diskette and read the software release guide (if any). The README file or a software release guide contains information not available at press time.

To start the environment, type the command line:

```
>IASM# [filename]
```

where

#	is the appropriate IASM number: 12, 16, or 32
filename	is the optional filename to be loaded into the editor immediately.

Note that the editor gives all file names the extension **.ASM** as a default. If you do not enter a file name, the editor starts with the blank file `Noname.asm`; when you save this file, the editor prompts you to change its name.

4.2 IASMINST CONFIGURATION

IASMINST.EXE is a configuration program for:

1. Changing editor commands.
2. Choosing a color scheme.
3. Setting up assembler parameters.
4. Setting up communications parameters.

To run this program, enter:

```
>IASMINST IASM#
```

where # is the appropriate IASM number: 12, 16, or 32.

First, the configuration program lets you modify certain editor parameters. In most cases, pressing <CR> accepts the default value shown. The program asks for a DOS string to be executed when you press F6. The F6 key is a dedicated shell to DOS; you may leave its definition blank if you wish. (It is possible to shell directly from IASM to SDbug, provided that you have at least 640 kilobytes of memory.)

Next, the configuration program prompts for default settings of all items in the ASSEMBLE sub-menu. Table 4-2 lists menu and sub-menu items; paragraph 4.7 includes explanations of the ASSEMBLE sub-menu items.

Then the configuration program prompts for default settings of all items in the COMM sub-menu. Paragraph 4.9 includes explanations of the COMM sub-menu items.

You may quit the configuration program at this point, if you have set all appropriate parameter values. Continue this program to configure colors or shading for the current monitor. To select an attribute, use the up or down arrow keys to highlight the attribute, then press <CR>. Then select the color for the attribute in the same way. When you are done selecting colors, press ESC.

Lastly, the configuration program asks if you want fast entry mode or random access mode for reconfiguring editor commands:

- If you choose fast entry, follow the on-line instructions. Commands appear one at a time. To leave the command as it is, press <CR>. To change the command, use the backspace key. To clear the command, press the C key. To restore the command, press the R key. You also may specify new keys for the command: press <CR> to terminate the new key sequence.(1) To leave fast entry mode and go to random access mode, press ESC.
- Random access mode lets you alter commands in any order. Instructions for this mode appear at the top of the screen. Use the cursor keys to select a command to be changed. The editing keys are the same as for fast entry mode.

To terminate the configuration program, press the ESC key, then either the Q key or the W key:

- Q terminates IASMINST.EXE, accepting all commands as they stand.
- W checks for conflicts among command definitions. Error messages point out the same sequence assigned to two commands; look for the highlighted items to correct such errors. Other error messages point out lists of keystrokes that are too long; to correct these errors, eliminate unnecessary key sequences.

NOTE

Generating a listing, option, or map file increases the time necessary for assembling a program. For efficient operation, set the listing, option, and debug-map defaults to OFF. Turn the options on only when you need them.

(1). To make <CR> part of the new key sequence, press the scroll-lock key, then <CR>, then press the scroll-lock key again.

4.3 HOTKEYS

Hotkey labels appear at the bottom of the screen. Table 4-1 explains the functions of these keys.

Table 4-1. IASM Hotkeys

Key	Name	Description
F1	Help	Brings up the help system.
F2	Save	Saves the file currently in the editor, makes a backup file, and returns the cursor to its position before you pressed this key.
F3	Load	Loads a new file. If you have changed the current file, prompts you to save the file, then asks for the name of the file to be loaded. (<CTRL> F3 loads the file specified in the command line when you entered IASM.)
F4	Assemble	Assembles the file currently in the editor; any options chosen from the main menu system will be in effect. (Note that only one window may be open during assembly.)
F5	Exit	Ends the editing–assembling session. You may save any changes to the current file before returning to DOS. If you are in a secondary window, this key closes the window currently active.
F7	Comm	Opens the communications window. Or, if this window already is open, makes it the active window.
F9	DOS shell	Puts you into DOS. (Typing EXIT at the DOS prompt returns you to IASM.)
F10	Menu	Brings up the main menu system on the bottom line of the screen.

4.4 MENU

The IASM menu system contains a highlighted bar along the bottom of the screen. To choose a menu item, highlight it via the cursor keys or enter the highlighted letter within your choice (for example, E in Edit). To go back to a previous menu or return to the editor, press the ESC key. Table 4-2 explains menu and submenu choices.

Table 4-2. IASM Menus

Menu	Submenu	Choice
Edit	(none)	Returns you to the editor.
File	**L**oad	Loads a new file. If you have changed the current file, prompts you to save the file, then asks for the name of the file to be loaded.
	Save	Saves the file currently in the editor, makes a backup file, and returns the cursor to its position before you selected Save. The backup filename includes the .BAK extension.
	Dir	Shows the directory listing of all files that fit a specific pattern. (Enter the pattern at the prompt or press <CR> to accept the default pattern. The original default pattern is *.asm, which specifies all files of the current directory that have the extension .ASM.) To load a file from the listing, position the cursor on the filename and press <CR>.
	Quit	Exits IASM.
Communicate	**C**omm	Toggles between operating system hardware ports COM1 and COM2.
	Baud	Steps through possible baud rates, 110 through 9600.
	Parity	Steps through N, E, and O, indicating none, even, or odd parity.
	Length	Toggles between 7 and 8, indicating the number of bits to use per character.
	Stop	Toggles between 1 and 2, indicating the number of stop bits to use per character.

Table 4-2. IASM Menus (continued)

Menu	Submenu	Choice
Assemble	**A**ssemble	Assembles the file currently in the editor; any options chosen from this submenu will be in effect. (The target system must be attached and powered up. Note that only one window may be open during assembly.)
	Object	Steps through S19, HEX, and OFF, indicating whether to create an object file during assembly and the type of object file. Choose the type appropriate for your development system or PROM programmer. (Creating an object file increases assembly time.)
	Listing	Toggles between ON and OFF, indicating whether to create a listing (.LST) file during assembly. (Creating a listing file increases assembly time.)
	Debug Map	Toggles between ON and OFF, indicating whether to create a map (.MAP) file during assembly. (Creating a map file increases assembly time.)
	Cycle Cntr (IASM12, IASM16)	Toggles between ON and OFF, indicating whether to include the base instruction cycle counts in the listing file.
	Save on F6 (IASM32 only)	Toggles between ON and OFF, controlling the prompt to save the file before shelling to SDbug32.
	Macro	Toggles between VIEW and HIDE. VIEW means print macro source code in the .LST file at every macro call. HIDE means suppress such printing.
	Include	Toggles between VIEW and HIDE. VIEW means print INCLUDE-file source code in the .LST file. HIDE means suppress such printing.
Help	(none)	Brings up the help menu.

4.5 HELP

To bring up the menu-driven help system, press the F1 key from within the editor or press the H key from the main menu.

The help system covers the editor and all original commands, as well as the assembler, including assembler commands, options, and structures.

The initial window shows a list of topics. There may be more topics than fit in the window; if so, use the up and down arrow keys to scroll through the topics. To choose a topic, use the cursor keys to move the highlight bar over the topic, then press <CR>. The first help page appears. Press the page-up or page-down key to see next or previous pages. Press the ESC key to exit to the previous help window. If you are in the first help window, pressing the ESC key exits the help system.

4.6 EDITOR

The IASM editor lets you type in text via the keyboard. New text starts at the cursor position. The editor includes several block commands for moving and copying text, several delete commands for correcting mistakes, and find and find-and-replace commands for changing text. In many cases you can undo your last several commands via the restore line command.

Paragraphs 4.6.1 through 4.6.8 explain the various editor commands.

4.6.1 The Editing Screen

The top line of the editing screen is a prompt line. This line displays messages, instructions, and responses to prompts. When you enter a two-key command, the editor echoes the first key at the left edge of the prompt line.

The top line of any editing window is a status line. Table 4-3 lists status-line information.

Table 4-3. Edit Window Status Line Information

Item	Role or Description
FILENAME.EXT	Name and extension of the file being edited. (You may specify full path names to the editor, but this item shows only name and extension.)
Line n	File line-number position of the cursor.
Col n	File column-number position of the cursor.
Byte n	Byte-number position of the cursor, relative to the first character in the file.
Insert	Indicates that the editor is in insert mode. (Press the Insert key to toggle between insert and overwrite modes).
Indent	Indicates that the editor is in auto-indent mode.
Tab	Indicates that the editor is using fixed tab stops.
Save	Indicates that the file has been modified since it last was saved.

4.6.2 Prompt Editor

Most IASM user-response prompts include default responses. To accept a default response, press <CR>. If you enter a specific response on the prompt line, you use the prompt editor. The prompt editor has the same commands as the full IASM editor, plus these three:

1. Accept entry <CR> or <CTRL>M
2. Abort ESC
3. Insert control character <CTRL>P

4.6.3 Tabs

The IASM editor has two kinds of tabs:

- **Smart tabs** – These tabs echo the appearance of the preceding line. The first character of any non-space sequence acts as a tab stop for the next line. The smart-tab mode usually is the easiest tab mode for entering source code.
- **Fixed tabs** – You may specify fixed tab stops when you run the installation program. The default stops are column 9 and every 8 columns. Note that the first tab stop always is in column (tab size + 1).

For either smart or fixed tabs, the editor automatically translates tabs to spaces. When a file is read into the editor, all tabs are expanded to the default settings. When a file is written, the editor encodes spaces into tabs if this option is enabled via the installation program. This saves disk space but slows the writing of files.

4.6.4 Window Commands

As many as five windows may be open at any time. But during assembly, only one window may be open and this window must contain the file to be assembled.

