ICD16Z User Guide - In-Circuit Debugger for 68HC16

1 Introduction

PEmicro's ICD16Z is a powerful tool for debugging code. It uses an 68HC16 processor's background debug mode (BDM), via one of PEmicro's hardware interfaces, to give the user access to all on-chip resources. ICD16Z is a great tool for creating repeatable testing, calibration, and debugging procedures.

2 Connection to Target

When you first run the software, you will see the Connection Assistant dialog box:

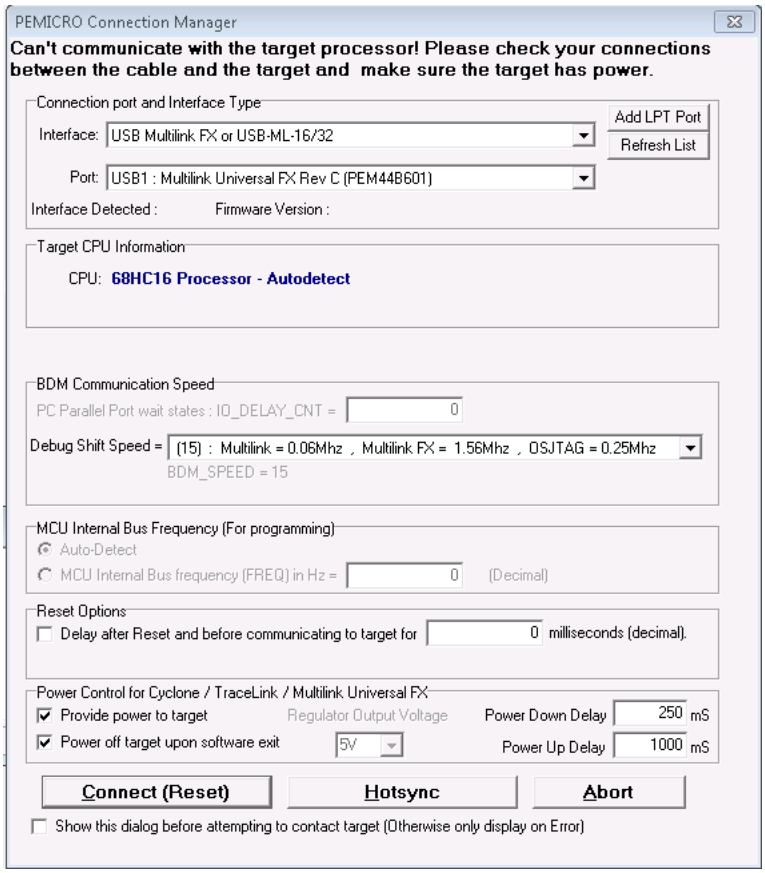


Figure 2-1: Connection Assistant Dialog

You may use this dialog box to connect to your target using one of PEmicro's debugging interface products. The dialog allows you to specify the type of PEmicro debugging interface hardware. You may also select the CPU type, communications speed, and reset delay. If you wish, you may disable the dialog box so that it will appear only on error.

Use the Connect button to reset the target. Use the Hotsync button to connect to the target without resetting it.

3 Command Line Parameters

To setup ICD16Z for windows to run with certain command line parameters, highlight the ICD16Z icon and select PROPERTIES from the Program Manager File Menu.

Syntax:

ICD16Z [option] ... [option]

[option]	Optional parameters are as follows:
lpt1…lpt3	Chooses lpt1, lpt2, or lpt3. The software will remember the last setting used.
pci1…pci6	Chooses which PCI card to communicate with. The software will remember the last setting used.
pci_delay n	Sets speed of PCI card shift clock, where n = 0…255. The equation for the PCI card shift clock frequency is 33 * 10^6 / (5 + 2n).
running	Starts ICD with CPU running (see RUNNING help)
io_delay_cnt n	Causes the background debug mode clock to be extended by 'n' Cycles where 1□=n□=64k. Used when using a very fast PC or a slow CPU clock. (default = 1);
reset_delay n	Causes a delay of 'n' milliseconds after the software pulses the reset line, and before the software checks the processor status to make sure that background mode has been entered. Used when reset pulse on the reset line is extended, for example by using a reset driver, which may add several hundred milliseconds to reset.
v1_reset	Allows the user to toggle the part to BERR immediately after reset, to avoid problems with BDM.
x1_reset	Disables CSBOOT chip select and enables CS0 at address 0 for 64k, words, read write, supervisor-user.
quiet	Starts the ICD without filling the memory windows and the disassembly window. Can be used for speed reasons or to avoid DSACK errors on startup until windows are positioned or chip selects enabled.
-or-	
path	A DOS path to the directory containing the source code for source level debug or a DOS path to a source file to be loaded at startup (path part is also saved).

Note:

If more then one option is given, they must be separated by spaces.

Examples:

ICD16Z.EXE lpt2 io_delay_cnt 2	Chooses lpt2, Causes the background debug mode clock to be extended by 2 Cycles.

Additionally, if a file named STARTUP.ICD exists in the current directory, it will be run as a macro at startup. See the MACRO command for more information.

4 Nomenclature

Note the following:

n	Any number from 0 to 0FFFFF (hex). The default base is hex. To enter numbers in another base use the suffixes 'T' for base ten, 'O' for base eight or 'Q' for base two. You may also use the prefixes '!' for base ten, '@' for base 8 and '%' for base two. Numbers must start with either one of these prefixes or a numeric character. Example: 0FF = 255T = 377O = 11111111Q = !255 = @377 = □11111111
add	Any valid address (default hex).
[ ]	Optional parameter.

PC	Program Counter points to the next instruction to be fetched (Equals IP+6).
str	ASCII string.
;	Everything on a command line after and including the “;” character is considered a comment. This helps in documenting macro (script) files.

5 User Interface

This section describes the windows that comprise the ICD16Z interface.

5.1 Code Window

The Code Window displays either disassembled machine code or the user's source code if it is available. The "Disassembly" mode will always show disassembled code regardless of whether a source file is loaded. The "Source/Disassembly" mode will show source code if source code is loaded and the current PC points to a valid line withing the source code, and shows disassembly otherwise. To show both modes at once, the user should have two code windows open and set one to "Disassembly" and the other to "Source/Disassembly".

Code windows also give visual indications of the Program Counter (PC) and breakpoints. Each code window is independent from the other and can be configured to show different parts of the user's code.

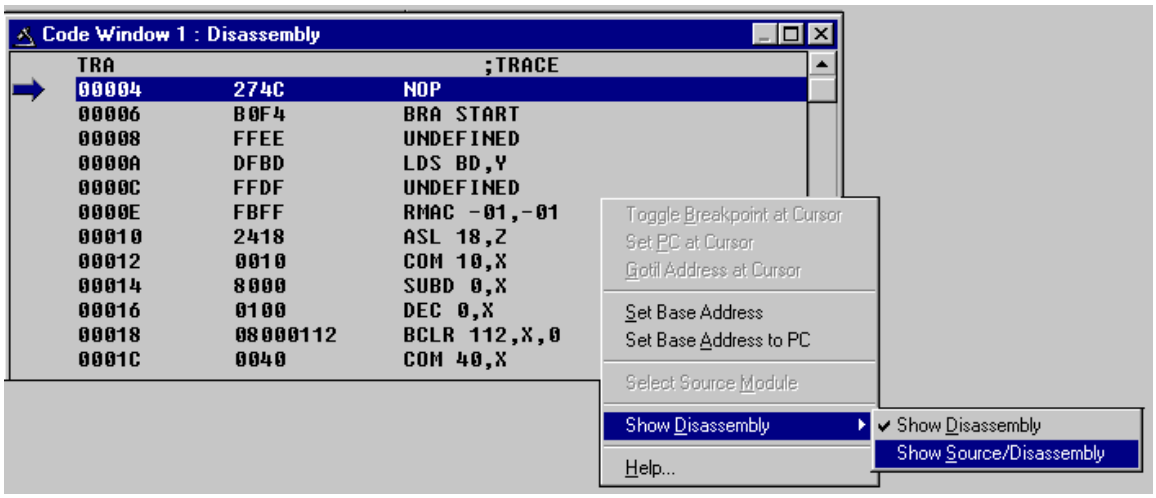


Figure 5-1: Code Window

POPUP MENU

By pressing the RIGHT MOUSE BUTTON while the cursor is over the code window, the user is given a popup menu which has the following options:

Toggle Breakpoint at Cursor

This option is enabled if the user has already selected a line in the code window by clicking on it with the LEFT MOUSE BUTTON. Choosing this option will set a breakpoint at the selected location, or if there is already a breakpoint at the selected location, will remove it.

Set PC at Cursor

This option is enabled if the user has already selected a line in the code window by clicking on it with the LEFT MOUSE BUTTON. Choosing this option will set the Program Counter (PC) to the selected location.

Gotil Address at Cursor

This option is enabled if the user has already selected a line in the code window by clicking on it with the LEFT MOUSE BUTTON. Choosing this option will set a temporary breakpoint at the selected line and starts processor execution (running mode). When execution stops, this temporary breakpoint is removed.

Set Base Address

This option allows the code window to look at different locations in the user's code, or anywhere in the memory map. The user will be prompted to enter an address or label to set the code window's base address. This address will be shown as the top line in the Code Window. This option is equivalent to the SHOWCODE command.

Set Base Address to PC

This option points the code window to look at the address where the program counter (PC) is. This address will be shown as the top line in the Code Window.

Select Source Module

This option is enabled if a source□level map file is currently loaded, and the windows mode is set to "Source/Disassembly". Selecting this option will pop up a list of all the map file's source filenames and allows the user to select one. This file is then loaded into the code window for the user to view.

Show Disassembly or Show Source/Disassembly

This option controls how the code window displays code to the user. The "Show Disassembly" mode will always show disassembled code regardless of whether a source file is loaded. The "Show Source/Disassembly" mode will show source-code if source code is loaded and the current PC points to a valid line withing the source code, and shows disassembly otherwise.

Help

Displays this help topic.

KEYSTROKES

The following keystrokes are valid while the code window is the active window:

UP ARROW	Scroll window up one line
DOWN ARROW	Scroll window down one line
HOME	Scroll window to the Code Window's base address.
END	Scroll window to last address the window will show.
PAGE UP	Scroll window up one page
PAGE DOWN	Scroll window down one page
F1	Shows this help topic
ESC	Make the STATUS window the active window

5.2 Status Window

The Status Window serves as the command prompt for the application. It takes keyboard commands given by the user, executes them, and returns an error or status update when needed.

Commands can be typed into the window, or a series of commands can be played from a macro file. This allows the user to have a standard sequence of events happen the same way every time. Refer to the MACRO command for more information.

It is often desirable to have a log of all the commands and command responses which appear in the status window. The LOGFILE command allows the user to start/stop the recording of all information to a text file which is displayed in the status window.

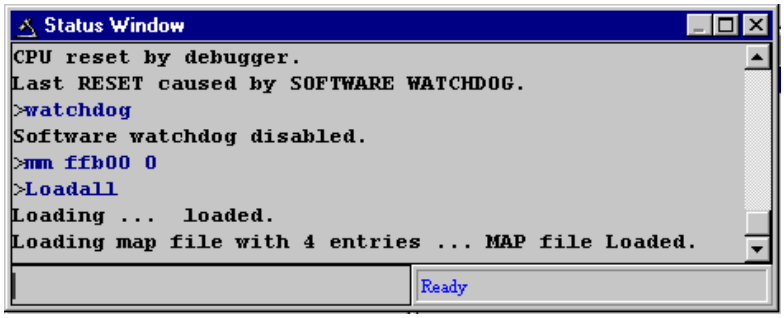


Figure 5-2: Status Window

POPUP MENU

By pressing the RIGHT MOUSE BUTTON while the cursor is over the status window, the user is given a popup menu which has the following options:

Help...

Displays this help topic.

KEYSTROKES

The following keystrokes are valid while the status window is the active window:

UP ARROW	Scroll window up one line
DOWN ARROW	Scroll window down one line
HOME	Scroll window to first status line
END	Scroll window to last status line
PAGE UP	Scroll window up one page
PAGE DOWN	Scroll window down one page
F1	Shows this help topic

To view previous commands and command responses, use the scroll bar on the right side of the window.

