

1 Introduction

PEmicro's ICD12Z is a powerful tool for debugging code. It uses an HC12 or MC9(S)12(X) processor's background debug mode (BDM), via one of PEmicro's hardware interfaces, to give the user access to all on-chip resources.

ICD12Z is a great tool for creating repeatable testing, calibration, and debugging procedures.

2 Command Line Parameters

To setup ICD12Z to run with certain command line parameters, highlight the ICD12Z icon and select PROPERTIES from the Program Manager File Menu.

Syntax:

ICD12Z [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 $1 \leq n \leq 64k$. 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.
- 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 than one option is given, they must be separated by spaces.

Examples:

ICD12Z 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.

3 Nomenclature

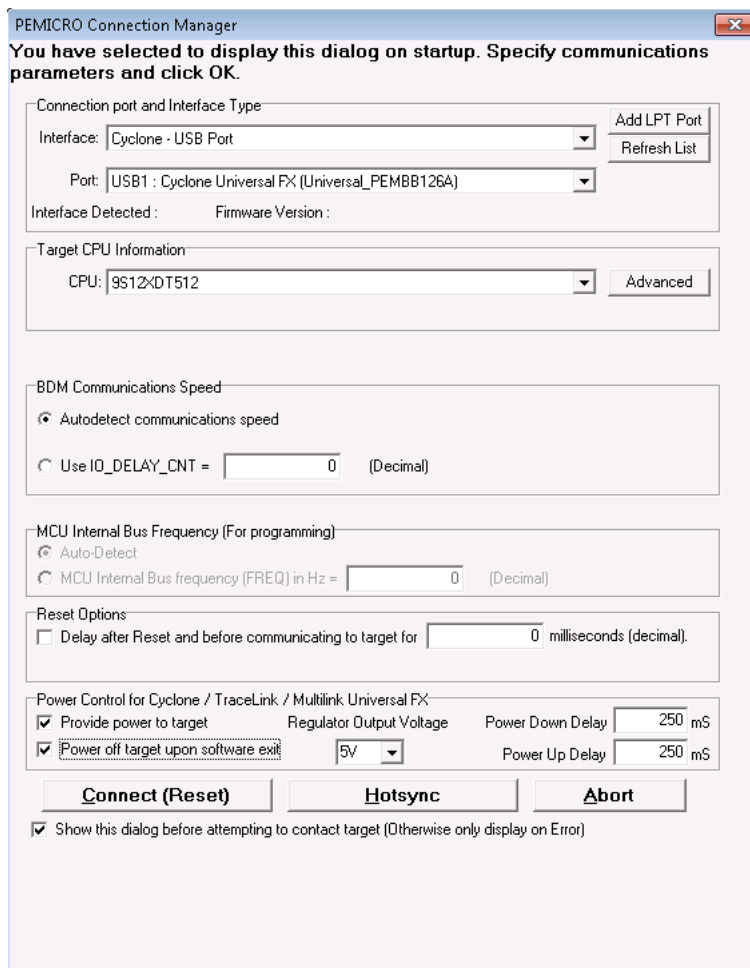
Note the following:

- n any number from 0 to 0FFFFFFF (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.
- str ASCII string.
- ; Everything on a command line after and including the “;” character is considered a comment. This helps in documenting macro (script) files.

4 Connection to Target

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

Figure 4-1: 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.

5 User Interface

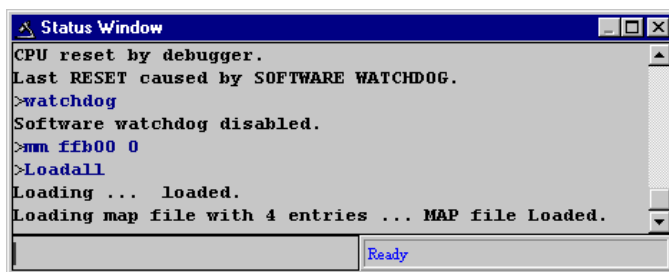
This section contains descriptions and instructions for the various windows that comprise the debugger user interface.

5.1 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.