Use the ALT key to access window commands. When you hold down the ALT key, the help line at the bottom of the display shows the command options. Table 4-4 explains the window commands.

Table 4-4. IASM Edit Window Commands

Command	Keys	Description
Add window	ALT-F3	Opens another text window and prompts for a file to edit. If you do not specify a file, the editor creates the file NONAME.ASM, which you may save later as a named file. If 5 windows already are open, an error message appears. If the active window is too small to divide in half (to make room for the new window), an error message appears.
Close window	ALT-X	If two or more windows are open, closes the current window. If one window is open, closes the window and leaves IASM.
Next window	ALT-F1	Makes the next text window the current window.
Previous window	ALT-F2	Makes the previous text window the current window.
Resize window	ALT-F4	Lets you change the size of the current window via the up- and down-arrow keys. To return to the editor, press <CR> or ESC.

4.6.5 Cursor Commands

There are two ways to move the screen cursor: via the cursor control keys or via control characters. To define or modify either method, run the installation program. Table 4-5 explains the cursor commands.

Table 4-5. IASM Cursor Commands

Command	Keys	Description
Beginning of file	<CTRL>PgUp or <CTRL>QR	Moves the cursor to the first character of the file.
Beginning of line	Home or <CTRL>QS	Moves the cursor to column 1 of the current line.
Bottom of block	<CTRL>QK	Moves the cursor to the block-end marker set via the <CTRL>KK command. This command works even if there is no block-begin marker or if the block is hidden.
Bottom of screen	<CTRL>End or <CTRL>QX	Moves the cursor to the last line on the screen.
Character left	Left arrow or <CTRL>S	Moves the cursor one character to the left.
Character right	Right arrow or <CTRL>D	Moves the cursor one character to the right.
End of file	<CTRL>PgDn or <CTRL>QC	Moves the cursor just beyond the end of the file.
End of line	End or <CTRL>QD	Moves the cursor to the end of the current line and removes trailing blanks.
Go to column	<CTRL>JC	Moves the cursor to the column you enter, 1–120. The system shows the current column number; overwrite the number to go directly to the new column. Alternatively, precede the value with + or - to offset from the current column.
Go to line	<CTRL>JL	Moves the cursor to the line you enter, 1–32,767. The system shows the current line number; overwrite the number to go directly to the new line (or end of file). Alternatively, precede the value with + or - to offset from the current line.

Table 4-5. IASM Cursor Commands (continued)

Command	Keys	Description
Jump to marker 0 .. 9	<CTRL>Q0 .. <CTRL>Q9	Moves the cursor to one of nine previously set invisible markers.
Line down	↓ or <CTRL>X	Moves the cursor down one line; may scroll the screen.
Line up	↑ or <CTRL>E	Moves the cursor up one line; may scroll the screen.
Page down	PgDn or <CTRL>C	Moves the cursor down one page, with a single line of overlap.
Page up	PgUp or <CTRL>R	Moves the cursor up one page, with a single line of overlap.
Previous cursor position	<CTRL>QP	Moves the cursor to its previous position; very useful after a save, find, or find-and-replace.
Scroll down	<CTRL>Z	Scrolls the screen down one line. The cursor does not change lines until it hits the top of the screen.
Scroll up	<CTRL>W	Scrolls the screen up one line. The cursor does not change lines until it hits the bottom of the screen.
Set marker 0 .. 9	<CTRL>K0 .. <CTRL>K9	Sets one of nine invisible markers at the current cursor position.
Top of block	<CTRL>QB	Moves the cursor to the block-begin marker set via the <CTRL>KB command. This command works even if there is no block-end marker or if the block is hidden.
Top of screen	<CTRL>Home or <CTRL>QE	Moves the cursor to the top line on the screen.
Word left	<CTRL>← or <CTRL>A	Moves the cursor to the beginning of the word to the left; may move across a line break.
Word right	<CTRL>→ or <CTRL>F	Moves the cursor to the beginning of the word to the right; may move across a line break.

4.6.6 Insert and Delete Commands

Table 4-6 explains the commands for inserting and deleting characters, words, and lines.

Table 4-6. IASM Insert and Delete Commands

Command	Keys	Description
Delete character left	Backspace or <CTRL>H	Moves the cursor left one position, deleting the character at that position. Following characters of the line also move left one position. If the cursor starts in column 1, this command joins the line to the preceding one.
Delete current character	Del or <CTRL>G	Deletes the character at the cursor position. Following characters of the line move left one position. (This command does not cross line breaks.)
Delete line	<CTRL>Y	Deletes the line containing the cursor. Following lines move up one, and the cursor moves to column 1 of the next line. (A line deleted via this command cannot be restored.)
Delete to end of line	<CTRL>QY	Deletes all characters from the cursor position to the end of the line.
Delete word	<CTRL>T	Deletes the word to the right of the cursor. This command works across line breaks, and can be used to remove line breaks.
Insert control character	<CTRL>P	Lets you insert editor control characters in the text. For example, <CTRL>P followed by <CTRL>G inserts <CTRL>G (the bell character). The editor displays control characters as upper-case, highlighted letters. (The assembler does not accept control characters.)
Insert line	<CTRL>N	Inserts a line break at the cursor position. The cursor remains at that position.
New line	<CR> or <CTRL>M	In insert mode, inserts a line break at the cursor position. In autoindent mode, the cursor moves to the next line, either to the column of the first non-blank character in the current line. or to column 1. In overwrite mode, the cursor moves to column 1 of the next line without inserting a new line.
Tab	Tab or <CTRL>I	In insert mode, moves the cursor and following text right to the next tab stop. In overwrite mode, moves only the cursor right to the next tab stop. The extent of movement depends on the kind of tabs (fixed or smart) in use.

4.6.7 Block Commands

A block is any defined, contiguous stream of text, from a single character, to many lines–even an entire file. To define a block, put a block-begin marker at the first character and a block-end marker after the last. Once you define a block in this way, you can move it, copy it, delete it, or write it to a file.

The editor highlights defined blocks, but you may change this display via the hide-block command. Block commands work only with non-hidden, fully defined blocks. Table 4-7 explains the block commands.

Table 4-7. IASM Block Commands

Command	Keys	Description
Begin block	<CTRL>KB	Sets the invisible block-begin marker, so you can return the cursor to the position at any time via the <CTRL>QB command. If a block-end marker already is set, this command also highlights the block.
Copy block	<CTRL>KC	Copies a marked and displayed block, placing the copy at the cursor position. Markers move to the copied block; the original block is not affected.
Delete block	<CTRL>KY	Deletes a marked and displayed block. The undo last deletion (<CTRL>QU) command usually can restore portions of a block accidentally deleted, but there is no command to restore a deleted block completely.
End block	<CTRL>KK or F8	Sets the invisible block-end marker, so you can return the cursor to the position at any time via the <CTRL>QK command. If a block-begin marker already is set, this command also highlights the block.
Hide block	<CTRL>KH	Turns highlighting on or off, for a displayed block, without affecting the markers.
Mark single word	<CTRL>KT	Marks as a block the word that contains the cursor or the word to the left of the cursor. (This single command puts block-begin and block-end markers around the word.)
Move block	<CTRL>KV	Moves a marked and displayed block to the cursor position. Markers remain with the block.
Print block	<CTRL>KP	Prints the selected block. To cancel this command, press the ESC key.

Table 4-7. IASM Block Commands (continued)

Command	Keys	Description
Read block from file	<CTRL>KR	Reads an entire file into the text stream at the current cursor position, marking the file as a block. The editor prompts for a filename; if you already have used this command, the prompt includes the previous filename. Press <CR> to accept the previous filename, change the previous filename via the backspace key, or enter a new name. The filename may include a drive or path identifiers. To cancel this command, press the ESC key.
Write block to file	<CTRL>KW	Copies a marked and displayed block to a file, without changing the block or its markers. The editor prompts for a filename; do not use the .BAK extension, which is reserved for editor backup files. If the filename exists, another prompt asks whether to overwrite the file. If you respond no (N), you can enter a new filename, change the displayed filename via the backspace key, or cancel the command via the ESC key. This command has no effect if no block is specified.

4.6.8 Miscellaneous Commands

Table 4-8 explains the remaining editor commands.

Table 4-8. IASM Miscellaneous Commands

Command	Keys	Description
Abort	ESC	Halts an operation in progress. The editor regularly checks the keyboard buffer for the abort command. If it finds one, the editor empties the buffer and stops the operation.
Exit editor	ALT-X	Exits to DOS. If an editor file has been modified, a prompt asks whether to save the file. If you respond yes, the editor saves the file before exiting to DOS.
Find	<CTRL>QF	Searches for a string as long as 67 characters globally, backwards, within the current block, or ignoring case. See text immediately following this table for more details of the find command.
Find and replace	<CTRL>QA	Searches for a string and replaces it with another string. See text immediately following this table for more details of the find-and-replace command.
Find next	<CTRL>L	Repeats the last find or find-and-replace command.
Restore line	<CTRL>QL	Undoes any changes to the line that contains the cursor. If the cursor has left the line, this command does not work.
Save to file	<CTRL>KN	Prompts for a file name, then saves the file in the current window to the specified file. This becomes the new file in the current window. (This is particularly useful for NONAME.ASM files.)
Set undo limit	<CTRL>JU	Sets the size of the undo buffer, which stores deleted lines. The default value is 40 lines (but you may change the default via the installation program).
Show available memory	<CTRL>JR	Shows the amount of RAM available to IASM.
Show version	<CTRL>JV	Displays the current version of IASM.