5.3 CPU Window

The CPU Window displays the current state of the registers.

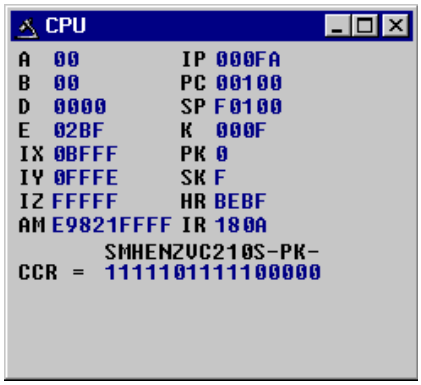


Figure 5-3: CPU Window

The popup window allows modification of these registers - double clicking on any of the registers displays a pop-up window which allows the user to modify the value. Right-clicking will also bring up a menu which allows you to modify these registers. The Condition Code Register must be changed using the CCR command.

KEYSTROKES

The following keystrokes are valid while the CPU window is the active window:

F1	Shows this help topic
ESC	Make the STATUS window the active window

5.4 Variables Window

The Variables Window is used to view variables while the part is not running. The user may add or remove variables through the Insert or Delete keys, the popup menu, or the VAR command. Variables can be viewed as bytes (8 bits), words (16 bits), longs (32 bits), or strings (ASCII).

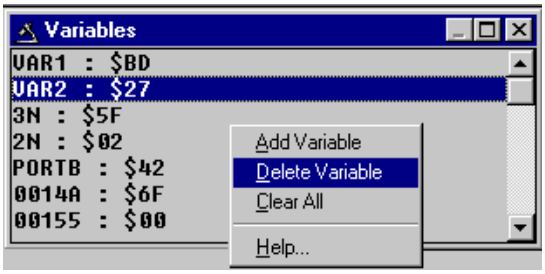


Figure 5-4: Variables Window

POPUP MENU

By pressing the RIGHT MOUSE BUTTON while the cursor is over the variables window, the user is given a popup menu which has the following options:

Add Variable

Adds a variable to the variables window at the currently selected line. A popup Add Variable box allows the user to specify the variable's address or label, type, and base.

Delete Variable

Removes the selected variable from the variables window. A variable is selected by placing the mouse cursor over the variable name and clicking the LEFT MOUSE BUTTON.

Clear All

Removes all variables from the variables window.

Help...

Shows this help topic.

KEYSTROKES

The following keystrokes aere valid while the variables window is the active window.

INSERT	Add a variable
DELETE	Delete a variable
UP ARROW	Scroll window up one variable
DOWN ARROW	Scroll window down one variable
HOME	Scroll window to the first variable
END	Scroll window to the last variable
PAGE UP	Scroll window up one page
PAGE DOWN	Scroll window down one page
F1	Shows this help topic
ESC	Make the Status Window the active window

5.5 Memory Window

The Memory Window is used to view and modify the memory map of a target. View bytes by using the scrollbar on the right side of the window. In order to modify a particular set of bytes, just double click on them. Double-clicking on bits brings up a byte modification window.

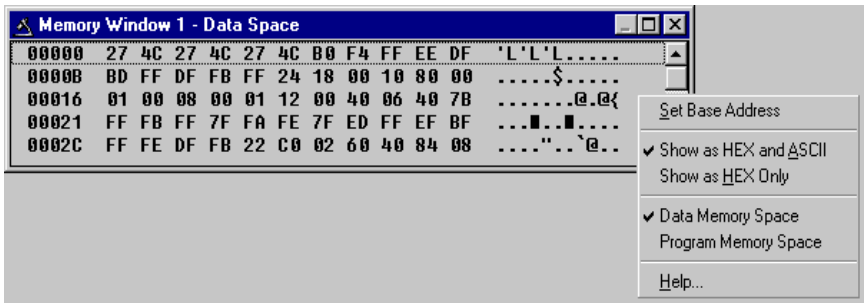


Figure 5-5: Memory Window

POPUP MENU

By pressing the RIGHT MOUSE BUTTON while the cursor is over the memory window, the user is given a popup menu which has the following options:

Set Base Address

Sets the memory window scrollbar to show whatever address the user specifies. Upon selecting this option, the user is prompted for the address or label to display. This option is equivalent to the Memory Display (MD) Command.

Show Memory and ASCII

Sets the current memory window display mode to display the memory in both HEX and ASCII formats.

Show Memory Only

Sets the current memory window display mode to display the memory in HEX format only.

Help...

Shows this help topic.

KEYSTROKES

The following keystrokes are valid while the memory window is the active window:

UP ARROW	Scroll window up one line
DOWN ARROW	Scroll window down one line
HOME	Scroll window to address $0000
END	Scroll window to last address in the memory map.
PAGE UP	Scroll window up one page
PAGE DOWN	Scroll window down one page
F1	Shows this help topic
ESC	Make the STATUS WINDOW the active window

5.6 Colors Window

The Colors Window shows the colors that are set for all of the debugger windows. In order to view the current color in a window, select the item of interest in the listbox and view the text in the bottom of the window. To change the color in a window, select the item and then use the left mouse button to select a color for the foreground or use the right mouse button to select a color for the background. Some items will only allow the foreground or background to be changed. Press the OK button to accept the color changes. Press the Cancel button to decline all changes.

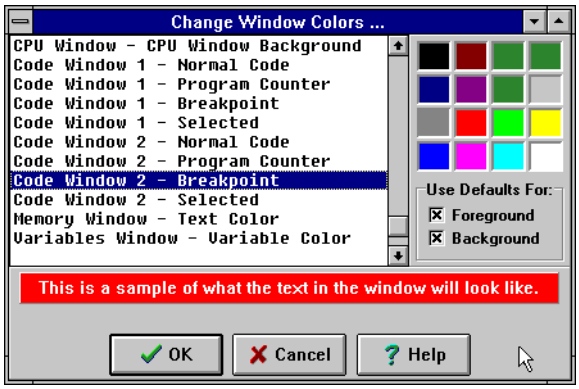


Figure 5-6: Colors Window

6 Command Recall

You can use the PgUp and PgDn keys to scroll through the past 30 commands issued in the debug window. Saved commands are those typed in by the user, or those entered through macro (script) files. You may use the ESC key to delete a currently entered line including one selected by scrolling through old commands.

Note that only "command lines" entered by the user are saved. Responses to other ICD prompts are not. For example, when a memory modify command is given with just an address, the ICD prompts you for data to be written in memory. These user responses are not saved for scrolling, however, the original memory modify command is saved.

7 ICD16Z Commands

7.1 A - Set Accumulator A

The A command sets the A accumulator to a specified value.

Syntax:

A <n>

Where:

<n> The value to be loaded into the accumulator.

Example:

>A $10 Set the accumulator to $10.

7.2 AMH Command

The AMH command allows the user to set the value of the 20-bit MAC Accumulator MSB (AM high). Same as MH or MMS commands.

Syntax:

AMH <n>

Where:

<n> Value to assign to accumulator.

Example:

AMH 4DC42 Assigns 4DC42 to accumulator.

7.3 AML Command

The AML command allows the user to set the value of the 16-bit MAC Accumulator LSB (AM low). Same as ML or MLS commands.

Syntax:

AML <n>

Where:

<n> Value to assign to accumulator.

Example:

AML D726 Assigns D726 to accumulator.

7.4 ASCIIF3 and ASCIIF6

Toggles the memory windows between displaying [data only] // [data and ASCII characters].

ASCIIF3 toggles memory window 1.

ASCIIF6 toggles memory window 2.

Syntax:

ASCIIF3

Example:

>ASCIIF3 Toggles memory window 1 between displaying [data only] // [data and ASCII characters].

7.5 ASM - Assemble Instructions

The ASM command assembles instruction mnemonics and places the resulting machine code into memory at the specified address. The command displays a window with the specified address (if given) and current instruction, and prompts for a new instruction. If an instruction is entered and the ENTER button is pressed, the command assembles the instruction, stores and displays the resulting machine code, then moves to the next memory location, with a prompt for another instruction. If there is an error in the instruction format, the address will stay at the current address and an 'assembly error' flag will show. To exit assembly, press the EXIT button. See Instruction Set for related information on instruction formats.

Syntax:

ASM [<address>]

Where:

<address> Address where machine code is to be generated. If you do not specify an <address> value, the system checks the address used by the previous ASM command, then uses the next address for this ASM command.

Examples:

With an address argument:

>ASM 100

The following window appears:

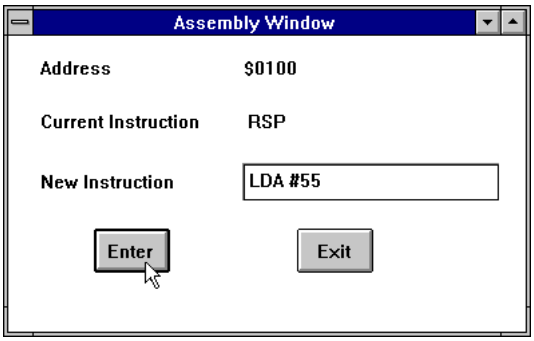


Figure 7-1: Assembly Window

The user can type a new instruction in the edit box next to the 'New Instruction' text. In this example, the instruction 'LDA #55' is typed and then the ENTER button is pressed. As soon as ENTER is click the following window will appears

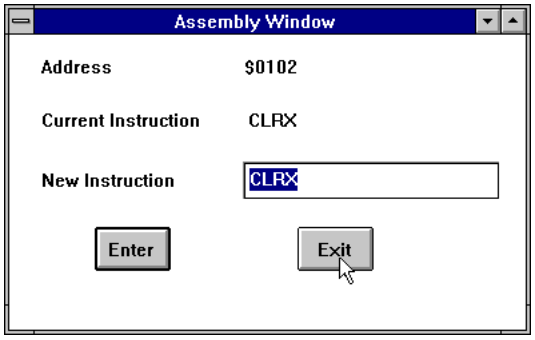


Figure 7-2: Assembly Window

This window shows that address is incremented by 2 and the instruction at address is CLRX. You can either enter another instruction or click EXIT to get out of this window.

7.6 B - Set Accumulator B

The B command sets the B accumulator to a specified value.

Syntax:

B <n>

Where:

<n> The value to be loaded into the accumulator.

Example:

>B $10 Set the accumulator to $10.

7.7 BELL - Sound Bell

The BELL command sounds the computer bell the specified hexadecimal number of times. The bell sounds once when no argument is entered. To turn off the bell as it is sounding, press any key.

Syntax:

BELL [<n>]

Where:

<n> The number of times to sound the bell.

Example:

>BELL 3 Ring PC bell 3 times.

7.8 BGND_TIME

First, the processor execution is started at the current PC. Then, each time a BGND instruction is encountered, the time since the last BGND instruction is logged in memory. Up to n points (default = 500 and max = 500 data points) may be logged. The accuracy is somewhere in the microsecond range. There is some positive time error to get in and out of background mode. In addition, while the ICD software is storing the information, the target processor is not running which introduces a real time error. One can determine the amount of time spent by the ICD to go into and out of BGND mode by timing the execution of a string of BGND instructions and deducting this from the times given. The data logging stops when 500 points have been logged or the operator presses a key. The logged points are then written to the debug window and also to the capture file if enabled.

Syntax:

BGND_TIME [n]

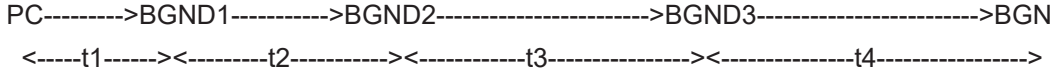
Where:

n	number of points logged

Example:

>BGND_TIME 4

The above command will give you four time differences (t1,t2,t3,t3).



7.9 BLOCK FILL or BF

The BF or FILL command fills a block of memory with a specified byte, word or long. The optional variant specifies whether to fill the block in bytes (.B, the default), in words (.W) or in longs (.L). Word and long must have even addresses.

Syntax:

BF[.B | .W | .L] <startrange> <endrange> <n>

FILL[.B | .W | .L] <startrange> <endrange> <n>

Where:

<startrange>	Beginning address of the memory block (range).
<endrange>	Ending address of the memory block (range).
<n>	Byte or word value to be stored in the specified block.