5.1.1 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:

5.1.2 Keystrokes

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

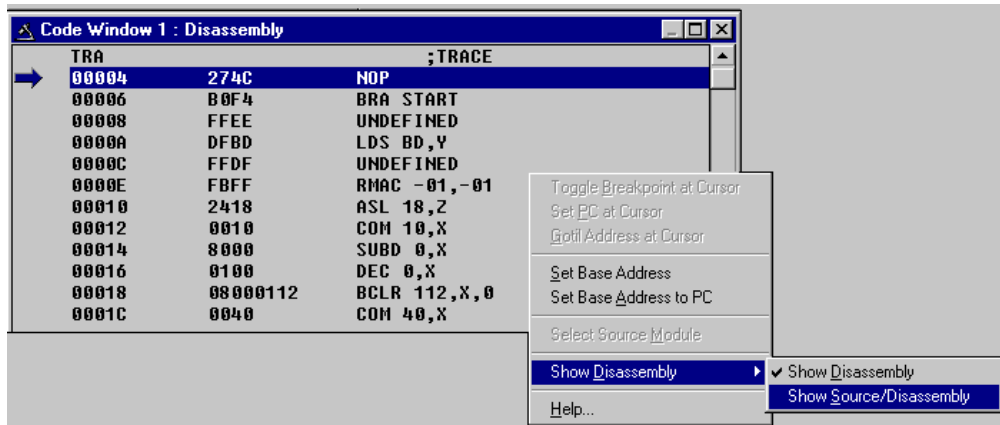
UP ARROW	Scroll window up one line
DOWN ARROW	Scroll window down one line
HOME	Scroll window to first status line
END	Scroll window to last status line
PAGE UP	Scroll window up one page
PAGE DOWN	Scroll window down one page

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

5.2 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 within 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.



5.2.1 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 within the source code, and shows disassembly otherwise.

5.2.2 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
ESC	Make the STATUS window the active window

5.2.3 Using Code Window Quick Execution Features

In the source code window, there will be a tiny red dot and a tiny blue arrow next to each source instruction that has underlying object code. If a large blue arrow is shown on a source line, this indicates that the program counter (PC) points to this instruction. If a large red stop sign appears on the source line, this indicates that a breakpoint is set on this line. A close-up of the code may be seen below:

```

// some test code. add variables to variables
- • local_int += 2;
- • date_var = Thursday;
- • date_var = PEMicroday;
- •  date_var = Sunday;
- • date_pointer = &date_var;

```

The user may set a breakpoint at an instruction by double-clicking the tiny red dot,. When the user issues the HGO command or clicks the high-level language GO button on the debugger button bar, execution will begin in real-time. If the debugger encounters a breakpoint, execution will stop on this source line. If a breakpoint is not encountered, execution will continue until the user presses a key or uses the stop button on the debugger button bar. To remove a breakpoint, double-click the large red stop sign.

By double-clicking the tiny blue arrow, the user will be issuing a GOTIL command to the address of this source line. A GOTIL command will set a single breakpoint at the desired address, and the processor will begin executing code in real-time from the point of the current program counter (PC). When the debugger encounters the GOTIL address, execution will stop. If this location is not encountered, execution will continue until the user presses a key or uses the stop button on the debugger button bar. Note that all set breakpoints are ignored when the GOTIL command is used.