Table 4-8. IASM Miscellaneous Commands (continued)

Command	Keys	Description
Toggle autoindent	<CTRL>QI	Enables or disables autoindent. When autoindent is enabled, <CR> or <CTRL>M jumps to the next line, to the column of the first non-blank character of the current line. Indent shows on the status line
Toggle fixed tabs	<CTRL>QT	Enables fixed tabs or smart tabs. When fixed tabs are enabled, Tab shows on the status line.
Toggle insert mode	Insert or <CTRL>V	Enables insert mode or overwrite mode. In insert mode, existing text moves right as new text is entered, and Insert shows in the status line. In overwrite mode, new text replaces existing text.
Undo last deletion	<CTRL>QU	Restores lines deleted via the delete or delete-line command. This command does not restore single characters or words. (To undo changes to the current line, use the restore-line command. To specify the size of the undo buffer, use the set-undo-limit command.)

4.6.8.1 The Find Command

The find command, as well as the find-and-replace command, needs additional explanation.

When you enter the find command, the status line clears and a prompt asks for the search string (as long as 67 characters). If you have used this command before, the prompt includes the most recent search string. To select the same search string, press <CR>. To edit or replace the search string, use these commands:

Backspace Deletes the character to the left

<CTRL>R Restores the previous string

<CTRL>S Moves cursor left

<CTRL>D Moves cursor right

ESC Cancels the command

<CTRL>P Enters a control character

After you enter the search string, a prompt asks for options. (The editor displays any options of the most recent search; you may use them again or edit them.) Search options are:

B Backwards search from cursor position

G Global search from start of file (or from end of file for a backwards search)

L Search only currently marked block

U Treat all characters as upper case

W Search for whole word only (ignore the target string if it is part of a longer word)

After you specify options, the search begins. If a matching pattern is found, the cursor appears at the end of the pattern. When you enter the find command with no option at the search string prompt, the search continues from the current cursor position.

The find next command uses the same parameters as the find command, but if you specify a global (G) option, the search restarts from the beginning and stops on the first occurrence. This means you will never get beyond the first occurrence of your string search.

4.6.8.2 The Find-and-Replace Command

This command is similar to the find command. However, after you specify the search string, this command prompts for a replacement string. Like the search string, the replacement string may be as long as 67 characters. The prompt includes the replacement string (if any) from a previous use of this command. You may accept, edit, or replace the replacement string, just as you can the search string.

After you specify the search and replacement strings, the option prompt appears. All the find-command options are available, plus one more:

N Replace without a prompt

After you specify options, the search begins. If the search finds a matching pattern and the N option is not in effect, a prompt requests confirmation that you want the replacement. Respond **Y** (replace), **N** (skip), or **A** (replace this and all subsequent matches without prompts).

If you specify the N option, replacement of the search string happens without any confirmation prompts. The screen does not update until all file updates are done.

To abort a find-and-replace operation, press Q or ESC.

4.7 ASSEMBLER

The assembler assembles the file currently in the editor. The assembler produces object, map, or listing files, according to the options the user chooses. (Optionally, the assembler only checks syntax, without producing any of these files.) The source file uses factory standard mnemonics. See the environment's help screens for a list of acceptable mnemonics.

Each line of the source contains an assembly-language statement. Such a statement contains as many as four fields, in this order:

```
label   operation operand     ;comment
```

Paragraph 4.7.1 explains labels, paragraph 4.7.10 explains operands and operators, and paragraph 4.7.11 explains comments. Paragraphs 4.7.2 through 4.7.9 collectively explain operations, which include assembler directives, the cycle adder, conditional assembly, and macros. Paragraphs 4.7.12 through 4.7.14 explain pseudo operations, listing directives, and the listing file.

4.7.1 Labels

A label may be as long as 16 characters. A label must start with a letter, but the assembler does not differentiate between upper- and lower-case letters when searching for labels. The second and subsequent characters of a label may be letters, numerals, underscores, or dashes. You may add a colon to the end of a label, but this is optional; a space suffices. Do not use the reserved words as labels:

- **IASM12:** Do not duplicate **A**, **B**, **CCR**, **D**, **PC**, **SP**, **TMP2**, **TMP3**, **X**, or **Y**.
- **IASM16:** Do not duplicate **E**, **PC**, **X**, **Y**, or **Z**.
- **IASM32:** Do not duplicate **A0**, **A1**, ..., **A7**; **CCR**; **DFC**; **D0**, **D1**, ..., **D7**; **PC**; **SFC**; **SR**; or **VBR**.

Note that labels within macros may not be longer than 10 characters. As paragraph 4.7.7 explains, this rule lets the assembler keep such labels unique, even for multiple macro calls.

Examples of labels are:

```
Label:
ThisIsALabel:
Loop_1
This_label_is_much_too_long:
```

The assembler would truncate the last example to 16 characters. So, to the assembler, it would be the same label as:

```
This_label_is_much_longer_than_needed
```

4.7.2 Assembler Directives

Directives are keywords that control the progress and modes of the assembler. To invoke an assembler directive, use a /, #, or **$** character in column 1, then start the directive in column 2.

NOTE

To use a directive, be sure to put the /, #, or **$** character in column 1 and the first letter of the directive in column 2. This is the mandatory directive format.

Table 4-9 lists the directives. The caret (^) indicates that a parameter value must follow the directive. Note that there must be a space between a directive and a parameter value. Paragraphs 4.7.3 through 4.7.9 give more detail on how to use these directives.

Table 4-9. Assembler Directives

Directive	Action
BASE ^	Change the default input base to binary, octal, decimal, or hexadecimal
CYCLE_ADDER_ON	Start accumulating instruction cycles (IASM12, IASM16)
CYCLE_ADDER_OFF	Stop accumulating instruction cycles and print the total (IASM12, IASM16)
DPZ	Make all direct addressing relative to index register Z, to make the source compatible with an HC11 device (IASM16 only)
ELSEIF	Alternate conditional assembly vis-à-vis the IF ^ or IFNOT ^ directive.
ENDIF	End conditional assembly
IF ^	Assemble specified code if condition is true
IFNOT ^	Assemble specified code if condition is false
INCLUDE '^'	Include specified file in source code
MACRO ^	Create a macro
MACROEND	End a macro definition
NODPZ	Turn off the DPZ directive (IASM16 only)
SET ^	Set specified condition to true
SETNOT ^	Set specified condition to false

4.7.3 Changing Base

The default numerical base of the current file is hexadecimal. The BASE ^ assembler directive changes this default base to binary, octal, or decimal – or back to hexadecimal. The new base remains in effect until the end of the file, or until you use the BASE ^ directive again.

The parameter value must be in the current base, or must have a base qualifier. (Qualifier prefixes: **%** for binary, **!** for decimal, and **$** for hexadecimal. The corresponding qualifier suffixes are **Q** for binary, **O** for octal, **T** for decimal, and **H** for hexadecimal. Use either a prefix or a suffix, but not both.)

Examples are:

`$BASE 2H`	Changes default base to binary
`$BASE !8`	Changes default base to octal
`$BASE $8`	Changes default base to octal
`$BASE 10O`	Changes default base to octal
`$BASE 10T`	Changes default base to decimal
`$BASE 0AH`	Changes default base to decimal
`$BASE $10`	Changes default base to hexadecimal

NOTE

As stated above, the parameter value must be in the current base (or must have a base qualifier).

P&E Microcomputer Systems and Motorola recommend always using a qualifier with the BASE ^ directive, to make certain of setting the default base correctly. Remember that the default base at the beginning of assembly is hexadecimal.

4.7.4 Cycle Adder

The assembler contains an internal counter for instruction cycles (IASM12, IASM16): the cycle adder. The two assembler directives CYCLE_ADDER_ON and CYCLE_ADDER_OFF control this counter.

When the assembler encounters the CYCLE_ADDER_ON directive, it clears the cycle adder. The cycle adder starts a running total of instruction cycles as subsequent instructions are assembled. (For instructions that have variable numbers of instruction cycles, the cycle adder takes the smallest number.)

When the assembler encounters the CYCLE_ADDER_OFF directive, it writes the current cycle-adder value into the .LST file, and disables the cycle adder.

NOTE

For the CYCLE_ADDER_ON and CYCLE_ADDER_OFF directives to work, the Listing and Cycle Ctr options (from the Assemble submenu) must be on.

4.7.5 Conditional Assembly

The assembler lets you specify blocks of code to be assembled only upon certain conditions. To set up such conditional assembly, use the directives SET ^, SETNOT ^, IF ^, IFNOT ^, ENDIF, and ELSEIF.

The SET ^ directive sets the value of its parameter to true. The SETNOT ^ directive sets the value of its parameter to false. The maximum number of SET ^ and SETNOT ^ directives is 25.

The directives IF ^ (or IFNOT ^) and ENDIF determine the block of code for conditional assembly. Code between IF ^ and ENDIF is assembled if the parameter value is true. Code between IFNOT ^ and ENDIF is assembled if the parameter value is false.

The ELSEIF directive can precede ENDIF, providing an alternative. For example, if the parameter value is true, code between IF ^ and ELSEIF is assembled, but code between ELSEIF and ENDIF is not. If the parameter value is false, code between IF ^ and ELSEIF is not assembled, but code between ELSEIF and ENDIF is. ELSEIF gives the same alternative arrangement to a directive sequence that starts with IFNOT ^.