The variant can either be .B, .W, .L, where:

.B	Each byte of the block receives the value.
.W	Each word of the block receives the value.
.L	Each word of the block receives the value.

Examples:

>BF C0 CF FF	Store hex value FF in bytes at addresses C0-CF.
>FILL C0 CF FF	Store hex value FF in bytes at addresses C0-CF
>BF.B CO CF AA	Store hex value AA in bytes at addresses C0-CF.
>FILL.B CO CF AA	Store hex value AA in bytes at addresses C0-CF.
>BF.W 400 41F 4143	Store word hex value 4143 at addresses 400-41F.
>FILL.W 400 41F 4143	Store word hex value 4143 at addresses 400-41F.
>BF.L 1000 2000 8F86D143	Store long hex value 8F86D143 at address 1000-2000
>FILL.L 1000 2000 8F86D143	Store long hex value 8F86D143 at address 1000-2000

7.10 BR Command

Sets or clears a breakpoint at the indicated address. Break happens if an attempt is made to execute code from the given address. There are at most 7 breakpoints. They cannot be set at an odd address. Typing BR by itself will show all the breakpoints that are set and the current values for n.

Syntax:

BR [add] [n]

Where:

dd	Address at which a break point will be set.
n	If [n] is specified, the break will not occur unless that location has been executed n times. After the break occurs, n will be reset to its initial value. The default for n is 1.

Examples:

>BR	; Shows all the breakpoints that are set and the current values for n.
>BR 100	; Set break point at hex address 100.
>BR 200 5	; Break will not occur unless hex location 200 has been executed 5 times.

7.11 C - Set/Clear C-Bit

The C command sets or clears (that is, assigns 0 or 1 to) the C bit in the condition code register (CCR).

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

C 0|1

Examples:

>C 0	Clear the C bit of the CCR.
>C 1	Set the C bit of the CCR.

7.12 CAPTURE

Opens a capture file named 'filename'. Most outputs to the debug window are also sent to the capture file. The user is prompted for information as to appending to or deleting the 'filename' file if it already exists.

Syntax:

CAPTURE <filename>

Where:

<filename>	Name of the file where commands and messages are stored.

Example:

>CAPTURE testfile	Capture all the command and messages displayed at the debug window into the file "TESTFILE.CAP".

7.13 CAPTUREOFF

Turns off capturing of commands and messages at the debug window and closes the current capture file.

Syntax:

CAPTUREOFF

Example:

>CAPTUREOFF	Turns off capturing of commands and messages at the debug and window closes the current capture file.

7.14 CCR - Set Condition Code Register

The CCR command sets the condition code register (CCR) to the specified hexadecimal value.

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

CCR <n>

Where:

<n>The new hexadecimal value for the CCR.

Example:

>CCR $C4	Assign the value C4 to the CCR. This makes the the binary pattern 11000100; the S, X and N bits set, other bits clear.

7.15 CLEARMAP - Clear Map File

The CLEARMAP command removes the current MAP file from memory. This will force the debugger to show disassembly in the code windows instead of user source code. The user defined symbols, defined with the SYMBOL command, will not be affected by this command. (The NOMAP command is identical to CLEARMAP.)

Syntax:

CLEARMAP

Example:

>CLEARMAP	Clears symbol and source information.

7.16 CLEARSYMBOL - Clear User Symbols

The CLEARSYMBOL command removes all the user defined symbols. The user defined symbols are all created with the SYMBOL command. The debug information from MAP files, used for source level debugging, will be unaffected. The NOSYMBOL command is identical.

Note:

Current user defined symbols can be listed with the SYMBOL command.

Syntax:

CLEARSYMBOL

Example:

>CLEARSYMBOL	Clears user defined symbols.

7.17 CLEARVAR

The CLEARVAR command removes all the variables from the variables window.

Syntax:

CLEARVAR

Example:

CLEARVAR Removes all the variables fmor the variables window.

7.18 CODE

Shows disassembled code in the code window starting at address add. If you specify an address in the middle of an intended instruction, improper results may occur.

Syntax:

CODE <add>

Where:

<add> Address where your code begins.

Example:

>CODE 100 Shows the disassembled code in the code window starting at hex address 100.

7.19 COLORS - Set Colors of Simulator

The COLORS command brings up a popup window, the Colors Window, that allows the user to choose the text and background colors for all windows in the debugger. Once colors are selected for the windows, use the SAVEDESK command to save them for all further debugging sessions. See Colors Window for more information.

Syntax:

COLORS

Example:

>COLORS Open the colors window.

7.20 COUNT

The COUNT command tells the user how many times each address in the internal counter table is executed. If no address parameters are provided, the processor will execute from the current Program Counter until an existing breakpoint is encountered, or the user presses a key. If the user provides a starting address [add1], the processor will begin executing from this address until it reaches the second address [add2], or if that parameter is not given, until an existing breakpoint is encountered, or the user presses a key. When a breakpoint or keypress occurs, you are put into the "Show Count" window. The count locations set in the source code window are shown in descending order of executions. The percent is a rough percent of all counts. You may scroll in this window using the cursor keys and return to the debug window by hitting F1.

The addresses in the internal counter table are set using the COUNTER command.

Syntax:

COUNT [add1] [add2]

Where:

add1 Go from first address.

add2 Set breakpoint at second address.

Example:

>COUNT 100 200 Start execution of the program at address 100 and stops at address 200.

7.21 COUNTER

Adds or subtracts a location from the internal counter table. The user may then use the COUNT command to count how many times each of the locations in the table executes. Using the COUNTER command with no address shows the current table of counters.

Syntax:

COUNTER [add]

Where:

add	Address to be added to, or removed from, the internal counter table.

Example:

>COUNTER 100	Add (or remove) a counter at hex location 100.
>COUNTER	Shows all the current internal counters.

7.22 D - Set Double Accumulator D

The D command sets the D double accumulator to a specified value.

Syntax:

D <n>

Where:

<n>	The value to be loaded into the accumulator.

Example:

>D $10	Set the accumulator to $10.

7.23 D_UPLOAD_SREC

The D_UPLOAD_SREC command uploads the content of the specified data memory block (range), in .S19 object file format, displaying the contents in the status window. If a log file is opened, then D_UPLOAD_SREC will put the information into it as well.

Note:

If the D_UPLOAD_SREC command is entered, sometimes the memory contents scroll through the debug window too rapidly to view. Accordingly, either the LOGFILE command should be used, which records the contents into a file, or use the scroll bars in the status window.

Syntax:

D_UPLOAD_SREC <startrange> <endrange>

Where:

<startrange>	Beginning address of the memory block.
<endrange>	Ending address of the memory block (range)

Example:

>D_UPLOAD_SREC 300 7FF	Upload the 300□7FF memory block in .S19 format.

7.24 DASM - Disassemble Memory

The DASM command disassembles machine instructions, displaying the addresses and their contents as disassembled instructions in the status window. The memory locations between the first and second addresses (add) are uploaded to the screen in the form of Bytes, Words, or Longwords. The first address must be on an even boundary for Words or Longwords. If the capture feature is active, the lines of dumped data are also sent to the capture file. Data is read as Bytes, Words, or Longwords from the data space.

- If the command includes an address value, one disassembled instruction is shown, beginning at that address.

- If the command is entered without any parameter values, the software finds the most recently disassembled instruction then shows the next instruction, disassembled.
- If the command includes startrange and endrange values, the software shows disassembled instructions for the range.

Note:

If the DASM command is entered with a range, sometimes the disassembled instructions scroll through the status window too rapidly to view. Accordingly, the LF command can be entered, which records the disassembled instructions into a logfile, or use the scroll bars in the status window.

Syntax:

DASM <address1> [<address2>] [n]

Where:

<address1>	The starting address for disassembly. <address1> must be an instruction opcode. If you enter only an <address1> value, the system disassembles three instructions.
<address2>	The ending address for disassembly (optional). If you enter an <address2> value, disassembly begins at <address1> and continues through <address2>. The screen scrolls upward as addresses and their contents are displayed, leaving the last instructions in the range displayed in the window.
n	The optional parameter n determines the number of Bytes, Words, or Longwords which are written on one line.

Examples:

```
>DASM 300
0300    A6E8    LDA #0E8
0302    B700    STA PORTA
0304    A6FE    LDA #FE

>DASM 400 408
0400    5F      CLRX
0401    A680    LDA #80
0403    B700    STA PORTA
0405    A6FE    LDA #FE
0407    B704    STA DDRA
```

7.25 DMM[.X] Command

The DMM[.X] command sets the data RAM location at the address given to the value given. If no value is specified, the user is prompted for one. Consecutive locations will be prompted for until an invalid number is input. Multiple values (separated by spaces) for consecutive addresses (based on the .X parameter) may be entered on a single line. The .W extension causes the input to be interpreted as words (16 bits), .L as long words (32 bits) and .B (the default) as bytes. See PMM for program memory modify.

Syntax:

DMM[.X] add [n]

Where:

.X	Units in which RAM is written. .B for bytes, .W for words, .L for long words

add	Starting RAM address to which values are written.
n	Data to write. Listing more than one value causes each value to be written to consecutive addresses.

Example:

DMM 200 1 2 3 4	Writes values 1-4 into RAM locations $200-$203
DMM 300	Will prompt for values to be written, starting at address $300...
300 = 32 >	Enter new value for RAM location $300, or nonsense character to quit.

7.26 DUMP - Dump Data Memory to Screen

The DUMP command sends contents of a block of data memory to the status window, in bytes, words, or longs. The optional variant specifies whether to fill the block in bytes (.B, the default), in words (.W), or in longs (.L).

Note:

When the DUMP command is entered, sometimes the memory contents scroll through the debug window too rapidly to view. Accordingly, either the LF command can be entered, which records the memory locations into a logfile, or the scroll bars in the status window can be used.

Syntax:

DUMP [.B | .W | .L] <startrange> <endrange> [<n>]

Where:

<startrange>	Beginning address of the data memory block.
<endrange>	Ending address of the data memory block (range).
<n>	Optional number of bytes, words, or longs to be written on one line.

Examples:

>DUMP C0 CF	Dump array of RAM data memory values, in bytes.
>DUMP.W 400 47F	Dump ROM code from data memory hex addresses 400 to 47F in words.
>DUMP.B 300 400 8	Dump contents of data memory hex addresses 300 to 400 in rows of eight bytes.

7.27 DUMP_TRACE

Dumps the current trace buffer to the debug window and to the capture file if enabled.

Syntax:

Dump_Trace

Example:

>Dump_Trace

7.28 E - Set Accumulator E

The E command sets the E accumulator to a specified value.

Syntax:

E <n>

Where:

<n>	The value to be loaded into the accumulator.

Example:

>E $10 Set the accumulator to $10.

7.29 EK - Set EK Extension Register

The EK command allows the user to set the value of the EK Extension Register.

Syntax:

EK n

Where:

n = value from 0-F

Example:

EK 8 Sets value of EK Register to 8.

7.30 EVAL - Evaluate Expression

The EVAL command evaluates a numerical term or simple expression, giving the result in hexadecimal, decimal, octal, and binary formats. In an expression, spaces must separate the operator from the numerical terms.

Note that octal numbers are not valid as parameter values. Operand values must be 16 bits or less. If the value is an ASCII character, this command also shows the ASCII character as well. The parameters for the command can be either just a number or a sequence of : number, space, operator, space, and number. Supported operations are addition (+), subtraction (□), multiplication (□), division (□), logical AND (&), and logical OR (^).

Syntax:

EVAL <n> [<op> <n>]

Where:

<n> Alone, the numerical term to be evaluated. Otherwise either numerical term of a simple expression.

<op> The arithmetic operator (+, -, *, /, &, or ^) of a simple expression to be evaluated.

Examples:

>EVAL 45 + 32

004DH 077T 000115O 0000000001001101Q "w"

>EVAL 100T

0064H 100T 000144O 0000000001100100Q "d"

7.31 EXIT or QUIT - Exit Program

The EXIT command terminates the software and closes all windows. If the debugger is called from WINIDE it will return there. The QUIT command is identical to EXIT.

Syntax:

EXIT

Example:

>EXIT Finish working with the program.