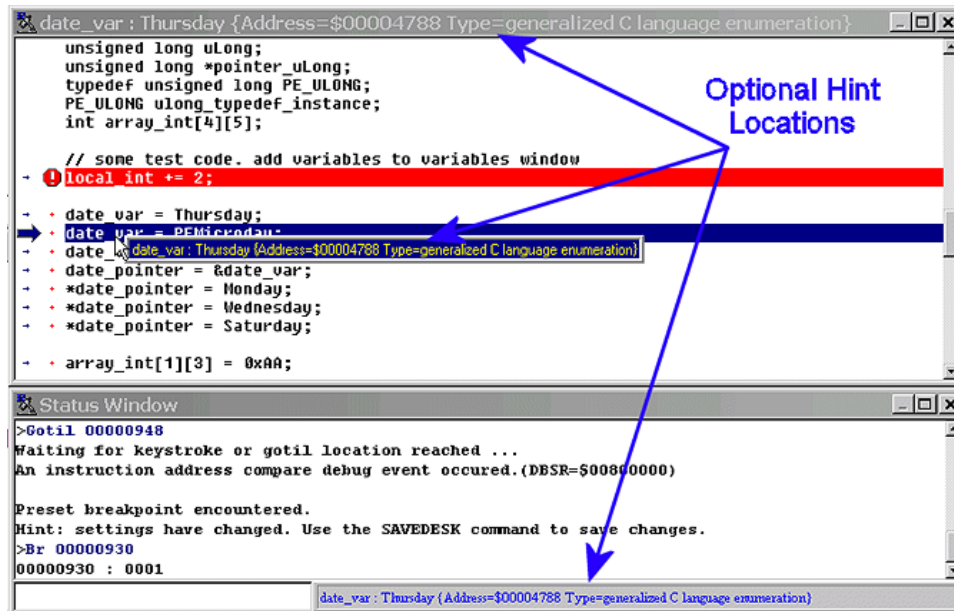
The disassembly window also supports double-clicking of the red and blue symbols, and there is an additional symbol that may appear: a small blue S enclosed in a box. This indicates that a source level instruction starts on this disassembly instruction. An image of this is shown here:

```

➔ • 0000093C 3D200000 LIS R9,0000
+ • 00000940 38000005 LI R0,0005
+ • 00000944 90094788 STW R0,4788(R9)
☐ • 00000948 3D200000 LIS R9,0000
  
```

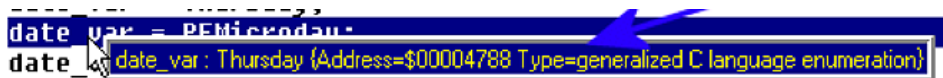
5.2.4 Using Code Window Popup Debug Evaluation Hints

When debugging source code, it is often advantageous to be able to view the contents of a variable that appears in the source code. The in-circuit debugger has a feature called “debug hints” which, when active, will display the value of a variable while the mouse cursor is held still over the variable name in the code window. The hint may be displayed in one of three locations, as shown below:



The three configurable locations are the code window title bar, the status window caption bar, or a popup that is displayed until the mouse is moved. The hint can be displayed in any combination of the three locations. Locations where the popup hints are displayed are set in the configuration menu of the debugger.

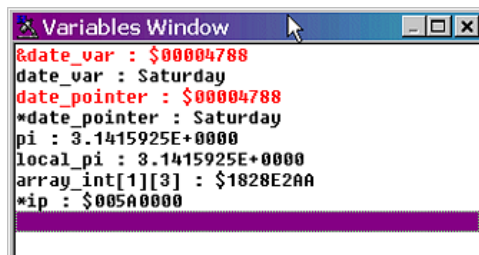
The information displayed in the hint box is similar to the information displayed in the variables window. A close-up image of this hint box is shown here:



The information shown is the variable name (`date_var`), value (`Thursday`), and type (`generalized C language enumeration`).

5.3 Variables Window

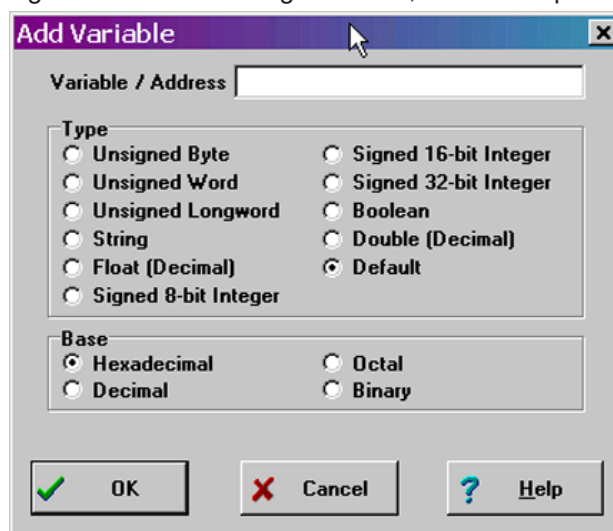
The variables window allows the user to constantly display the value of application variables. The following window shows a display of selected variables in the demonstration application:



Variables that are pointers are displayed in red. Normal variables are displayed in black. Real-time variables are displayed in blue. A real-time variable is a variable that is updated even while the processor is running.