Example:

```
$SET debug          Sets debug value to true
$SETNOT test        Sets test value to false
   nop              Always assembles
   nop              Always assembles
$IF debug           Starts block for assembly if debug is true
   jmp start        Assembles
$ELSEIF             Starts block for assembly if debug is false
   jmp end          Does not assemble
$ENDIF              Ends block for conditional assembly
   nop              Always assembles
   nop              Always assembles
$IF test            Starts block for assembly if test is true
   jmp test         Does not assemble
$ENDIF              Ends block for conditional assembly
```

NOTE

If **start4** is a SET/SETNOT parameter value, there also can be a **start4** label or parameter value elsewhere in code. IASM treats such duplicate values as two distinct values.

4.7.6 Include

When the assembler encounters the INCLUDE '^' directive, it takes source code from the specified file. This continues until the end of the specified file or until the assembler encounters another INCLUDE '^' directive If the assembler reaches the end of the specified file, it continues taking source code from the file that contained the INCLUDE '^' directive.

The file specification must be in quotes (single or double). If the file is not in the current directory, the specification must be a full path name. (The screen window shows the full specification, provided that it does not exceed 32 characters.)

Includes may be nested to a maximum depth of 10.

The Assemble submenu lets you choose whether to show the source code from include files within the .LST file.

Examples:

```
$INCLUDE 'init.asm'
$INCLUDE "c:\project\init.asm"
```

NOTE

IASM include files become modules during source-level debugging via SDbug.

4.7.7 Macros

A macro is a named block of text to be assembled in lieu of its name. Although similar to an include file, a macro is more flexible. For example, a macro can receive parameter values.

To define a macro, enter the MACRO ^ directive; the name of the macro is the parameter value for this directive. Code on subsequent lines, to the MACROEND directive, is the macro definition.

No directives may be within a macro, nor does the macro definition need any parameter names. Instead, the definition includes the sequential indicators %n for the nth parameter values of the macro call. Your code may pass as many as nine parameter values to a macro.

Example 1: Shows a macro that divides the accumulator by 4

```
$MACRO divide_by_4      Starts macro definition
   asra                 Divides the accumulator by 2
   asra                 Divides the accumulator by 2 again
$MACROEND               Ends macro definition
```

Example 2: Shows a macro that creates a time delay

```
$MACRO delay count
   ldaa #%1
loop:  deca
   bne loop
$MACROEND
```

The name of the second macro is *delay*. Note, however, that the MACRO ^ directive line also contains the word *count*. The assembler ignores any such extra words in the MACRO ^ directive line, so you may include them to identify the parameters of the macro. In this example, *count* indicates the role of the single parameter value passed to the macro. That value is substituted for the sequential indicator *%1*.

If the calling line

```
delay 100t
```

invokes this macro, the loop occurs 100 times. (Note the **t** decimal qualifier.)

The assembler ignores extra parameter values sent to a macro. But if not enough parameter values are sent to a macro, the assembler issues an error message.

Make sure that labels within macros are no longer than 10 characters. The assembler makes certain that labels in macros always are unique by changing them each time they are used To do this, the assembler appends **:nnnn** (a four-digit hexadecimal number) to each label. Each successive time the macro is called, the assembler increments the value of **:nnnn** for the label.

Note that code cannot jump into a macro, but code may jump out of a macro. Macros cannot be forward referenced (that is, the definition of the macro must appear before a reference to the macro).

If you do not want the listing file to contain code generated during a macro, select **MACROS hide**, from the Assemble submenu. If you do want the listing file to contain this code, select **MACRO view.** Note that such code in the listing file is not identical to the macro definition: it is all upper case, and comments are stripped out. However, this appearance does not affect the definition itself.

4.7.8 Constants

Numerical constants are specific numbers entered into assembly language instructions. The default for all constants in the assembler is hexadecimal, but you may override the default by adding a qualifier prefix or suffix to a value. (Do not use both a prefix and a suffix.)

Qualifier prefixes for binary, octal, decimal, and hexadecimal are **%**, **!**, and **$**, respectively. The corresponding qualifier suffixes are **Q**, **O**, **T**, and **H**.

You may change the default base to binary, octal, or decimal – or back to hexadecimal – via the BASE ^ directive.

Note that the symbol $ or *, by itself, indicates the current program counter value.

To specify an ASCII constant or string, put it in single or double quotes.

Examples of constants are:

```
10010111Q = %10010111 = 227O = 151T = !151 = 97H = $97
JMP $                   jump to myself
JMP *                   jump to myself
db "this is a string"
LDAA '?'
```

4.7.9 Opcodes

The assembler supports all factory opcode mnemonics. To see the full list of these mnemonics, look under INSTRUCTION SET, in the on-line help system.

Opcodes cannot start in column 1. If a label starts the line, there must be at least one space (or a colon) between the label and the opcode.

4.7.10 Operands and Operators

An operand may be an address, a label, or a constant, as defined by the opcode. Arithmetic, logic, and shift operations may be performed, within parameters, during assembly.

The operators are:

- ***** multiplication
- / division
- + addition
- - subtraction or negation
- < left shift
- > right shift
- **%** remainder after division
- **&** bitwise and
- | bitwise or
- ^ bitwise xor
- ~ shift right 16

Operator precedence follows the rules of algebra; to alter this precedence, use parentheses.

NOTES

If an expression contains more than one **operator**, **parenthesis**, or **embedded space**, braces ({ }) must enclose the entire expression.

IASM software does check for a missing close brace, but cannot identify expressions that need braces. **You must use braces when required, or IASM will give you unpredictable results without giving you an error message.**

Examples of operands and operators are:

`ldab #~table`	load upper four bits of address **table** into B register
`jmp start`	start is a previously defined label
`jmp start+3`	jump to location start + 3
`jmp {start>2}`	jump to location start divided by 4

4.7.11 Comments

A semicolon (**;**) delineates comments, which may start in any column and extend to the end of the line. Additionally, if an asterisk (*) or semicolon is in column 1, the entire line is a comment.

Examples of comments are:

```
;this comment is the only thing on the line.
   nop                ;this is a comment
*this entire line is a comment
```

4.7.12 Pseudo Operations

You may use pseudo operations in place of opcode mnemonics. Table 4-10 lists these pseudo operations.

Table 4-10. Pseudo Operations

Pseudo-Op Code	Action
rmb n or **ds n**	Defines storage, reserving n bytes, where n = number or label. No forward references of n are allowed.
fcb m or **db m**	Defines byte storage, where m = label, number, or string. Strings generate ASCII code for multiple bytes; number and label parameters receive single bytes. Separate multiple parameters with commas.
fdb n or **dw n**	Defines word storage, where n = label, number, or string. Two bytes are generated for each number or label. Separate multiple parameters with commas.
lab: **equ n**	Assigns the value of the number or label n to the label lab. No forward references of n are allowed.
org n	Sets the origin to the value of the number or label n. No forward references of n are allowed.

4.7.13 Listing Directives

Listing directives are source-code keywords that control output to the listing file. These directives pertain only to viewing the source-code output; the directives, which may be interspersed anywhere in source code, do not affect the actual code assembled. Table 4-11 lists these directives. Note the character (^), which indicates a mandatory parameter value.

To invoke a listing directive, put a period (**.**) in column 1, and start the directive in column 2. The directive itself does not appear in the listing file. (If you want the directive to appear in the listing file, use /, #, or **$** in column 1, instead of the period.)

Table 4-11. Listing Directives

Directive	Action
eject or **page**	Begins a new page.
header '^'	Specified string, in quotes, will be a header on listing pages. The header can be defined only once. The default header is blank.
list	Turns on the .lst file output. (For this directive to work, the **list** choice in the **assemble** sub-menu must be on.)
nolist	Turns off the .lst file output. This directive is the counterpart of the **list** directive. At the end of a file, this directive keeps the symbol-table from being listed.
pagelength ^	Sets the length of the page, !10 – !255 lines. The default parameter value is !66.
pagewidth ^	Sets the width of the output, !40 – !255 columns, wordwrapping additional text. The default parameter value is !80.
subheader '^'	Makes the string specified in quotes a subheader on listing pages. The subheader takes effect on the next page.

4.7.14 Listing File

A listing file requested via the menu system is created during assembly. This listing file has the same name as the file being assembled, but with the extension .LST. (Any existing file that has the same name is overwritten.) The listing file has this format:

```
AAAAA VVVVVVVV [CC] LLLL Source Code .....
```

The first five hexadecimal digits (AAAAA) are the address of the instruction in the target processor memory.

The next hexadecimal digits (VVVVVVVV) are the values put into that address (and possibly the next several addresses). The actual opcode determines the size of this field.

The CC field of the format is the number of machine cycles used by the opcode (IASM12 or IASM16 only). Note that this value appears only if **Cycle Ctr** was turned on before assembly. Also note that the CC value, which always appears in brackets, is a decimal value. If an instruction has several possible cycle counts, the CC value is the lowest possibility.

The LLLL field, as many as four digits, gives the line count.

The actual source code follows the line count.

At the end of the listing file is the symbol table. This table lists each label and the value of each label.

The listing directives **list** and **nolist** affect the .LST file. If the **nolist** directive is at the end of the file, it suppresses the symbol table.

4.8 OBJECT AND MAP FILES

If an object file is requested via the menu system, it is created during assembly. The object file has the same name as the file being assembled, but with the extension .S19 or .HEX. The choice of name depends on the choice made in the assemble sub-menu. Any existing file with the same name is overwritten.

If a map file is requested via the menu system, it is generated during assembly. SDbug and P&E Microcomputer Systems products use map files during symbolic debugging.

4.9 COMMUNICATIONS

Press the F7 key to open the communications window for computer serial ports COM1 or COM2. The parameters of this window (port, baud, parity, word length, and number of stop bits) come from the options in the COMM sub-menu. (If necessary, consult your development-board manual for appropriate settings.)

Once the communications window is open, the F7 key toggles between communicating and editing.