7.32 G or GO or RUN - Begin Program Execution

The G or GO or RUN command starts execution of code in the Debugger at the current Program Counter (PC) address, or at an optional specified address. When only one address is entered, that address is the starting address. When a second address is entered, execution stops at that address. The G or GO or RUN commands are identical. When only one address is specified,

execution continues until a key or mouse is pressed, a breakpoint set with a BR command occurs, or an error occurs.

Syntax:

GO [<startaddr> [<endaddr>]]

Where:

<startaddr>	Optional execution starting address. If the command does not have a <startaddr> value, execution begins at the current PC value.
<endaddr>	Optional execution ending address.

Examples:

>GO	Begin code execution at the current PC value.
>GO 346	Begin code execution at hex address 346.
>G 400 471	Begin code execution at hex address 400. End code execution just before the instruction at hex address 471.
>RUN 400	Begin code execution at hex address 400.

7.33 GOEXIT

Similar to GO command except that the target is left running without any breakpoints and the debugger software is terminated.

Syntax:

GOEXIT [add]

Where:

add	Starting address of your code.

Example:

>GOEXIT 100	This will set the program counter to hex location 100, run the program and exit from the background debugging mode.

7.34 GONEXT

Go from the current PC until the next instruction is reached. Used to execute past a subroutine call or past intervening interrupts.

Syntax:

GONEXT

Example:

>GONEXT	Goes from the current PC until the next instruction is reached.

7.35 GOTIL - Execute Program until Address

The GOTIL command executes the program in the Debugger beginning at the address in the Program Counter (PC). Execution continues until the program counter contains the specified ending address or until a key or mouse is pressed, a breakpoint set with a BR command occurs, or an error occurs.

Syntax:

GOTIL <endaddr>

Where:

<endaddr>	The address at which execution stops.

Example:

>GOTIL 3F0	Executes the program in the Debugger up to hex address 3F0.

7.36 GOTILROM

Executes fast single steps without updating the screen, until the address is reached. This is the fastest way to breakpoint in ROM.

Syntax:

GOTILROM [add]

Where:

add	Starting address of your code.

Example:

>GOTILROM 1000	This will do fast single steps from the location where your program counter is set at and stops at hex location 1000 which in this example is the starting location of the ROM. Starting location of the ROM depends on the memory map of your system. After reaching hex 1000 you can do single step to debug the code.

7.37 H - Set/Clear H-Bit

The H command sets or clears (that is, assigns 0 or 1 to) the H bit in the condition code register (CCR).

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

H 0|1

Examples:

>H 0	Clear the H bit of the CCR.
>H 1	Set the H bit of the CCR.

7.38 HELP - Open Help File

The HELP command opens the Windows help file for the program. If this command is entered with an optional parameter, help information specifically for that parameter appears. If this command is entered without any parameter value, the main contents for the help file appears.

An alternative way to open the help system is to press the F1 key.

Syntax:

HELP [<topic>]

Where:

<topic>	a debug command or assembly instruction

Examples:

>HELP	Open the help system
>HELP GO	Open GO command help information.

7.39 HR - Set MAC Register H

The HR command sets the MAC register H to a specified value. The HREG command is identical to the HR command.

Syntax:

HR <value>

Where:

<value> The new value for the HR register.

Example:

>HR $05F3 Set the index register value to 05F3.

>HREG $3CC0 Set the index register value to 3CC0.

7.40 INFO - Display Line Information

The INFO command displays information about the line that is highlighted in the source window. Information displayed includes the name of the file being displayed in the window, the line number, the address, the corresponding object code, and the disassembled instruction.

Syntax:

INFO

Example:

>INFO Display information about the cursor line.

Shows:

Filename: PODTEST.ASMLine number:6

Address: $0100

Disassembly: START 5F CLRX

7.41 IP - Interrupt Priority Field

The IP command allows the user to assign a value from 0 to 7 to the Interrupt Priority Field in the condition code register (CCR).

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

IP <n>

Where:

<n> A value from 0 - 7

Examples:

>IP 6 Assigns 6 to the IP Field.

7.42 IR - Set MAC Register I

The IR command sets the MAC register I to a specified value. The IREG command is identical to the IR command.

Syntax:

IR <value>

Where:

<value> The new value for the IR register.

Example:

>IR $05F3	Set the index register value to 05F3.
>IREG $3CC0	Set the index register value to 3CC0.

7.43 IX - Set X Index Register

The IX command sets the index register (X) to a specified value. The IX command is identical to the X command.

Syntax:

IX <value>

Where:

<value>	The new value for the X register.

Example:

>IX $25	Set the index register value to 25.
>X $F0	Set the index register value to F0.

7.44 IY - Set Y Index Register

The IY command sets the index register (Y) to a specified value. The IY command is identical to the Y command.

Syntax:

IY <value>

Where:

<value>	The new value for the Y register.

Example:

>IY $05	Set the index register value to 05.
>Y $F0	Set the index register value to F0.

7.45 IZ - Set Z Index Register

The IZ command sets the index register (Z) to a specified value.

Syntax:

IZ <value>

Where:

<value>	The new value for the Z register.

Example:

>IZ $05	Set the index register value hex to 05.

7.46 K - Set Address Extension Registers

The K command allows the user to set the value of the EK, XK, YK and ZK Extension Registers.

Syntax:

K <n1><n2><n3><n4>

Where:

<n1> = value from 0-F

<n2> = value from 0-F

<n3> = value from 0-F

<n4> = value from 0-F

Example:

K $4D0A Sets value of EK to 4, XK to D, YK to 0, and ZK to A.

7.47 LF or LOGFILE - Open / Close Log File

The LF command opens an external file to receive log entries of commands and copies of responses in the status window. If the specified file does not exist, this command creates the file. The LOGFILE command is identical to LF.

If the file already exists, an optional parameter can be used to specify whether to overwrite existing contents (R, the default) or to append the log entries (A). If this parameter is omitted, a prompt asks for this overwrite/append choice.

While logging remains in effect, any line that is appended to the command log window is also written to the log file. Logging continues until another LOGFILE or LF command is entered without any parameter values. This second command disables logging and closes the log file.

The command interpreter does not assume a filename extension.

Syntax:

LF [<filename> [<R | A>]]

Where:

<filename> The filename of the log file (or logging device to which the log is written).

Examples:

>LF TEST.LOG R Start logging. Overwrite file TEST.LOG (in the current directory) with all lines that appear in the status window.

>LF TEMP.LOG A Start logging. Append to file TEMP.LOG (in the current directory) all lines that appear in the status window.

>LOGFILE (If logging is enabled): Disable logging and close the log file.

7.48 LISTON - Show Info during Steps

The LISTON command turns on the screen listing of the step by step information during stepping. The register values and program instructions will be displayed in the status window while running the code. The values shown are the same values seen by the REG instruction.

To turn off this step display, use the LISTOFF command.

Syntax:

LISTON

Example:

>LISTON Show step information.

7.49 LISTOFF - Do Not Show Info During Steps

The LISTOFF command turns off the screen listing of the step-by-step information for stepping. Register values and program instructions do not appear in the status window as code runs. (This display state is the default when the software is first started.)

To turn on the display of stepping information, use the LISTON command.

Syntax:

LISTOFF

Example:

>LISTOFF Do not show step information.

7.50 LOAD - Load S19 and MAP

The LOAD command loads a file in .S19 format into the Debugger. Entering this command without a filename value brings up a list of .S19 files in the current directory. You can select a file to be loaded directly from this list.

Syntax:

LOAD [<filename>]

Where:

<filename>	The name of the .S19 file to be loaded. You can omit the .S19 extension. The filename value can be a pathname that includes an asterisk (*) wildcard character. If so, the command displays a window that lists the files in the specified directory, having the .S19 extension.

Examples:

>LOAD PROG1.S19	Load file PROG1.S19 and its map file into the Debugger at the load addresses in the file.
>LOAD PROG2	Load file PROG2.S19 and its map file into the Debugger at the load addresses in the file.
>LOAD A:	Display the names of the .S19 files on the diskette in drive A:, for user selection.
>LOAD	Display the names of the .S19 files in the current directory, for user selection.

7.51 LOADALL

Does a LOAD and a LOADMAP command.

Syntax:

LOADALL [filename]

Where:

filename	Filename of your source code

Example:

LOADALL myprog	This command will load the S19 object file and the PEmicro Map file.

7.52 LOADDESK - Load Desktop Settings

The LOADDESK command loads the desktop settings that set the window positions, size, and visibility. This allows the user to set how the windows are set up for the application. Use SAVEDESK to save the settings of the windows of the debugger into the desktop file.

Syntax:

LOADDESK

Example:

>LOADDESK	Get window settings from desktop file.

7.53 LOADMAP - Load Map File

The LOADMAP command loads a map file that contains source level debug information into the debugger. Entering this command without a filename parameter brings up a list of .MAP files in the current directory. From this a file can be selected directly for loading map file information.

Syntax:

LOADMAP [<filename>]

Where:

<filename>	The name of a map file to be loaded. The .MAP extension can be omitted. The filename value can be a pathname that includes an asterisk (□) wildcard character□ If so, the command displays a lists of all files in the specified directory that have the .MAP extension.

Examples:

>LOADMAP PROG.MAP	Load map file PROG.MAP into the host computer.

>LOADMAP PROG1	Load map file PROG1.MAP into the host computer.
>LOADMAP A:	Displays the names of the .MAP files on the diskette in drive A:
>LOADMAP	Display the names of the .MAP files in the current directory.

7.54 LOADV

First performs the LOAD command and then automatically does a VERIFY command with the same file.

Syntax:

LOADV [filename]

Where:

filename	Filename of your source code

Example:

LOADV myprog	This command will load the S19 on to the target and then it will read the contents of the S19 file from the target board and compare it with the 'myprog' file.

7.55 LOAD_BIN

Loads a binary file of bytes starting at address add. The default filename extension is .BIN.

Syntax:

LOAD_BIN [filename] [add]

Where:

filename	Name of the binary file
add	Starting address

Example:

>LOAD_BIN myfile.bin 100Loads a binary myfile of bytes starting at hex address 100

7.56 LOADV_BIN

First performs the LOAD_BIN command and then does a verify using the same file.

Syntax:

LOADV_BIN [filename] [add]

Where:

filename	Name of the binary file
add	Starting address

Example:

>LOADV_BIN myfile.bin 100	Loads a binary myfile of bytes starting at hex address 100 and then does a verify using the same file.

7.57 MAC Command

The MAC command shows the contents of the MAC (Multiply and ACcumulate) unit, which includes the IR and HR registers, AM accumulator, XMASK and YMASK mask registers, and the SIGNLATCH.

Syntax:

MAC

Example:

>MAC	Shows the contents of the MAC unit.

7.58 MACS

Brings up a window with a list of macros. These are files with the extension .ICD (such as the STARTUP.ICD macro). Use the arrow keys and the <ENTER> key or mouse click to select. cancel with the <ESC> key.

Syntax:

MACS

Example:

>MACS	Brings up a list of MACROS

7.59 MACRO or SCRIPT - Execute a Batch File

The MACRO command executes a macro file, a file that contains a sequence of debug commands. Executing the macro file has the same effect as executing the individual commands, one after another. Entering this command without a filename value brings up a list of macro (.MAC) files in the current directory. A file can be selected for execution directly from this list. The SCRIPT command is identical.

Note:

A macro file can contain the MACRO command; in this way, macro files can be nested as many as 16 levels deep. Also note that the most common use of the REM and WAIT commands is within macro files. The REM command displays comments while the macro file executes.

If a startup macro file is found in the directory, startup routines run the macro file each time the application is started. See STARTUP for more information.

Syntax:

MACRO <filename>

Where:

<filename>	The name of a macro file to be executed, with or without extension .MAC. The filename can be a pathname that includes an asterisk(*) wildcard character. If so, the software displays a list of macro files, for selection.

Examples:

>MACRO INIT.MAC	Execute commands in file INIT.MAC.
>SCRIPT □	Display names of all .MAC files (then execute the selected file).
>MACRO A:□	Display names of all .MAC files in drive A (then execute the selected file).
>MACRO	Display names of all .MAC files in the current directory, then execute the selected file.