5.3.1 Adding And Deleting Variables

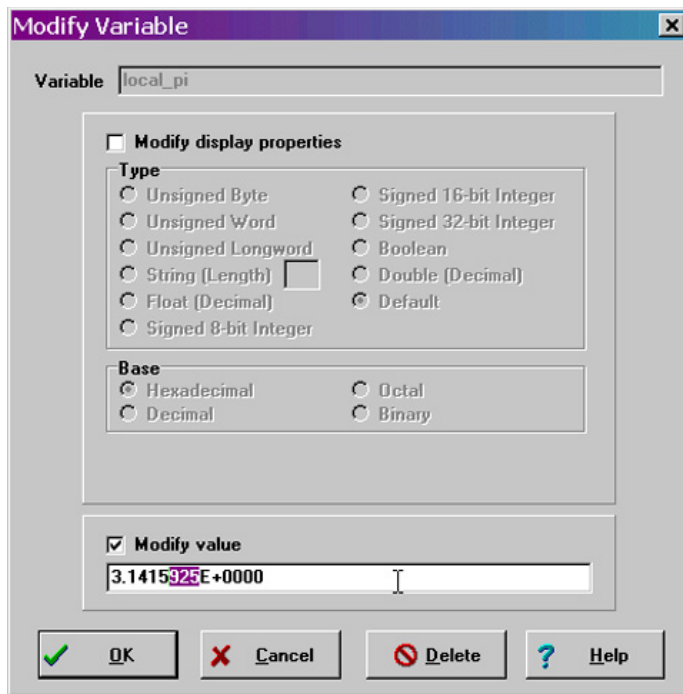
Variables may be added via the VAR command in the status window, or by right clicking the variables window and choosing “Add a variable.” Variables may be deleted by selecting them and choosing delete. When adding variables, the user is presented with the following dialog:



In the variable field, the user should input the address or name of the variable that they would like displayed in the variables window. The type of the variable should most often be set to “Default,” which means that the variable will be shown as it is defined in the compiled/loaded application. When adding a variable the user may also specify the numeric base in which the variable should be displayed.

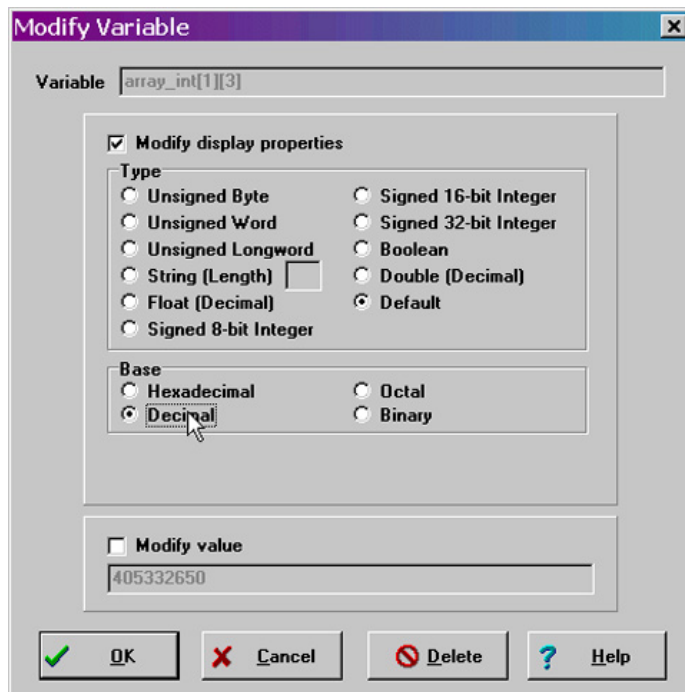
5.3.2 Modifying A Variable’s Value

To modify the current value of a variable, double-click the variable name in the variables window. If the debugger supports modification of this type of variable, the variable modification dialog will be displayed. Make sure to check the “Modify value” checkbox. At this point the value may be altered by the user. When the OK button is clicked, the variable value in the processor’s memory will be updated and the variable window will be refreshed to display this value. Note that some user-defined types, such as enumerated types, may not be editable in this fashion.