If you change communication parameter values while the communications window is open, the new values do not take effect until you close and reopen the window. The selected baud rate and inter-character delays set in the installation program determine the speed of the download command.

Table 4-12 lists the hot keys available in the communications window.

Table 4-12. Communications Window Hot Keys

Key	Name	Description
F1	Help	Brings up the help system.
F6	Download	Prompts for a file name, then downloads the file through the appropriate serial port. This download transfers the file as it is; no special protocol is used.
F7	Edit	Moves to the editor window, closing an open view file.
F8, F9	Resize	Make the communications window larger or smaller.
F10	Close	Closes the communications window, returning control, and the full screen, to the editor (the cursor must be in the current window).

CHAPTER 5
PROGRAMMING MCUS

5.1 INTRODUCTION

PROGxxS is a software product for programming M68HC12, M68HC16, or M68300 microcontroller (MCU) devices. Use PROG12S to program any programmable memory module of an M68HC12 MCU, use PROG16S to program any programmable memory module of an M68HC16 MCU, or use PROG32S to program any programmable memory module of an M68300 MCU.

NOTE

A module is any programmable memory that is a built-in part of an MCU integrated circuit. There are several types of modules, including EEPROM, flash EEPROM, block erasable flash EEPROM, and TPU EEPROM. Modules communicate with the MCU via the intermodule bus; a given MCU can have more than one module. For more information about the memory modules of a particular MCU type, see the corresponding technical data book.

(The term module also can apply to programmable memory not part of your MCU but accessible via your target board. To use PROGS to program such an external module, you must create a custom .12P, .16P, or .32P file, which is available from P & E Microcomputer Systems.)

5.2 OVERVIEW

PROGS software resides in your host computer. To program an MCU memory module, connect your SDI™ interface between your host computer and your target system.

Next, run PROGS software. This software consists of two parts:

- General program – This program consists of the routines and general interface functions that control the erasing, programming, verifying, and viewing of programmable memory modules. This program, which pertains to any MCU programmable memory module, runs on your host computer; it communicates with your MCU in background debug mode, via the SDI cable.

- .#P files – These files, which have the extension .12P, .16P or .32P, implement the general routines and functions for specific modules. A .#P file consists of addresses, documentation information, and initialization code (in the form of S-records). The .#P files run in the target MCU processor.

The full names of .#P files relate them to a type of module and a type of MCU. There are two .#P files for each module: one that contains control-register information, and a larger array file. The names of control-register .#P files usually include the letter C; array files include the letter K. A digit following the C or K in the file name indicates that the MCU has multiple modules; the digit identifies the specific module. (The software release guide explains the complete naming convention for .#P files.)

Standard PROG.12P, PROG.16P, or PROG.32P files, which serve the needs of most users, are available in the Motorola AMCU bulletin board system MCU12, MCU16, or MCU32 areas. (To access this bulletin board, phone 1-512-891-3733. The bulletin-board serial transmission format is eight data bits, no parity, and one stop bit.)

It is possible for advanced programmers to customize .#P files; the AMCU bulletin board gives additional information about customization.

5.3 PROGRAMMING REQUIREMENTS

To program an M68HC12, M68HC16, or MC68300 MCU memory module, you need:

- PROGS software.
- An SDI interface.
- An IBM or compatible personal computer. The computer must run DOS 3.1 or higher and must have an IBM-compatible serial port.
- A target system that contains the MCU device to be programmed. This target system can be a test board, an emulator board, or a dedicated programmer board, but it must have a 10-pin Berg-type connector, to connect to the SDI interface cable. The target system must include an appropriate programming-voltage circuit or you must provide external programming voltage.
- Code to be programmed into the MCU. This code must be in Motorola S-record format.

The rest of this chapter explains how to load PROGS software, and how to use the programming commands.

5.4 STARTING PROGRAMMING

PROGS downloads S-records to the MCU device. That is, the code you program into the MCU must be in Motorola S-record format. (If you do not provide a filename extension, the system uses the default extension .S19.) To create S-records, use an M68HC12, M68HC16, or M68300 assembler (or compiler) that outputs S-records. Although you may store S-records anywhere on your computer hard drive, it is most convenient to store them in the directory that contains the PROGS software (or a subdirectory of this directory).

NOTE

To select commands in the programming software, use the cursor control keys to highlight your selection; alternatively, type one or more starting letters to highlight the command.

To activate the highlighted command, press the ENTER, RETURN, or carriage-return key. This manual uses the symbol **<CR>** for this key. Brackets (**[ ]**) denote optional parameters.

To abort a command in progress, press the ESC key. (Pressing the ESC key a second time selects the quit command.)

When you are ready to program MCUs, follow steps 1 through 10:

1. Enter the appropriate startup command, such as:

   ```
   PROG12S baud 28800 freq 4000000 sim 4
   ```

   which starts PROG12 at a communication rate of 28800 baud, for a target-system oscillator frequency of 8Mhz, with a non-multiplexed light integration module (LIM).

   Table 5-1 explains parameter values for the startup command syntax:

   ```
   PROG#S [s] [m] [baud n] [v] [freq n] [sim n]
   ```

Table 5-1. PROGS Startup Command Parameters

Parameter	Action
#	PROGS number: 12, 16, or 32.
s	Sets serial communication port (the default is com1). The range is com1 through com9.
m	Sets monitor as monochrome. For a color monitor, do not use this parameter.
baud *n*	Sets I/O port baud rate to n value. The rate range is 2400 to 57600; the default rate is 9600.
v	Deactivates S-record verification. For verification, omit this parameter.
freq *n*	Set target frequency n (half the MPB oscillator frequency), entered with all trailing zeros. The PROG12 default is 2000000 (2Mhz), the default for PROG16 or PROG32 is 8000000 (8Mhz).
sim *n*	Set SIM type value n specifies — one of these possible values: 0 SIM 4 LIM (NOMUX) 1 SCIM 5 LIM (MUX) 2 RPSCIM 6 SLIM (MUX) 3 SCIM2 7 SLIM (NOMUX) (The PROG12 default is 5; the PROG16 and PROG32 default is 0.)

The main programming screen appears, with the Choose Module command already activated. (Paragraph 5.4 explains the main screen.)

2. The choose module window appears in the center of the screen. This window contains the path to the parent directory, names of any subdirectories, then the names of all .#P files in the current directory. Use the arrow, PgUp, or PgDn keys to scroll through this window.

CAUTION

Selecting a .#P file that does not match the current MCU can lead to unpredictable programming results or prevent downloading altogether. It could even cause the loss of important data. Be sure to select a .#P file appropriate for your MCU type.

3. Select the .#P file for the memory module, then press <CR>. The choose module window disappears, and the message `Initializing` appears in the status window, at the bottom of the screen.

4. The `Base Address?` prompt appears in the base address window. Enter the hexadecimal address at which you want the module to reside, then press <CR>. (Paragraph 5.4 includes more information about base addresses.)

   The base address window disappears; the name and base address of the .#P file appear in the .PRG file selected window. Additionally, comments from the first few lines of the .#P file appear in the status window. These comments specify the MCU type and module type to which the .#P file pertains. (Such comments also identify a special user function, if one exists for the .#P file.) If somebody has altered the original .#P file, a message so informs you.

   NOTE

   If you see from the status-window comments that you have selected the wrong .#P file, merely select the CM (choose module) command to select the correct .#P file.

5. To program the module, select appropriate commands from the main screen. (Paragraph 5.6 explains each command.)
6. When you are done programming the module, proceed to one of the steps below.
7. To program another module of the same MCU, select the CM (choose module) command, then return to step 3.
8. To use the same target system to program another MCU, disconnect power from your target system, if appropriate, then remove the current MCU from the target system. Install a new MCU in the target system and restore power. Select the RE (reset chip) command, then activate the CM (choose module) command. Return to Step 3.
9. To use a different target system to program another MCU, turn off or disconnect SDI-interface power, then disconnect the SDI interface from the target system. Disconnect power from the new target system, if necessary. With SDI-interface power still off, connect the SDI interface to the new target system. Restore power to the new target system and the SDI interface. Select the RE (reset chip) command, then activate the CM (choose module) command. Return to Step 3.

   NOTE

   The SDI interface is an active system component. Altering hardware connections with interface power on can cause interface or computer-port failure.

10. To quit programming, select the QU (quit) command. This exits the programming software, returning to DOS.

5.5 PROGRAMMING SCREEN

Figure 5-1 shows the programming screen, which consists of these windows:

- **Command** – This window, at the upper left of the screen, lists and gives you access to the programming commands. (The title of this window includes the software version number.) Paragraph 5.6 explains each command.

  Before you select a .#P file, a *not active* indication is beside each command name; if this indication remains beside a command name after you select a .#P file, the command does not pertain to the particular module.

- **PRG File Selected** – This window, at the top right of the screen, shows the path, filename, and base address of the selected .#P file. (When the programming screen first appears, the word **none** is in this window.)

- **S19 File Selected** – This window, at the upper right of the screen (just below the PRG file selected window), shows the path and filename of the selected S-record file. (If you have not yet specified an S-record, the word **none** is in this window.)

- **Status** – This window extends across the bottom of the screen. Prompts, status indications, and error messages appear in this window.

Temporary windows overlay the programming screen when appropriate. The most common temporary windows are:

- **Choose Module** – This window appears at the center of the programming screen when you select the CM (choose module) command. This window contains the path to the parent directory, names of any subdirectories, then the names of all .#P files in the current directory. To scroll through this window, use the arrow, PgUp, or PgDn keys. When you select a .#P file, this window disappears.