7.60 MACROEND - Stop Saving Commands to File

The MACROEND command closes the macro file in which the software has saved debug commands. (The MACROSTART command opened the macro file). This will stop saving debug commands to the macro file.

Syntax:

MACROEND

Example:

>MACROEND	Stop saving debug commands to the macro file, then close the file.

7.61 MACROSTART - Save Debug Commands to File

The MACROSTART command opens a macro file and saves all subsequent debug commands to that file for later use. This file must be closed by the MACROEND command before the debugging session is ended.

Syntax:

MACROSTART [<filename>]

Where:

<filename>	The name of the macro file to save commands. The .MAC extension can be omitted. The filename can be a pathname followed by the asterisk (□) wildcard character; if so, the command displays a list of all files in the specified directory that have the .MAC extension.

Example:

>MACROSTART TEST.MAC Save debug commands in macro file TEST.MAC

7.62 MD or SHOW - Display Memory at Address

The MD command displays (in the memory window) the contents of memory locations beginning at the specified address. The number of bytes shown depends on the size of the window and whether ASCII values are being shown. See Memory Window for more information. If a log file is open, this command also writes the first 16 bytes to the log file.

The MD and SHOW commands are identical.

Syntax:

MD <address>

Where:

<address>	The starting memory address for display in the upper left corner of the memory window.

Examples:

>MD 200	Display the contents of memory beginning at hex address 200.
>SHOW 100	Display the contents of memory beginning at hex address 100.

7.63 MD2 Command

The MD2 command displays the contents of 32 emulation memory locations in the second memory window. The specified address is the first of the 32 locations. If a logfile is open, this command also writes the first 16 values to the logfile.

Syntax:

MD2 <address>

Where:

<address>	The starting memory address for display in the memory window.

Example:

>MD2 1000	Display the contents of 32 bytes of memory in the second memory window, beginning at address 1000.

7.64 MDF3 / MDF6 or SHOWF3 / SHOWF6

Sets a memory screen to show code starting at a specified address or label.

MDF3 displays the code in memory window F3 (same as SHOWF3).

MDF6 displays the code in memory window F6 (same as SHOWF6).

Syntax:

MDF3 add

Example:

>MDF3 1000	Dislpays code beginning at address $1000 in memory window F3.

7.65 MM or MEM - Modify Memory

The MM command directly modifies the contents of memory beginning at the specified address. The optional variant specifies whether to fill the block in bytes (.B, the default), in words (.W), or in longs (.L).

If the command has only an address value, a Modify Memory window appears with the specified address and its present value and allows entry of a new value for that address. Also, buttons can be selected for modifying bytes (8 bit), words (16 bit), and longs (32 bit). If only that address is to be modified, enter the new value in the edit box and press the OK button. The new value will be placed at the location. If the user wishes to modify several locations at a time, enter the new value in the edit box and press the >> or << or = button. The new value will be placed at the specified address, and then the next address shown will be the current address incremented, decremented, or the same, depending on which button is pressed. To leave the memory modify window, either the OK or CANCEL buttons must be pressed.

If the MM command includes optional data value(s), the software assigns the value(s) to the specified address(es) (sequentially), then the command ends. No window will appear in this case.

Syntax:

MM [.B|.W|.L] <address>[<n> ...]

Where:

<address>	The address of the first memory location to be modified.
<n>	The value(s) to be stored (optional).

Examples:

With only an address:

>MM 90	Start memory modify at address $90.

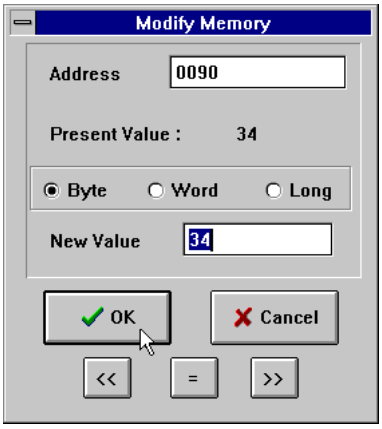


Figure 7-3: Modify Memory Window

With a second parameter:

>MM 400 00	Do not show window, just assign value 00 to hex address 400.
>MM.L 200 123456	Place long hex value 123456 at hex address 200.

7.66 N - Set/Clear N-Bit

The N command sets or clears (that is, assigns 0 or 1 to) the N bit in the condition code register (CCR).

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

N 0|1

Examples:

>N 0	Clear the N bit of the CCR.
>N 1	Set the N bit of the CCR.

7.67 NOBR

Clears all break points.

Syntax:

NOBR

Example:

>NOBRClears all break points.

7.68 PC - Set Program Counter

The PC command assigns the specified value to the program counter (PC). As the PC always points to the address of the next instruction to be executed , assigning a new PC value changes the flow of code execution.

An alternative way for setting the Program Counter if source code is showing in a code window is to position the cursor on a line of code, then press the right mouse button and select the Set PC at Cursor menu item. This assigns the address of that line to the PC.

Syntax:

PC <address>

Where:

<address>	The new PC value.

Example:

>PC $0500	Sets the PC value to 0500.

7.69 PDUMP - Dump Program Memory To Screen

The PDUMP command sends contents of a block of program memory to the status window, in bytes, words, or longs. The optional variant specifies whether to fill the block in bytes (.B, the default), in words (.W), or in longs (.L).

Note:

When the PDUMP command is entered, sometimes the memory contents scroll through the debug window too rapidly to view. Accordingly, either the LF command can be entered, which records the memory locations into a logfile, or the scroll bars in the status window can be used.

Syntax:

PDUMP [.B | .W | .L] <startrange> <endrange> [<n>]

Where:

<startrange>	Beginning address of the program memory block.
<endrange>	Ending address of the program memory block (range).
<n>	Optional number of bytes, words, or longs to be written on one line.

Examples:

>PDUMP C0 CF	Dump array of RAM programming memory values, in bytes.
>PDUMP.W 400 47F	Dump ROM code from program memory hex addresses 400 to 47F in words.
>PDUMP.B 300 400 8	Dump contents of program memory hex addresses 300 to 400 in rows of eight bytes.

7.70 PK - Set PC Extension Register

The PK command assigns a value to PK [3:0], the PC extension register.

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear. IP should be written using the CCR command.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

PK n

Where:

n = 0-F

Examples:

>PK $D	Sets PK to 1101 (PK[0]=1 PK[1]=1 PK[2]=0 PK[3]=1)
>PK 1101B	Sets PK to 1101 (PK[0]=1 PK[1]=1 PK[2]=0 PK[3]=1)

7.71 PMM[.X] Command

The PMM[.X] command sets the program location at the address given to the value given. If no value is specified, the user is prompted for one. Consecutive locations will be prompted for until an invalid number is input. Multiple values (separated by spaces) for consecutive addresses (based on the .X parameter) may be entered on a single line. The .W extension causes the input to be interpreted as words (16 bits), .L as long words (32 bits) and .B (the default) as bytes. See DMM for RAM memory modify.

Syntax:

PMM[.X] add [n]

Where:

.X	Units in which program location is written. .B for bytes, .W for words, .L for long words
add	Starting program location address to which values are written.
n	Data to write. Listing more than one value causes each value to be written to consecutive addresses.

Example:

PMM 200 1 2 3 4Writes values 1-4 into program locations $200-$203

PMM 300	Will prompt for values to be written, starting at address $300...
300 = 32 >	Enter new value for program location $300, or nonsense character to quit.

7.72 QUIET

Turns off (or on) refresh of memory based windows. This command can be used on the startup command line. Default = on.

Syntax:

QUIET

Example:

>QUIET Turns off (or on) refresh of memory based windows

7.73 QUIT

Exit the program.

Syntax:

QUIT

Example:

>QUIT Exit the application

Figure 7-4: Exit Dialog

7.74 R - Use Register Files

The R command open a processor's register files (sold separately by PEmicro) and starts interactive setup of such system registers as the I/O, timer and COP.

Entering this command opens the register files window, which initially shows a list of register files. Selecting a file brings up a display of values and significance for each bit of the register.

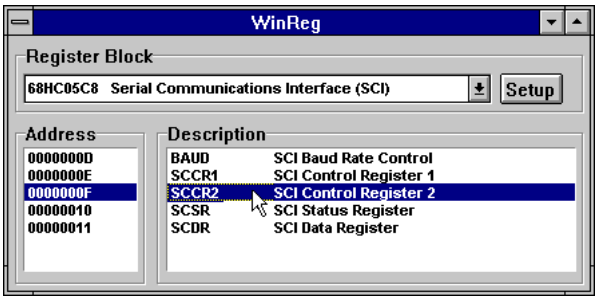


Figure 7-5: Register Editor Dialog

The user can view any of the registers, modify their values, and store the results back into Debugger memory. This is a good tool for gaining quick information on a register.

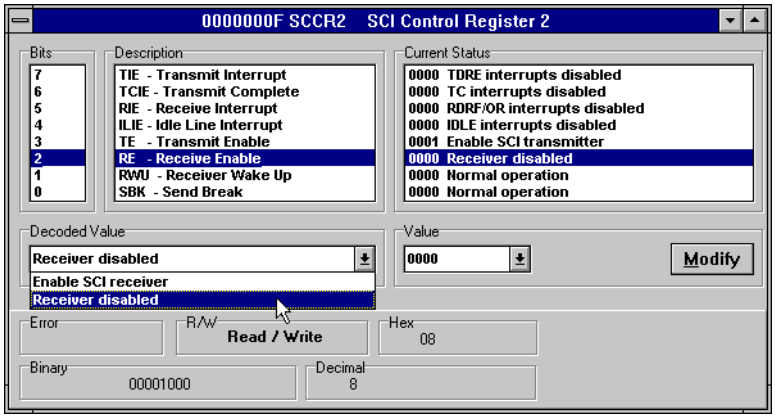


Figure 7-6: Register Editor Dialog

An alternate way to bring up the register files window is to press the Register button.

Syntax:

R

Example:

>R Start interactive system register setup.

7.75 REG or STATUS - Show Registers

The REG command displays the contents of the CPU registers in the status window. This is useful for logging CPU values while running macro files. The STATUS command is identical to the REG command.

Syntax:

REG

Example:

>REG Displays the contents of the CPU registers.

7.76 REM - Place Comment in Macro File

The REM command allows a user to display comments in a macro file. When the macro file is executing, the comment appears in the status window. The text parameter does not need to be enclosed in quotes.

Syntax:

REM <text>

Where:

<text> A comment to be displayed when a macro file is executing.

Example:

>REM Program executing Display message "Program executing" during macro file execution.

7.77 RESET - Reset Emulation MCU

The RESET command simulates a reset of the MCU and sets the program counter to the contents of the reset vector. This command does not start execution of user code.

Syntax:

RESET

Example:

>RESET Reset the MCU.

7.78 S - Set/Clear S-Bit

The S command sets or clears (that is, assigns 0 or 1 to) the S bit in the condition code register (CCR). The S Control Bit disables the STOP instruction.

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

S 0|1

Examples:

>S 0	Clear the S bit of the CCR.
>S 1	Set the S bit of the CCR.

7.79 SAVEDESK - Save Desktop Settings

The SAVEDESK command saves the desktop settings for the application when it is first opened or for use with the LOADDESK command. The settings saved are window position, size, visibility, etc.

Syntax:

SAVEDESK

Example:

>SAVEDESK	Save window settings for the application.

7.80 SERIAL

Sets up parameters for serial port. This port may then be attached to the Serial Port on your target for real-time debugging of communications software. See SERIALON command. COM1 or COM2, baud = 9600, 4800, 2400, 1200, 600, 300, 150 or 110, parity = N, E or O, data bits = 7 or 8, stop bits = 1 or 2. Example: SERIAL 1 9600 n 8 1

Syntax:

SERIAL (1 or 2) (baud) (parity) (data bits) (stop bits)

Where:

1 or 2	COM1 or COM2
baud	Baud rate ranging from 110 to 9600
parity	No, Even or Odd parity
data bits	7 or 8 data bits
stop bits	1 or 2 stop bits

Example:

>SERIAL 2 9600 E 8 2	Sets serial port to Com2 port with 9600 baud rate, even parity, 8 data bits and 2 stop bits

7.81 SERIALOFF

Turns off serial port use during GO.