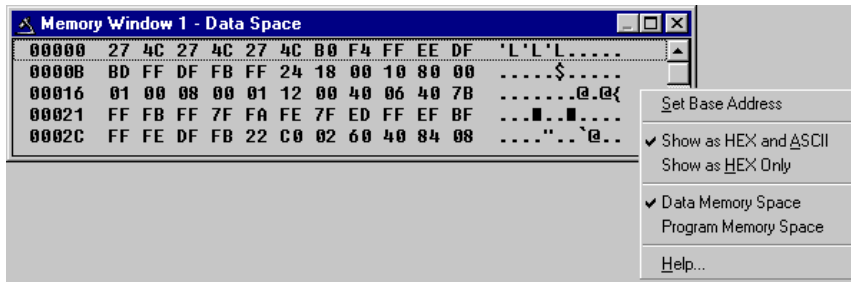
5.3.3 Modifying A Variable's Properties

To modify a variable's display properties, such as the type or numeric display base, double-click the variable in the variables window. Check "Modify display properties" in the dialog that is then displayed. At this point the type and base may be modified. When the OK button is clicked, the variable in the variables window will update its value according to the new settings.



5.4 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.



5.4.1 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.

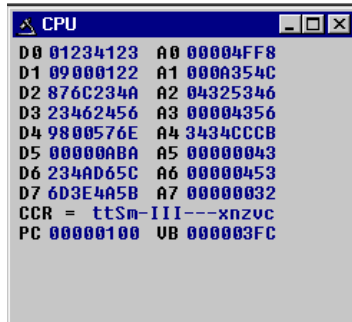
5.4.2 Keystrokes

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
ESC	Make the STATUS WINDOW the active window

5.5 CPU Window

The CPU Window displays the current state of the registers.



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.

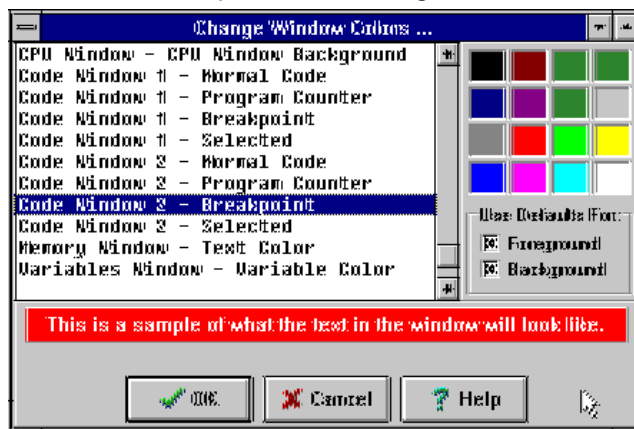
5.5.1 Keystrokes

The following keystroke is valid while the CPU window is the active window:

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.



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 Commands List

A

A(x)	Set Address Register A(x).
ASM	Assemble directly to ROM at address given or starting at last address used by this command.
ASCIIF3	
ASCIIF6	Toggle the F3(F6) memory window between showing hexadecimalbytes and ASCII characters. Nonprintable characters are shown as an ASCII period ('.').

B

BGND_TIME	Starts processor execution at the current Program Counter and logs time since the last BGND instruction each time a BGND instruction is encountered.
BF	Block fill. Same as FILL command.
BR	Sets or clears a breakpoint.

C

C	Set/clear C bit.
CAPTURE	Open a capture file named 'filename'.
CAPTUREOFF	Turn off capturing and close the current capture file.
CCR	Set Condition Code Register.
CLEARMAP	Remove all symbolic mapfile names.
CLEARSYMBOL	Remove all temporary symbols.
CODE	Show 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.
CONTROL	Display the value of the Control register at address add.
COUNT	Counts the number of times breakpoints in internal counter table are executed. Allows optional stop and start parameters to be set.
COUNTER	Add or subtract a location from the internal counter table. Using this command with no address shows the current counters.

D

D(x)	Set Data Register D(x).
DUMP_TRACE	Dump the current trace buffer to the debug window and to the capture file if enabled.