- **Help** – The Topics help window appears near the center of the programming screen when you select the HE (help) command. When you select one of the topics, a larger help window appears, giving you the appropriate information. To exit a help window, press the ESC key.

- **Base Address** – This window appears near the right center of the screen when the system prompts for a base address. This is the MCU address you want for the module; this address should match the address of any S-record you will program into the module. PROGS also uses the base address to determine the starting and ending addresses for such commands as PM (program module), SM (show module), and VM (verify module). When you enter the address, the base address window disappears.

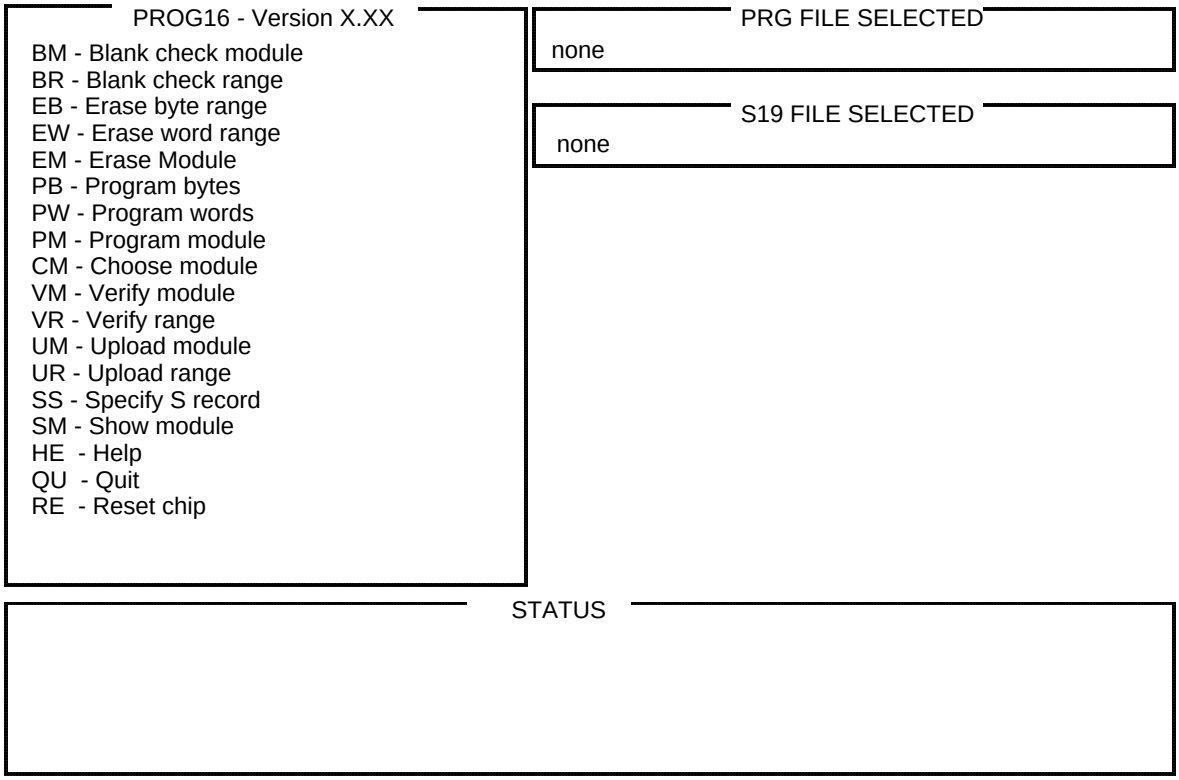


Figure 5-1. Programming Screen

5.6 SPECIAL USER FUNCTION

The command window of Figure 5-2 lists the standard programming commands: those that pertain to virtually all programmable modules. After you have selected a .#P file, look at the bottom right corner of the command window. If a down-arrow symbol is at the widow edge, the module has an extra command (or user function). Comments (from the first few lines of the .#P file) appear in the status window to identify what the user function does.

Note that a user function depends on the module; if three different modules have user functions, the three user functions may perform completely different actions.

The SDbug software release guide explains the user function, if any, for the .#P file you select. To highlight and activate the user function, you may need to scroll down a line in the command window (via the down-arrow key).

5.7 STANDARD PROGRAMMING COMMANDS

Table 5-2 lists the standard programming commands, in the order they appear in the command window. Individual explanations of each command, in alphabetical order, follow the table. Paragraph 5.7 lists a typical programming sequence. (To abort a command in progress, press the ESC key.)

Table 5-2. Standard Programming Commands

Command	Function
BM	Blank check module
BR	Blank check range
EB	Erase byte range
EW	Erase word range
EM	Erase module
PB	Program bytes
PW	Program words
PM	Program module
CM	Choose module (.#P file)
VM	Verify module
VR	Verify range
UM	Upload module
UR	Upload range
SS	Specify S-record
SM	Show module
HE	Help
QU	Quit
RE	Reset chip

BM Blank Check Module

5.7.1 Blank Check Module

```
BM
```

This command blank checks the selected memory module. That is, the system verifies that each module location is in its erased state.

If the blank check is successful, the message **`Erased`** appears in the status window. Otherwise, the status window shows the address and contents of the first unerased location.

BR Blank Check Range

5.7.2 Blank Check Range

BR

This command blank checks an MCU memory range that you specify. When you select this command, prompts ask for the hexadecimal starting and ending addresses of the range. (Both these addresses must be inside the module.) Then the system verifies that each location of the range is in its erased state.

If the blank check is successful, the message **`Erased`** appears in the status window. Otherwise, the status window shows the address and contents of the first unerased location.

NOTE

Unless the .#P file includes an erase-check-byte routine, the starting address must be a word boundary and the ending address must be a word boundary plus 1. Otherwise, the system displays a message that the .#P file does not implement byte checking.

CM Choose Module

5.7.3 Choose Module

```
CM
```

This command selects the module to be programmed and the associated .#P file. When you select this command, the choose module window appears in the center of the screen. This window contains the path to the parent directory, names of any subdirectories, then the names of all .#P files in the current directory. Use the arrow, PgUp, or PgDn keys to scroll through this window, then select the .#P file.

The choose module window disappears, and the message Initializing appears in the status window. Then the Base Address? prompt appears in the base address window. Enter the hexadecimal base address for the module, then press <CR>. The base address window disappears; the name and base address of the .#P file appear in the PRG file selected window. Additionally, comments from the first few lines of the .#P file appear in the status window. These comments specify the MCU type and module type to which the .#P file pertains. (Such comments also identify a special user function, if one exists for the .#P file.) If somebody has altered the original .#P file, a message so informs you.

CAUTION

Selecting a .#P file that does not match the current MCU can lead to unpredictable programming results or prevent downloading altogether. It could even cause the loss of important data. Be sure to select a .#P file appropriate for your MCU type. (If you see from the status-window comments that you have selected the wrong .#P file, merely select the CM command again to select the correct .#P file.)

EB Erase Byte Range

5.7.4 Erase Byte Range

EB

For specific modules, this command erases bytes of an MCU memory range that you specify. When you select this command, prompts ask for the hexadecimal starting and ending addresses of the range. When you enter these addresses (both of which must be inside the module), the system begins erasing bytes.

When erasing is done, a confirmation message appears in the status window. Should the system not be able to erase a byte, erasing activity stops and the status window shows the address and contents of the byte.

NOTE

For most EEPROM modules, it is not possible to erase individual bytes or words.

EM Erase Module

5.7.5 Erase Module

```
EM
```

This command erases the entire module. When you select this command, the system begins erasing. When erasing is done, a confirmation message appears in the status window. Should the system not be able to erase the entire module, erasing activity stops and an error message appears in the status window.

IMPORTANT NOTE

For flash EEPROM and certain other module types, the EM command erases both the data and control-register portions of the module.

EW Erase Word Range

5.7.6 Erase Word Range

```
EW
```

For specific modules, this command erases words of an MCU memory range that you specify. When you select this command, prompts ask for the hexadecimal starting and ending addresses of the range. When you enter these addresses (both of which must be word-boundary addresses, and both of which must be inside the module), the system begins erasing words.

When erasing is done, a confirmation message appears in the status window. Should the system not be able to erase a word, erasing activity stops and the status window shows the address and contents of the word.

NOTE

For most EEPROM modules, it is not possible to erase individual bytes or words.

HE Help

5.7.7 Help

HE

This command calls up help information. When you select this command, the help topics window appears near the center of the programming screen. Use the arrow and <CR> keys to select a topic. A larger help window appears, giving you the appropriate information. If there are several pages of information, use the PgDn and PgUp keys to scroll through the pages. To exit a help window, press the ESC key.

PB Program Bytes

5.7.8 Program Bytes

PB

This command programs one or more bytes of the module. When you select this command, the system prompts for a starting address. Enter the address and press <CR>; the system shows the contents of that byte and prompts for new data.

- To change the contents of the byte but hold the address constant, enter the new data and the = symbol, then press <CR>. (This lets you confirm the change in byte contents by displaying the new contents.)
- To change the contents of the byte and advance to the next byte, enter the new data and press <CR>. (Optionally, enter the + symbol after the new data, then press <CR>.)
- To advance to the next byte without changing the contents of the current byte, press <CR> (or enter the + symbol, then press <CR>).
- To change the contents of the byte and return to the previous byte, enter the new data and the - symbol, then press <CR>.
- To return to the previous byte without changing the contents of the current byte, enter the - symbol, then press <CR>.

NOTE

To continue backwards, that is, to step to the previous byte two or more times, your must use the - symbol for each step.

If you try to program beyond the end of the module array, an error message appears This message asks you to enter a period (.) to exit the PB command or a minus sign (-) to go backward. Pressing <CR> is not required in such a case.