Syntax:

SERIALOFF

Example:

>SERIALOFFTurns off serial port use during GO command

7.82 SERIALON

Turns the communication window into a dumb terminal during a GO command using the serial port set up with the SERIAL command. To terminate the GO command from the keyboard, hit F1.

Syntax:

SERIALON

Example:

>SERIAL 2 9600 N 8 1

>SERIALON

>GO

7.83 SHOWCODE - Display Code at Address

The SHOWCODE command displays code in the code windows beginning at the specified address, without changing the value of the program counter (PC). The code window shows either source code or disassembly from the given address, depending on which mode is selected for the window. This command is useful for browsing through various modules in the program. To return to code where the PC is pointing, use the SHOWPC command.

Syntax:

SHOWCODE <address>

Where:

<address> The address or label where code is to be shown.

Example:

>SHOWCODE 200 Show code starting at hex location 200.

7.84 SHOWMAP or MAP - Show Information in Map File

The SHOWMAP command enables the user to view information from the current MAP file stored in the memory. All symbols defined in the source code used for debugging will be listed. The debugger defined symbols, defined with the SYMBOL command, will not be shown. (The MAP command is identical to the SHOWMAP command.)

Syntax:

SHOWMAP

Example:

>SHOWMAP Shows symbols from the loaded map file and their values.

7.85 SHOWPC - Display Code at PC

The SHOWPC command displays code in the code window starting from the address in the program counter (PC). The code window shows either source code or disassembly from the given address, depending on which mode is selected for the window. This command is often useful immediately after the SHOWCODE command.

Syntax:

SHOWPC

Example:

>SHOWPC Show code from the PC address value.

7.86 SHOWTRACE

After executing the TRACE command, which monitors the execution of the CPU and logs the

address of (up to) the last 256 instructions that have been executed into an internal array, the SHOWTRACE command (or F7) allows the user to view this trace buffer.

Syntax:

SHOWTRACE

Example:

>SHOWTRACE	Displays the trace buffer logged during a previously executed TRACE command.

7.87 SIGNLATCH Command

The SIGNLATCH command sets the value (0 or 1) of the sign latch.

Syntax:

SIGNLATCH n

Where:

n = 0 or 1

Example:

>SIGNLATCH 1	Sets the sign latch to 1.

7.88 SK - Set Stack Extension Register

The SK command assigns a value to SK [3:0], the Stack Extension Register.

Syntax:

SK n

Where:

n = 0-F

Examples:

>SK $D	Sets SK to 1101 (SK[0]=1 SK[1]=1 SK[2]=0 SK[3]=1)
>SK 1101B	Sets SK to 1101 (SK[0]=1 SK[1]=1 SK[2]=0 SK[3]=1)

7.89 SNAPSHOT

Takes a snapshot (black and white) of the current screen and sends it to the capture file if one exists. Can be used for test documentation and system testing.

Syntax:

SNAPSHOT

Example:

>LOGFILE SNAPSHOT	This command will open a file by the name SNAPSHOT.LOG and stores all the command at the status window.
>SNAPSHOT	This command will take a snapshort of all the open windows of ICD and store it in SNAPSHOT.LOG file.
>LF	This command will close SNAPSHOT.LOG file

Now you can open the SNAPSHOT.LOG file with any text editor, such as EDIT.

7.90 SOURCE

If a valid map file has been loaded, the SOURCE command will toggle between showing actual source code and disassembled code.

Syntax:

SOURCE

Example:

>SOURCE Toggles between source code and disassembled code in debug window.

7.91 SOURCEPATH

Either uses the specified filename or prompts the user for the path to search for source code that is not present in the current directory.

Syntax:

SOURCEPATH filename

Where:

filename Name of the source file

Example:

>SOURCEPATH d:\mysource\myfile.asm

7.92 SP - Set Stack Pointer

The SP command sets the Stack Pointer to a specified value

Syntax:

SP <n>

Where:

<n> The value to be loaded into the Stack Pointer.

Example:

>SP $FF Set the Stack Pointer to hex FF.

7.93 SS

Does one step of source level code. Source must be showing in the code window.

Syntax:

SS

Example:

>SS Does one step of source level code.

7.94 ST or STEP or T - Single Step

The ST or STEP or T command steps through one or a specified number of assembly instructions, beginning at the current Program Counter (PC) address value, and then halts. When the number argument is omitted, one instruction is executed. If you enter the ST command with an <n> value, the command steps through that many instructions.

Syntax:

STEP <n>

or

ST <n>

or

T <n>

Where:

<n> The hexadecimal number of instructions to be executed by each command.

Example:

>STEP Execute the assembly instruction at the PC address value.

>ST 2 Execute two assembly instructions, starting at the PC address value.

7.95 STEPFOR - Step Forever

STEPFOR command continuously executes instructions, one at a time, beginning at the current Program Counter address until an error condition occurs, a breakpoint occurs, or a key or mouse is pressed. All windows are refreshed as each instruction is executed.

Syntax:

STEPFOR

Example:

>STEPFOR Step through instructions continuously.

7.96 STEPTIL - Single Step to Address

The STEPTIL command continuously steps through instructions beginning at the current Program Counter (PC) address until the PC value reaches the specified address. Execution also stops if a key or mouse is pressed, a breakpoint set with a BR command occurs, or an error occurs.

Syntax:

STEPTIL <address>

Where:

<address> Execution stop address. This must be an instruction address.

Example:

>STEPTIL 0400 Execute instructions continuously until PC hex value is 0400.

7.97 SYMBOL - Add Symbol

The SYMBOL command creates a new symbol, which can be used anywhere in the debugger, in place of the symbol value. If this command is entered with no parameters, it will list the current user defined symbols. If parameters are specified, the SYMBOL command will create a new symbol.

The symbol label is case insensitive and has a maximum length of 16T. It can be used with the ASM and MM command, and replaces all addresses in the Code Window (when displaying disassembly) and Variables Window.

The command has the same effect as an EQU statement in the assembler.

Syntax:

SYMBOL [<label> <value>]

Where:

<label> The ASCII-character string label of the new symbol.

<value> The value of the new symbol (label).

Examples:

>SYMBOL Show the current user-defined symbols.

>SYMBOL timer_control $08 Define new symbol 'timer_control', with hex value 08. Subsequently, to modify hex location 08, enter the command 'MM timer_control'.

7.98 TIME

Will give you an estimate of real time to execute the command from one address to another.

Set breakpoint at second address. Go from first address. If only one address given, it is the start address. If no stop address is given, the ICD will run forever or until a breakpoint is encountered or a key on the keyboard is hit. If no address is given the command is a "Time forever" command. When the command ends (either a break or a key) the debug window will show the amount of real-time that passed since the command was initiated.

Syntax:

TIME <[add1] [add2]>

Where:

add1	Starting address
add2	Ending address

Example:

>TIME 800 805	Will give you an estimate of real time to execute the command from hex location 800 to hex location 805.

7.99 TRACE

The TRACE command is similar to the GO command except that execution does not occur in real-time. The ICD software monitors the execution of the CPU and logs the address of (up to) the last 256 instructions that have been executed into an internal array .

The trace executes from the first address until the breakpoint at the second address. If only one address given, it is the start address. If no stop address is given, the ICD will run forever or until a breakpoint is reached or a key on the keyboard is hit. If no address is given the command is a "Trace forever" command.

After execution, you may use the SHOWTRACE command or hit F7 to view the trace buffer.

Syntax:

TRACE <[add1] [add2]>

Where:

add1	Starting address
add2	Ending address

Example:

>TRACE 800 805	Will give you an estimate of real time to execute the command from hex location 800 to hex location 805.

7.100 UPLOAD_SREC - Upload S-Record to Screen

The UPLOAD_SREC command uploads the content of the specified program memory block (range), in .S19 object file format, displaying the contents in the status window. If a log file is opened, then UPLOAD_SREC will put the information into it as well. Same as P_UPLOAD_SREC.

Note:

If the UPLOAD_SREC command is entered, sometimes the memory contents scroll through the debug window too rapidly to view. Accordingly, either the LOGFILE command should be used, which records the contents into a file, or use the scroll bars in the status window.

Syntax:

UPLOAD_SREC <startrange> <endrange>

Where:

<startrange>	Beginning address of the memory block.
<endrange>	Ending address of the memory block (range)

Example:

>UPLOAD_SREC 300 7FF	Upload the 300□7FF memory block in .S19 format.

7.101 V - Set/Clear V-Bit

The V command sets or clears (that is, assigns 0 or 1 to) the V bit in the condition code register (CCR).

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is

S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

V 0|1

Examples:

>V 0	Clear the V bit of the CCR.
>V 1	Set the V bit of the CCR.

7.102 VAR - Display Variable

The VAR command displays the specified address and its contents in the Variables Window for viewing during code execution. Variants of the command display a byte, a word, a long, or a string. As the value at the address changes, the variables window updates the value. The maximum number of variables is 32. You may also enter the requisite information using the Add Variable box, which may be called up by double-clicking on the Variables Window or executing the VAR command without a parameter.

In the ASCII displays, a control character or other non-printing character is displayed as a period (.). The byte, word, long, or string variant determines the display format:

- Byte (.B): hexadecimal (the default)
- Word (.W): hexadecimal
- Long (.L): hexadecimal
- String (.S): ASCII characters

To change the format from the default of hexdecimal, use the Add Variable box.

The optional <n> parameter specifies the number of string characters to be displayed; the default value is one. The <n> parameter has no effect for byte, word, or long values.

Syntax:

VAR [.B|.W|.L|.S] <address> [<n>]

Where:

<address>	The address of the memory variable.
<n>	Optional number of characters for a string variable; default value is 1, does not apply to byte or word variables.

Examples:

>VAR C0	Show byte value of address C0 (hex and binary)
>VAR.B D4	Show byte value of address D4 (hex and binary)
>VAR.W E0	Show word value of address E0 (hex & decimal)
>VAR.S C0 5	Show the five-character ASCII string at hex address C0.

7.103 VERIFY

Compares the contents of program memory with an S-record file. You will be prompted for the name of the file. The comparisons will stop at the first location with a different value.

Syntax:

VERIFY

Example:

>LOADALL test.s19

>VERIFY As soon as you press <ENTER> key it will give you a message "Verifying...verified"

7.104 VERSION or VER - Display Software Version

The VERSION command displays the version and date of the software. VER is an alternate form of this command.

Syntax:

VERSION

Examples:

>VERSION Display debugger version.

>VER Display debugger version.

7.105 WATCHDOG

Disables watchdog timer (toggles the state of the SWE bit in the SYPCR). Remember that this register may only be written once following a reset of the hardware. Reset enables the watchdog timer.

Syntax:

WATCHDOG

Example:

>WATCHDOG

7.106 WHEREIS - Display Symbol Value

The WHEREIS command displays the value of the specified symbol. Symbol names are defined through source code or the SYMBOL command.

Syntax:

WHEREIS <symbol> | <address>

Where:

<symbol> A symbol listed in the symbol table.

<address> Address for which a symbol is defined.

Examples:

>WHEREIS START Display the symbol START and its value.

>WHEREIS 0300 Display the hex value 0300 and its symbol name if any.

7.107 XK - Set XK Extension Register

The XK command allows the user to set the value of the XK Extension Register.

Syntax:

XK n

Where:

n = value from 0-F

Example:

XK 6 Sets value of XK Register to 6.

7.108 XMASK

The XMASK command allows the user to assign a value to the XMSK Register in the MAC unit.

Syntax:

XMASK <n>

Where:

<n>	Value to assign to XMSK Register.

Example:

XMASK $7C	Assigns 7C to XMSK Register.

7.109 Y Command - Set Y Index Register

The Y command sets the index register (Y) to the specified value. The Y command is identical to the YREG command.

Syntax:

Y <value>

Where:

<value>	The new value for the Y register.

Example:

>Y 05	Set the index register value hex to 05.
>YREG F0	Set the index register value hex to F0.

7.110 Y MASK

The YMASK command allows the user to assign a value to the YMSK Register in the MAC unit.

Syntax:

YMASK <n>

Where:

<n>	Value to assign to YMSK Register.

Example:

YMASK $54	Assigns 54 to YMSK Register.

7.111 YK - Set YK Extension Register

The YK command allows the user to set the value of the YK Extension Register.

Syntax:

YK n

Where:

n = value from 0-F

Example:

YK $C	Sets value of YK Register to $C.

7.112 Z - Set/Clear Z-Bit

The Z command sets or clears (that is, assigns 0 or 1 to) the Z bit in the condition code register (CCR).

Note:

The CCR bit designators are at the lower right of the CPU window. The CCR pattern is S,MV,H,EV,N,Z,V,C,IP,SM,PK. S is STOP control, MV is an overflow flag, H is half-carry, EV is an overflow flag, N is negative, Z is zero, V is overflow, C is carry, IP is a masking interrupt, SM is saturate mode, and PK is a program counter extension. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

MV, EV, and SM cannot be set by the user, but a value should be used as a placeholder when writing to the Condition Code Register.

Syntax:

Z 0|1

Examples:

>Z 0	Clear the Z bit of the CCR.
>Z 1	Set the Z bit of the CCR.

7.113 ZK - Set ZK Extension Register

The ZK command allows the user to set the value of the ZK Extension Register.

Syntax:

ZK n

Where:

n = value from 0-F

Example:

ZK 9	Sets value of ZK Register to 9.

8 CPU Values & Names

CPU Values:

Any on-screen location with an alphabetic name (SP, PC, etc.) may be changed by entering the name of the location followed by a value and <ENTER> key; for example:

>A 44

This will change the value of accumulator A to hex value 44.

>E FF04

This will change the value of E register to hex value FF04

>PC 100

The value of Program Counter register (PC) will be changed to hex value 100

>SP 3FF

This will set the Stack Pointer register to hex value 3FF.

To change the value of the AM register, refer to either AMH (AM High) for AM bits 35-16 and AML (AM Low) for bits 15-0.

Example:

AMH 1F321

AML F033

CPU Names:

A	AMH	AML	B	C
CCR	D	E	EK	H
HR	IR	IX	IY	IZ
K	N	PC	PK	S
SIGNLATCH	SK	SP	V	XK
XMASK	YK	YMASK	Z	ZK

9 Source Level Debugging

Once a valid map file (generated by CASM16Z) is loaded into the ICD via the LOADMAP or LOADALL command, source level debugging is enabled. When the IP is at a location for which

there is source code available, the source code will be shown in the code window. The user can set a breakpoint by using the BR command, or by clicking the appropriate line in the code window, then clicking the right mouse button and selecting “Toggle a Breakpoint.” For more details, see the Code Window section.

10 Errors

Various errors may appear in the DEBUG F1 window during your debugging. Most are self explanatory. The following four errors are due to the debugger's interface with the background mode of the CPU16.

Debugger supplied DSACK	Probable memory implementation error!
Warning	Not ready response from chip.
Warning	BERR Terminated bus cycle - Debugger Supplied DSACK
Warning	Illegal command error from chip - Debugger Supplied DSACK

All four errors are probably due to some type of memory problem involving the DSACK signal. In most cases, the Debugger will assert DSACK to end a bus cycle that is taking much too long.

Note: The debugger rewrites the windows showing on the screen often. If a window is showing memory that does not exist, one of these errors will occur every time the debugger tries to update that window. This concerns the two memory windows and the code window. Additionally, reading or writing non-existing memory areas mapped internally may cause one of these errors.

As stated, in most cases the debugger will supply DSACK and recover. If the system starts acting erratic after this message, it may be due to a fatal memory error and you may have to reset the system.

11 Instruction Set

In the instruction format shown below, the "^" symbol stands for a parameter which reduces to a number. All numbers are tested to determine if they fit in the space provided in the instruction. Parameters which are specified as signed such as offsets or signed integers must lie in the appropriate range. For example, signed 16 bit offsets must be between -32768 and 32767 decimal or equivalently between -08000 and 7FFF in hexadecimal.

ABA	ABX	ABY	ABZ
ACE	ACED	ADCA #^	ADCA E,X
ADCA E,Y	ADCA E,Z	ADCA ^	ADCA ^,X
ADCA ^,X	ADCA ^,Y	ADCA ^,Y	ADCA ^,Z
ADCA ^,Z	ADCB #^	ADCB E,X	ADCB E,Y
ADCB E,Z	ADCB ^	ADCB ^,X	ADCB ^,X
ADCB ^,Y	ADCB ^,Y	ADCB ^,Z	ADCB ^,Z
ADCD #^	ADCD E,X	ADCD E,Y	ADCD E,Z
ADCD ^	ADCD ^,X	ADCD ^,X	ADCD ^,Y
ADCD ^,Y	ADCD ^,Z	ADCD ^,Z	ADCE #^
ADCE ^	ADCE ^,X	ADCE ^,Y	ADCE ^,Z
ADDA #^	ADDA E,X	ADDA E,Y	ADDA E,Z
ADDA ^	ADDA ^,X	ADDA ^,X	ADDA ^,Y
ADDA ^,Y	ADDA ^,Z	ADDA ^,Z	ADDB #^
ADDB E,X	ADDB E,Y	ADDB E,Z	ADDB ^

ADDB ^,X	ADDB ^,X	ADDB ^,Y	ADDB ^,Y
ADDB ^,Z	ADDB ^,Z	ADDD #^	ADDD #^
ADDD E,X	ADDD E,Y	ADDD E,Z	ADDD ^
ADDD ^,X	ADDD ^,X	ADDD ^,Y	ADDD ^,Y
ADDD ^,Z	ADDD ^,Z	ADDE #^	ADDE #^
ADDE ^	ADDE ^,X	ADDE ^,Y	ADDE ^,Z
ADE	ADX	ADY	ADZ
AEX	AEY	AEZ	AIS #^
AIS #^	AIX #^	AIX #^	AIY #^
AIY #^	AIZ #^	AIZ #^	ANDA #^
ANDA E,X	ANDA E,Y	ANDA E,Z	ANDA ^
ANDA ^,X	ANDA ^,X	ANDA ^,Y	ANDA ^,Y
ANDA ^,Z	ANDA ^,Z	ANDB #^	ANDB E,X
ANDB E,Y	ANDB E,Z	ANDB ^	ANDB ^,X
ANDB ^,X	ANDB ^,Y	ANDB ^,Y	ANDB ^,Z
ANDB ^,Z	ANDD #^	ANDD E,X	ANDD E,Y
ANDD E,Z	ANDD ^	ANDD ^,X	ANDD ^,X
ANDD ^,Y	ANDD ^,Y	ANDD ^,Z	ANDD ^,Z
ANDE #^	ANDE ^	ANDE ^,X	ANDE ^,Y
ANDE ^,Z	ANDP #^	ASL ^	ASL ^,X
ASL ^,X	ASL ^,Y	ASL ^,Y	ASL ^,Z
ASL ^,Z	ASLA	ASLB	ASLD
ASLE	ASLM	ASLW ^	ASLW ^,X
ASLW ^,Y	ASLW ^,Z	ASR ^	ASR ^,X
ASR ^,X	ASR ^,Y	ASR ^,Y	ASR ^,Z
ASR ^,Z	ASRA	ASRB	ASRD
ASRE	ASRM	ASRW ^	ASRW ^,X
ASRW ^,Y	ASRW ^,Z	BCC ^	BCLR ^,X,^
BCLR ^,X,^	BCLR ^,Y,^	BCLR ^,Y,^	BCLR ^,Z,^
BCLR ^,Z,^	BCLR ^,^	BCLRW ^,X,^	BCLRW ^,Y,^
BCLRW ^,Z,^	BCLRW ^,^	BCS ^	BEQ ^
BGE ^	BGND	BGT ^	BHI ^
BITA #^	BITA E,X	BITA E,Y	BITA E,Z
BITA ^	BITA ^,X	BITA ^,X	BITA ^,Y
BITA ^,Y	BITA ^,Z	BITA ^,Z	BITB #^
BITB E,X	BITB E,Y	BITB E,Z	BITB ^
BITB ^,X	BITB ^,X	BITB ^,Y	BITB ^,Y
BITB ^,Z	BITB ^,Z	BLE ^	BLS ^
BLT ^	BMI ^	BNE ^	BPL ^
BRA ^	BRCLR ^,X,^,^	BRCLR ^,X,^,^	BRCLR ^,Y,^,^
BRCLR ^,Y,^,^	BRCLR ^,Z,^,^	BRCLR ^,Z,^,^	BRCLR ^,^,^
BRN ^	BRSET ^,X,^,^	BRSET ^,X,^,^	BRSET ^,Y,^,^
BRSET ^,Y,^,^	BRSET ^,Z,^,^	BRSET ^,Z,^,^	BRSET ^,^,^
BSET ^,X,^	BSET ^,X,^	BSET ^,Y,^	BSET ^,Y,^

BSET ^,Z,^	BSET ^,Z,^	BSET ^,^	BSETW ^,X,^
BSETW ^,Y,^	BSETW ^,Z,^	BSETW ^,^	BSR ^
BVC ^	BVS ^	CBA	CLR ^
CLR ^,X	CLR ^,X	CLR ^,Y	CLR ^,Y
CLR ^,Z	CLR ^,Z	CLRA	CLRB
CLRD	CLRE	CLRM	CLRW ^
CLRW ^,X	CLRW ^,Y	CLRW ^,Z	CMPA #^
CMPA E,X	CMPA E,Y	CMPA E,Z	CMPA ^
CMPA ^,X	CMPA ^,X	CMPA ^,Y	CMPA ^,Y
CMPA ^,Z	CMPA ^,Z	CMPB #^	CMPB E,X
CMPB E,Y	CMPB E,Z	CMPB ^	CMPB ^,X
CMPB ^,X	CMPB ^,Y	CMPB ^,Y	CMPB ^,Z
CMPB ^,Z	COM ^	COM ^,X	COM ^,X
COM ^,Y	COM ^,Y	COM ^,Z	COM ^,Z
COMA	COMB	COMD	COME
COMW ^	COMW ^,X	COMW ^,Y	COMW ^,Z
CPD #^	CPD E,X	CPD E,Y	CPD E,Z
CPD ^	CPD ^,X	CPD ^,X	CPD ^,Y
CPD ^,Y	CPD ^,Z	CPD ^,Z	CPE #^
CPE ^	CPE ^,X	CPE ^,Y	CPE ^,Z
CPS #^	CPS ^	CPS ^,X	CPS ^,X
CPS ^,Y	CPS ^,Y	CPS ^,Z	CPS ^,Z
CPX #^	CPX ^	CPX ^,X	CPX ^,X
CPX ^,Y	CPX ^,Y	CPX ^,Z	CPX ^,Z
CPY #^	CPY ^	CPY ^,X	CPY ^,X
CPY ^,Y	CPY ^,Y	CPY ^,Z	CPY ^,Z
CPZ #^	CPZ ^	CPZ ^,X	CPZ ^,X
CPZ ^,Y	CPZ ^,Y	CPZ ^,Z	CPZ ^,Z
DAA	DEC ^	DEC ^,X	DEC ^,X
DEC ^,Y	DEC ^,Y	DEC ^,Z	DEC ^,Z
DECA	DECB	DECW ^	DECW ^,X
DECW ^,Y	DECW ^,Z	EDIV	EDIVS
EMUL	EMULS	EORA #^	EORA E,X
EORA E,Y	EORA E,Z	EORA ^	EORA ^,X
EORA ^,X	EORA ^,Y	EORA ^,Y	EORA ^,Z
EORA ^,Z	EORB #^	EORB E,X	EORB E,Y
EORB E,Z	EORB ^	EORB ^,X	EORB ^,X
EORB ^,Y	EORB ^,Y	EORB ^,Z	EORB ^,Z
EORD #^	EORD E,X	EORD E,Y	EORD E,Z
EORD ^	EORD ^,X	EORD ^,X	EORD ^,Y
EORD ^,Y	EORD ^,Z	EORD ^,Z	EORE #^
EORE ^	EORE ^,X	EORE ^,Y	EORE ^,Z
FDIV	FMULS	IDIV	INC ^
INC ^,X	INC ^,X	INC ^,Y	INC ^,Y