PM Program Module

5.7.9 Program Module

```
PM
```

This command programs the selected S-record file into the module. When you enter this command, the system verifies that an S-record file is selected; if you have not yet selected one, a reminder message appears in the status window.

If you have selected an S-record file, the system checks whether all addresses of the S-record fit into the module. If so, the system programs the S-record file into the module. When programming is done, a confirmation message appears in the status window. If the system cannot program a module location, programming activity stops, and an error message in the status window identifies the location.

If all S-record file addresses do not fit into the module, a prompt asks whether to continue. If you enter Y (yes), the system programs into the module all the S-record addresses that do fit.

PW Program Words

5.7.10 Program Words

```
PW
```

This command programs one or more words of the module. When you select this command, the system prompts for a starting address. Enter the address and press <CR>; the system shows the contents of that word and prompts for new data.

- To change the contents of the word but hold the address constant, enter the new data and the = symbol, then press <CR>. (This lets you confirm the change in word contents by displaying the new contents.)
- To change the contents of the word and advance to the next word, enter the new data and press <CR>. (Optionally, enter the + symbol after the new data, then press <CR>.)
- To advance to the next word without changing the contents of the current word, press <CR> (or enter the + symbol, then press <CR>).
- To change the contents of the word and return to the previous word, enter the new data and the - symbol, then press <CR>.
- To return to the previous word without changing the contents of the current word, enter the - symbol, then press <CR>.

NOTE

To continue backwards, that is, to step to the previous word two or more times, your must use the - symbol for each step.

If you try to program beyond the end of the module array, an error message appears This message asks you to enter a period (.) to exit the PB command or a minus sign (-) to go backward. Pressing <CR> is not required in such a case.

QU Quit

5.7.11 Quit

```
QU
```

This command quits the programming software, returning you to the DOS prompt.

(Another way to select the QU command is to press the ESC key, then press <CR>.)

RE Reset Chip

5.7.12 Reset Chip

RE

This command does a hardware reset of the MCU, restoring its defined reset values. (The technical data book for the MCU lists these values.) When you select this command, the system carries out the hardware reset, then automatically highlights the CM (choose module) command.

If you program another MCU without quitting the programming software, use the RE command to reset the new MCU. Another use of this command is to recover from an error that prevents communication between the programming software and the MCU.

SM Show Module

5.7.13 Show Module

```
SM
```

This command displays module contents, 48 bytes at a time. When you select this command, a prompt shows the current default starting address. To accept this default address, press <CR>. Otherwise, type a different starting address, then press <CR>. The temporary show module window appears. This window shows module contents, starting with the specified address. The ASCII representation of each byte is at the right edge of the window.

Use the PgUp, PgDn, arrow, Home, and End keys to scroll through module contents; when you are done viewing module contents, press the ESC key.

SS Specify S-Record

5.7.14 Specify S-Record

SS

This command specifies an S-record file. When you select this command, the select S19 file window appears. Enter the file name at the prompt (including the path, as appropriate), then press <CR>. The select S19 file window disappears, and the name of the selected file appears in the S19 file selected window.

If you press <CR> at the filename prompt without entering a filename (or if you enter an invalid filename), the choose file window replaces the select S19 file window. The choose file window lists S-record files; select one just as you select a module from the choose modules window. The choose file window disappears, and the name of the selected file appears in the S19 file selected window.

NOTE

Both the select S19 file and choose file windows let you change directories as well as select a file. The directory names begin with a backslash (\ ) character.

UM Upload Module

5.7.15 Upload Module

```
UM
```

This command uploads module contents to a specified file, in S-record format. When you enter this command, the system prompts for a filename. Enter the filename and press <CR>; the system begins uploading.

If the file already exists, a prompt asks whether to overwrite it. If you enter Y (yes), the system overwrites the existing file.

NOTE

This command uploads the complete contents of MCU memory. This process can take 30 minutes or longer.

UR Upload Range

5.7.16 Upload Range

UR

This command uploads contents of an MCU memory range that you specify to a specified file, in S-record format. When you select this command, prompts ask for the hexadecimal starting and ending addresses of the range. (Both these addresses must be inside the module.) Then the system prompts for a filename. Enter the filename and press <CR>; the system begins uploading.

If the file already exists, a prompt asks whether to overwrite it. If you enter Y (yes), the system overwrites the existing file.

VM Verify Module

5.7.17 Verify Module

```
VM
```

This command compares the contents of the selected S-record to the contents of the module. When you enter this command, the system verifies that an S-record file is selected; if you have not yet selected one, a reminder message appears in the status window.

If you have selected an S-record file, the system checks whether all S-record addresses fit into the module. If so, the system verifies the contents of the S-records. When verification is done, a confirmation message appears in the status window. If the system cannot verify a module location, verification activity stops, and an error message in the status window identifies the location.

If all S-record file addresses do not fit into the module, a prompt asks whether to continue. If you enter Y (yes), the system verifies the contents of all the S-records that do fit.

This command ignores module contents at addresses the S-record does not mention.

VR Verify Range

5.7.18 Verify Range

VR

This command verifies an MCU memory range that you specify. That is, the system makes sure that contents of the range match those of the selected S-record. (If you have not yet specified an S-record, the system prompts for one.)

When you select this command, prompts ask for the hexadecimal starting and ending addresses of the range. (Both these addresses must be inside the module.) Then the system verifies that contents of each byte (or word) of the range match the contents of the corresponding S-record location.

If the verification is successful, the message **`Verified`** appears in the status window. Otherwise, the status window shows the address and contents of the first location not verified.

5.8 TYPICAL PROGRAMMING SEQUENCE

Although your specific situation determines the most appropriate programming actions, steps 1 through 11 will be typical for most users:

1. Use the **CM** command to choose a module.
2. Use the **BM** command to blank check the module. (Most modules must be blank before you can program them.)
3. If any module locations are not blank, use the **EB**, **EW**, or **EM** command to erase them.
4. Use the **SS** command to specify an S-record.
5. Use the **PM** command to program the module.
6. Use the **VM** command to verify successful programming.
7. Optionally, use the **SM** command to see module contents.
8. To program another module of the same MCU, return to Step 1, above.
9. To use the same target system to program another MCU, disconnect power from your target system, if appropriate, then remove the current MCU from the target system. Install a new MCU in the target system and restore power. Select the **RE** command, then return to Step 1.
10. To use a different target system to program another MCU, turn off or disconnect SDI power, then disconnect the SDI interface from the target system. Disconnect power from the new target system, if necessary. With SDI-interface power still off, connect the SDI interface to the new target system. Restore power to the new target system and the SDI interface. Select the **RE** command, then activate the **CM** command. Return to Step 1.

NOTE

The SDI interface is an active system component. Altering hardware connections with interface power on can cause interface or computer-port failure.

11. To quit programming, select the **QU** command to exit the programming software and return to DOS.

APPENDIX A
S-RECORD INFORMATION

A.1 INTRODUCTION

The S-record format for output modules encodes programs or data files in a printable format for transportation between computer systems. This facilitates S-record editing and permits visual monitoring of such transportation.

A.2 S-RECORD CONTENT

S-records are character strings of five fields: record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a two-character hexadecimal number: the first character represents the high-order four bits, and the second character represents the low-order four bits of the byte.

The diagram below shows the S-record layout. Table A-1 shows the composition of each field.

There are three possible terminators for an S-record: CR, LF, and NULL. Additionally, an S-record may have an optional initial field to accommodate such other data as line numbers generated by some time-sharing systems. An S-record file is a normal ASCII text file in the operating system in which it resides.

The record length (byte count) and checksum fields ensure accuracy of transmission.

Table A-1. S-Record Field Composition

Field	Printable Characters	Contents
Type	2	S-record type: S0, S1, and so forth.
Record length	2	Number of character pairs in the record, excluding type and record length pairs.
Address	4, 6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
Code/data	0—n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (of 56 printable characters in the S-record).
Checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the records length, address, and the code/data fields.

A.3 S-RECORD TYPES

There are eight types of S-records, to accommodate the various needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other record transportation control programs, as well as cross assemblers, linkers and other file-creating or debugging programs, use only the S-record types that serve the purpose of the program. For specific information on which S-records a particular program supports, consult the user manual for that program. SDebug supports the S-record types listed in Table A-2.

Table A-2. S-Record Types

Type	Description
S0	The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. The address field is normally zeros.
S1	A record containing code/data and the 2-byte address at which the code/data is to reside.
S2	A record containing code/data and the 3-byte address at which the code/data is to reside.
S3	A record containing code/data and the 4-byte address at which the code/data is to reside.
S7	A termination record for a block of S3 records. The address field may optionally contain the 4-byte address of the instruction to which control is to be passed. There is no code/data field.
S8	A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.
S9	A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. If such an address is not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

There is only one termination record for each block of S-records. The usual reason to use S7 or S8 records is when control is to be passed to a 3- or 4-byte address. Normally, there is only one header record, although multiple header records are possible.

A.4 S-RECORD CREATION

Several dump utilities, debuggers, linkage editors, cross assemblers, or cross linkers may produce S-record-format programs. Several programs are available for downloading a file in S-record format from a host system to a microprocessor-based system.

A.5 S-RECORD EXAMPLE

The following example shows how a typical S-record format module is printed or displayed. The module consists of one S0, four S1, and one S9 records.

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S113003000144ED492
S9030000FC
```

The S0 record consists of these character pairs:

S0	Type indicator S0, identifying a header record.
06	Hexadecimal value 06, indicating that six character pairs (or ASCII bytes) follow.
00 00	Four-character 2-byte address field; zeros.
48 44 52	ASCII H, D, and R — "HDR"
1B	Checksum of the S0 record

The explanation of the first S1 code/data record is:

S1	S-record type S1, indicating a code/data record to be loaded/verified at a 2-byte address.
13	Hexadecimal 13, indicating that 19 character pairs, representing 19 bytes of binary data, follow.
00 00	Four-character, 2-byte, address field; hexadecimal address 0000, where the `00` following data is to be loaded
	The next 16 character pairs of the first S1 record are the ASCII bytes of the actual program code/data.
2A	The checksum of the first S1 record.

The second and third S1 records each also contain $13 (19) character pairs. These records end with checksums 13 and 52, respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The explanation of the S9 termination record is:

S9	S-record type S9, indicating that it is a termination record.
03	Hexadecimal 03, indicating that three character pairs (3 bytes) follow.
00 00	The address field, zeros.
FC	Checksum of the S9 record

Each printable character in an S-record is encoded in a hexadecimal representation (ASCII in this example) of the binary bits actually transmitted. For example, Figure A-1 is a diagram of the first S1 record described above.

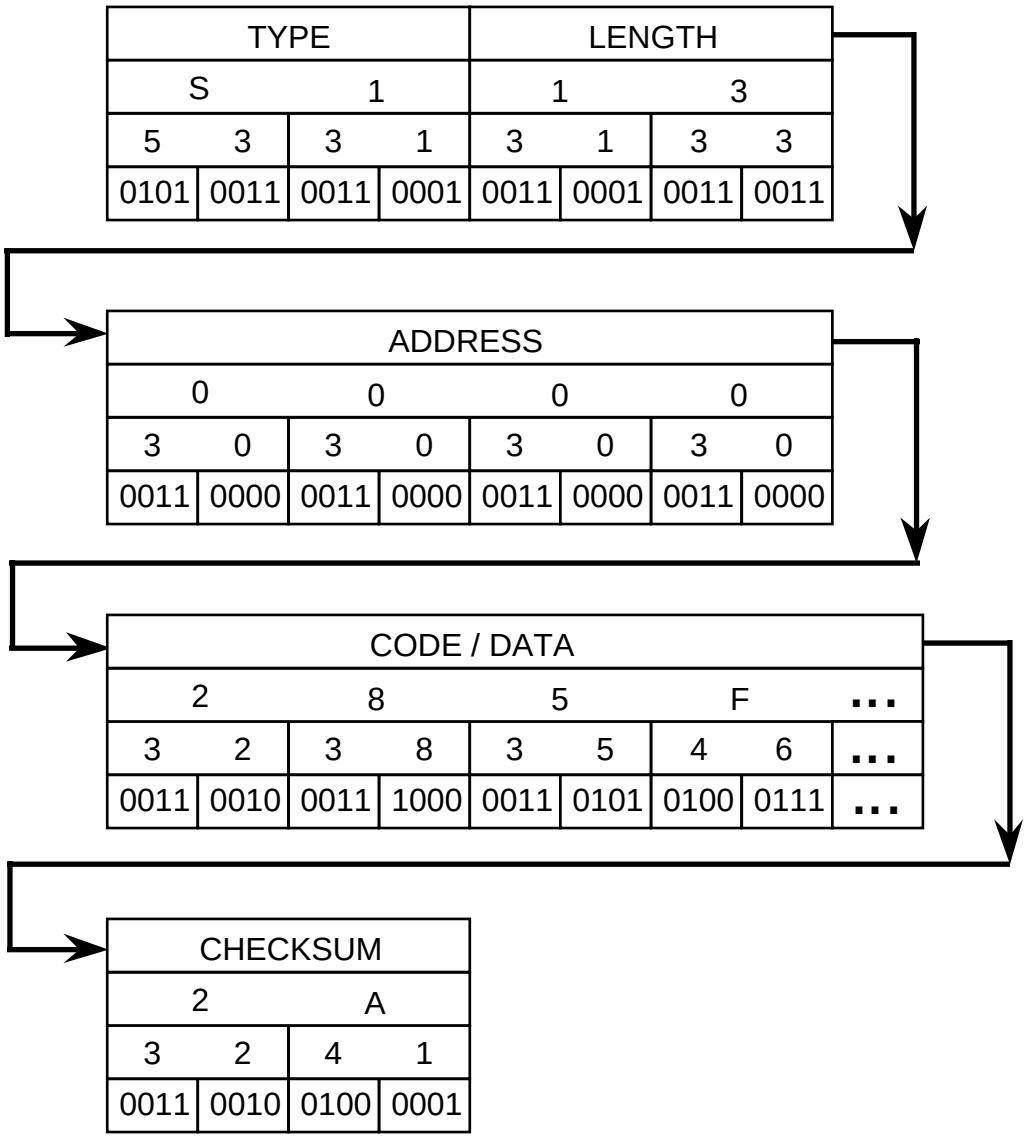


Figure A-1. S1 Record Diagram

APPENDIX B

STATUS AND ERROR MESSAGES

In most situations, the SDbug system reports conditions and changes, via messages in the debug window. Table B-1 lists these messages, with probable causes and corrective actions, as appropriate.

The IASM assembler, however, reports its own error messages on the prompt line. Table B-2 lists IASM assembler error messages.

Table B-1. SDbug Status and Error Messages

Message	Probable Cause	Corrective Action
Aborted by user	User action.	None (status message).
Assembly terminated – opcode form not valid	During interactive assembly, user entered a period or other inappropriate value.	None (status message).
Attempt to set a breakpoint at previously defined counter	Breakpoint address duplicates an existing counter address.	Change the breakpoint address or delete the counter.
Attempt to set breakpoint on odd address	Breakpoint address value is odd.	Change the address value to an even number.
Attempt to set counter at previously defined breakpoint	Counter address duplicates an existing breakpoint address.	Change the counter address or erase the breakpoint.
Attempt to set counter on odd address	Counter address value is odd.	Change the counter value to an even number.
Bus error not ready	A debugger window points to unimplemented memory.	Move the window.
Bus error BDM error	A debugger window points to unimplemented memory.	Move the window.

Table B-1. SDbug Status and Error Messages (continued)

Message	Probable Cause	Corrective Action
Could not write breakpoint/ counter to hardware	No system memory available or attempt to write to ROM.	Fix hardware or write to RAM.
Debugger supplied DSACK	A debugger window points to unimplemented memory.	Move the window.
DOS error after shell	An error occurred while running a program from its DOS shell.	Correct the error in the outer program.
Instructions allowed only on even byte boundaries	Instruction written to an odd address.	Change the address value to an even number.
Preset breakpoint encountered	Execution stopped at breakpoint.	None (status message).
Too many breakpoints set	Attempt to set an eighth breakpoint.	Delete a current breakpoint before setting a new one.
Too many counters set	Attempt to set a 51st counter.	Delete a current counter before setting a new one.
Unable to go into background mode	Hardware could not go into background mode.	Fix the hardware.
Unrecognizable or improper parameter value	Parameter value is out of range, wrong type, or otherwise not appropriate for this command.	Correct the parameter value.

Table B-2. IASM Assembler Error Messages

Message	Probable Cause	Corrective Action
Conditional assembly variable not found	The variable in the IF or IFNOT statement has not been declared via a SET or SETNOT directive.	Declare the variable via the SET or SETNOT directive.
Duplicate label	The label in the highlighted line already has been used.	Change the label to one not used already.
Error writing .LST or .MAP file–check disk space	Insufficient disk space or other reason prevents creation of an .LST or .MAP file.	Make sure there is sufficient disk space. Make sure (per your DOS manual) that your CONFIG.SYS file lets multiple files be open at the same time.
Error writing object file–check disk space	Insufficient disk space or other reason prevents creation of an object file.	Make sure there is sufficient disk space. Make sure (per your DOS manual) that your CONFIG.SYS file lets multiple files be open at the same time.
Include directives nested too deep	INCLUDE directives are nested 11 or more levels deep.	Nest includes no more than 10 levels deep.
INCLUDE file not found	Assembler could not find the file specified in the INCLUDE directive.	Make sure quotes enclose the file name. Specify any extension that exists. If necessary, specify the full path name.
Invalid base value	Value inconsistent with current default base (binary, octal, decimal, or hexadecimal).	Use a qualifier prefix or suffix for the value, or change the default base.
Invalid opcode, too long	The opcode on the highlighted line is wrong.	Correct the opcode.
MACRO label too long	A label in the macro has 11 or more characters.	Change the label to have no more than 10 characters.
MACRO parameter error	The macro did not receive sufficient parameter values.	Send sufficient parameter values to the macro.

Table B-2. IASM Assembler Error Messages (continued)

Message	Probable Cause	Corrective Action
Out of memory	The assembler ran out of system memory.	Create a file that consists only of an INCLUDE directive, which specifies your primary file. Assembling this file leaves the maximum memory available to the assembler.
Parameter: invalid, too large, missing or out of range	Operand field of the highlighted line has an invalid number representation. Or the parameter value evaluates to a number too large for memory space allocated to the instruction.	Correct the representation or change the parameter value.
Too many conditional assembly variables	There are 26 or more conditional variables.	Limit conditional variables to 25 or fewer.
Too many labels	The assembler ran out of system memory.	Create a file that consists only of an INCLUDE directive, which specifies your primary file. Assembling this file leaves the maximum memory available to the assembler.
Undefined label	The label parameter in the highlighted line has not been declared.	Declare the label (or correct the declaration).
Unrecognized operation	The opcode of the highlighted line is unknown or is inconsistent with the number and type of parameters.	Correct the opcode or make it consistent with parameters.
'}' not found	A mathematical expression is missing its close brace.	Insert the close brace.