INC ^,Z	INC ^,Z	INCA	INCB
INCW ^	INCW ^,X	INCW ^,Y	INCW ^,Z
JMP ^	JMP ^,X	JMP ^,Y	JMP ^,Z
JSR ^	JSR ^,X	JSR ^,Y	JSR ^,Z
LBCC ^	LBCS ^	LBEQ ^	LBEV ^
LBGE ^	LBGT ^	LBHI ^	LBLE ^
LBLS ^	LBLT ^	LBMI ^	LBMV ^
LBNE ^	LBPL ^	LBRA ^	LBRN ^
LBSR ^	LBVC ^	LBVS ^	LDAA #^
LDAA E,X	LDAA E,Y	LDAA E,Z	LDAA ^
LDAA ^,X	LDAA ^,X	LDAA ^,Y	LDAA ^,Y
LDAA ^,Z	LDAA ^,Z	LDAB #^	LDAB E,X
LDAB E,Y	LDAB E,Z	LDAB ^	LDAB ^,X
LDAB ^,X	LDAB ^,Y	LDAB ^,Y	LDAB ^,Z
LDAB ^,Z	LDD #^	LDD E,X	LDD E,Y
LDD E,Z	LDD ^	LDD ^,X	LDD ^,X
LDD ^,Y	LDD ^,Y	LDD ^,Z	LDD ^,Z
LDE #^	LDE ^	LDE ^,X	LDE ^,Y
LDE ^,Z	LDED ^	LDHI	LDS #^
LDS ^	LDS ^,X	LDS ^,X	LDS ^,Y
LDS ^,Y	LDS ^,Z	LDS ^,Z	LDX #^
LDX ^	LDX ^,X	LDX ^,X	LDX ^,Y
LDX ^,Y	LDX ^,Z	LDX ^,Z	LDY #^
LDY ^	LDY ^,X	LDY ^,X	LDY ^,Y
LDY ^,Y	LDY ^,Z	LDY ^,Z	LDZ #^
LDZ ^	LDZ ^,X	LDZ ^,X	LDZ ^,Y
LDZ ^,Y	LDZ ^,Z	LDZ ^,Z	LPSTOP
LSR ^	LSR ^,X	LSR ^,X	LSR ^,Y
LSR ^,Y	LSR ^,Z	LSR ^,Z	LSRA
LSRB	LSRD	LSRE	LSRW ^
LSRW ^,X	LSRW ^,Y	LSRW ^,Z	MAC ^,^
MOVB ,X(^) ^	MOVB ^ ,X(^)	MOVB ^,^	MOVW ,X(^) ^
MOVW ^ ,X(^)	MOVW ^,^	MUL	NEG ^
NEG ^,X	NEG ^,X	NEG ^,Y	NEG ^,Y
NEG ^,Z	NEG ^,Z	NEGA	NEGB
NEGD	NEGE	NEGW ^	NEGW ^,X
NEGW ^,Y	NEGW ^,Z	NOP	ORAA #^
ORAA E,X	ORAA E,Y	ORAA E,Z	ORAA ^
ORAA ^,X	ORAA ^,X	ORAA ^,Y	ORAA ^,Y
ORAA ^,Z	ORAA ^,Z	ORAB #^	ORAB E,X
ORAB E,Y	ORAB E,Z	ORAB ^	ORAB ^,X
ORAB ^,X	ORAB ^,Y	ORAB ^,Y	ORAB ^,Z
ORAB ^,Z	ORD #^	ORD E,X	ORD E,Y
ORD E,Z	ORD ^	ORD ^,X	ORD ^,X

ORD ^,Y	ORD ^,Y	ORD ^,Z	ORD ^,Z
ORE #^	ORE ^	ORE ^,X	ORE ^,Y
ORE ^,Z	ORP #^	PSHA	PSHB
PSHM ^	PSHMAC	PULA	PULB
PULM ^	PULMAC	RMAC ^,^	ROL ^
ROL ^,X	ROL ^,X	ROL ^,Y	ROL ^,Y
ROL ^,Z	ROL ^,Z	ROLA	ROLB
ROLD	ROLE	ROLW ^	ROLW ^,X
ROLW ^,Y	ROLW ^,Z	ROR ^	ROR ^,X
ROR ^,X	ROR ^,Y	ROR ^,Y	ROR ^,Z
ROR ^,Z	RORA	RORB	RORD
RORE	RORW ^	RORW ^,X	RORW ^,Y
RORW ^,Z	RTI	RTS	SBA
SBCA #^	SBCA E,X	SBCA E,Y	SBCA E,Z
SBCA ^	SBCA ^,X	SBCA ^,X	SBCA ^,Y
SBCA ^,Y	SBCA ^,Z	SBCA ^,Z	SBCB #^
SBCB E,X	SBCB E,Y	SBCB E,Z	SBCB ^
SBCB ^,X	SBCB ^,X	SBCB ^,Y	SBCB ^,Y
SBCB ^,Z	SBCB ^,Z	SBCD #^	SBCD E,X
SBCD E,Y	SBCD E,Z	SBCD ^	SBCD ^,X
SBCD ^,X	SBCD ^,Y	SBCD ^,Y	SBCD ^,Z
SBCD ^,Z	SBCE #^	SBCE ^	SBCE ^,X
SBCE ^,Y	SBCE ^,Z	SDE	STAA E,X
STAA E,Y	STAA E,Z	STAA ^	STAA ^,X
STAA ^,X	STAA ^,Y	STAA ^,Y	STAA ^,Z
STAA ^,Z	STAB E,X	STAB E,Y	STAB E,Z
STAB ^	STAB ^,X	STAB ^,X	STAB ^,Y
STAB ^,Y	STAB ^,Z	STAB ^,Z	STD E,X
STD E,Y	STD E,Z	STD ^	STD ^,X
STD ^,X	STD ^,Y	STD ^,Y	STD ^,Z
STD ^,Z	STE ^	STE ^,X	STE ^,Y
STE ^,Z	STED ^	STS ^	STS ^,X
STS ^,X	STS ^,Y	STS ^,Y	STS ^,Z
STS ^,Z	STX ^	STX ^,X	STX ^,X
STX ^,Y	STX ^,Y	STX ^,Z	STX ^,Z
STY ^	STY ^,X	STY ^,X	STY ^,Y
STY ^,Y	STY ^,Z	STY ^,Z	STZ ^
STZ ^,X	STZ ^,X	STZ ^,Y	STZ ^,Y
STZ ^,Z	STZ ^,Z	SUBA #^	SUBA E,X
SUBA E,Y	SUBA E,Z	SUBA ^	SUBA ^,X
SUBA ^,X	SUBA ^,Y	SUBA ^,Y	SUBA ^,Z
SUBA ^,Z	SUBB #^	SUBB E,X	SUBB E,Y
SUBB E,Z	SUBB ^	SUBB ^,X	SUBB ^,X
SUBB ^,Y	SUBB ^,Y	SUBB ^,Z	SUBB ^,Z

SUBD #^	SUBD E,X	SUBD E,Y	SUBD E,Z
SUBD ^	SUBD ^,X	SUBD ^,X	SUBD ^,Y
SUBD ^,Y	SUBD ^,Z	SUBD ^,Z	SUBE #^
SUBE ^	SUBE ^,X	SUBE ^,Y	SUBE ^,Z
SWI	SXT	TAB	TAP
TBA	TBEK	TBSK	TBXK
TBYK	TBZK	TDE	TDMSK
TDP	TED	TEDM	TEKB
TEM	TMER	TMET	TMXED
TPA	TPD	TSKB	TST ^
TST ^,X	TST ^,X	TST ^,Y	TST ^,Y
TST ^,Z	TST ^,Z	TSTA	TSTB
TSTD	TSTE	TSTW ^	TSTW ^,X
TSTW ^,Y	TSTW ^,Z	TSX	TSY
TSZ	TXKB	TXS	TXY
TXZ	TYKB	TYS	TYX
TYZ	TZKB	TZS	TZX
TZY	WAI	XGAB	XGDE
XGDX	XGDY	XGDZ	XGEX
XGEY	XGEZ		

12 Running

Sometimes it is desirable to leave the CPU running and exit the ICD debug software. To do this, use the GOEXIT command. To re-enter the ICD debug software, use the option RUNNING as a parameter on the start up command line (see STARTUP). This option causes the debugger NOT to do a RESET at startup and to ignore any STARTUP.ICD macro file. In order to use this option, the CPU must have previously been left executing by the debugger.

It is possible to remove the ICD cable from your target system and then reconnect it provided that the following sequence is observed:

1) Exit the ICD by using the GOEXIT command.

2) Disconnect cable without removing power connections**.

3) Remove power from cable.

4) Do whatever...

5) Connect power to cable.

6) Start the ICD software using the RUNNING option.

7) Connect cable to your target.

** Normally the Multilink debug probe receives its power (5 volts and Ground) from the target system. You can modify the cable to maintain power when the 10 pin berg connector is removed from your target. You can do this in two ways:

a) Cut the Vdd lead in the cable and use a separate power supply or jumper the cable power back to your target. If you use a separate supply, you should maintain a common ground.

b) Attach a second insulation displacement connector to the ICD cable and jumper from it to the target power supply.

13 Disassembly Mode

In disassembly mode, the base address of the code window is the first line showing in the window when the scrollbar is at the top. Due to the nature of disassembly, you cannot scroll backwards arbitrarily, and hence you must have a starting point. This starting point is the base address. The base address can be set using the SHOWCODE command or by using the popup menu of the code window. The base address has no meaning in source-level mode unless the user tries to change it (again, refer to the showcode command).

14 'Add Variable' Box

The Add Variable box allows the user to display a variable specified by label or memory address, and to choose the format and base of the data that is displayed. The Add Variable box pops up when the user double-clicks the Variables Window.

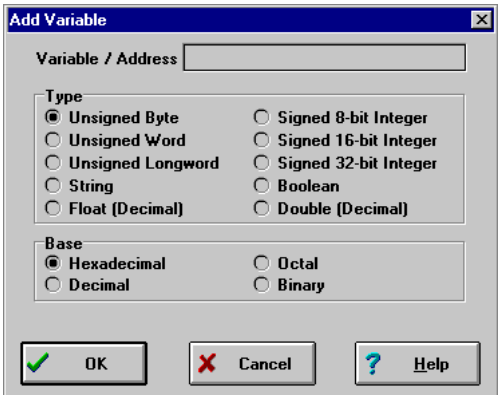


Figure 14-1: Add Variable Dialog

Use the Variable / Address input field to enter the label or memory location for the data you wish to be displayed.

Below, select from the Type and Base buttons to specify in which format and base you would like the data to de displayed.

15 Memory Access

1. When you modify bytes, words, or longs they are read/written using the corresponding background debug mode read/write.
2. When you modify program words, they are read/written using the corresponding background debug mode program space word read/write. Modifying bytes in program space is accomplished using program space reads and writes. Modifying longs in program space is done using double program space reads/writes.
3. The program memory window is read using program space word reads.
4. The data memory window is read using data space word reads.
5. The VAR window is read using the appropriate data space reads.
6. Code window data is read using program space word reads.

16 V1_RESET

The user should be aware of an aspect of the V1 part: BERR is necessary for BDM to work properly, but upon reset, the default setting for the V1 part is IRQX. The V1_RESET command

allows the user to toggle the part to BERR immediately after reset, to avoid problems with BDM. V1_RESET may be used as either a command or a command-line parameter.

17 X1_RESET

The X1_RESET command has been added to support the 68HC16X1 chip in the ICD16 debugger. This command can be used as a command line parameter or a a regular ICD16 command from the keyboard or a macro (script) file.

As of 10/95, current versions of the 68HC16X1 part do not bring out the CSBOOT line due to the small footprint of the IC package. To facilitate easy startup in the debugger (after a reset), the ICD software does the following steps when it detects a RESET condition:

1. Disables CSBOOT chip select.
2. Enables CS0 at address 0 for 64k, words, read write, supervisor-user.

NOTES:

1. The debugger detects a RESET condition by either sensing a change in the reset status register or by seeing that CSBOOT has been enabled. Hence, your code running under this option should not attempt to enable CSBOOT.
2. To terminate the X1_RESET mode, you must exit ICD16 and restart it.
3. When running on a Motorola Modular Platform Board, the pseudo ROM chip select should be connected to CS0.