	DUMP	Dumps data memory to the Status Window.
E		
	EVAL	Evaluate input in decimal, hex, and binary. Can do simple calculations where op may be +, -, /, or *. Must have a space between numbers and [op].
	EXIT	Return to DOS.
F		
	FILL	See BF.
G		
	G or GO	Begin program execution. Same as RUN command.
	GOEXIT	Similar to GO command except that the target is left running without any breakpoints and the debugger software is terminated.
	GONEXT	Go from the current PC until the next instruction is reached. Used to execute past a subroutine call or past intervening interrupts.
	GOTIL	Go from the current PC until the PC = add.
	GOTILROM	Execute fast single steps without updating the screen, until the address is reached. This is the fastest way to breakpoint in ROM.
H		
	HELP	Bring up the help window.
	HGO	Begin Program Execution
	HLOAD	Load ELF/DWARF/S19/MAP Object And Debug Information
	HLOADMAP	Load DWARF/MAP Debug Info Only
	HSTEP	High-Level Language Source Step
	HSTEPFOR	High-Level Language Step Forever
I		
	I0, I1, I2	Set interrupt priority bits.
L		
	LOAD	Load a file (in S format). The load may be aborted by hitting a key. The default file-name extension is .S19.
	LOADALL	Do a LOAD and a LOADMAP command.
	LOADMAP	Loads debug map.
	LOADV	First performs the LOAD command and then automatically does a VERIFY command with the same file.

LOAD_BIN	Load a binary file of byte. The default filename extension is .BIN.
LOADV_BIN	First perform the LOAD_BIN command and then do a verify using the same file.
LPT1	
LPT2	
LPT3	Specifies which DOS compatible parallel port should be used for the debugger. See LPTx.
M	
M	Set/clear the M bit.
MACRO	Executes script file. Same as SCRIPT command.
MACROEND	Stop the logging commands into a macro file. Same as SCRIPTEND command.
MACROSTART	Start the recording of commands into a macro file. Same as SCRIPTSTART command.
MACS	Bring 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 cancel with the <ESC> key.
MD	Set the Data memory window to a specific address.
MD2	Set the Program memory window to a specific address. If the window is showing the VARIABLES, this command will return the window to the default hex display.
MM	Memory modify.
N	
N	Set/clear N bit.
NOBR	Clear all break points.
Q	
QUIET	Turn off (on) refresh of memory based windows.
QUIT	Return to DOS.
P	
PC	Set Program Counter.
R	
R	Display and edit registers both numerically and symbolically. See REGISTER DISPLAY section.
REM	Write a REMark to the debug window and to the capture file if enabled. Used for documenting capture files and for macro files.

RESET	Causes a hardware reset.
S	
SCRIPT	Executes script file. Same as MACRO command.
SERIAL	Set up parameters for serial port.
SERIALOFF	Turn off serial port use during GO.
SERIALON	Turn the F1 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.
SNAPSHOT	Take 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.
SOURCE	If a valid map file has been loaded, this command will toggle between showing actual source or disassembled memory while viewing code space represented in the source code.
SOURCEPATH	Either uses the specified filename or prompts the user for the DOS path to search for source code that is not present in the current directory.
SS	Do one step of source level code. Source must be showing in the code window.
ST	Single step. Cause the execution of n instructions. If no number is given, one instruction will be executed.
STATUS	Dump the current status of the processor to the status window. Often used for logging this information to a capture file.
STEP	Same as ST.
STEPFOR	Continuously single step until a breakpoint is reached or a key is hit.
STEPTIL	Repeatedly single step until the instruction pointer is equal to the given address.
SYMBOL	Add the given label to a temporary symbol table with the value given. Used without parameters, lists the temporary symbols in the debug window.
T	
T	Same as ST.
T1	Sets/clears the T bit in the Status Register.
TIME	Displays the amount of real time that elapses during execution of code. Address parameters may be preset or execution may be terminated by keystroke or existing breakpoint.
TRACE	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 in an internal array the address of up to the last 256 instructions executed.
U	
UPLOAD_SREC	Uploads the program memory locations to the screen in the form of S records.

V

V	Set/clear V bit.
VAR	Displays a specified address and its contents in the Variables Window for viewing during code execution.
VBR	Set vector base register.
VERIFY	Compare 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.

W

WATCHDOG	Toggle the state of the SWE bit in the SYPCR. Remember that this register may only be written once following a reset of the hardware.
WHEREIS	Display the value for the given symbol either temporary or MAP file based. See SYMBOL.

X

X	Set/clear X bit.
---	------------------

Z

Z	Set/clear Z bit.
---	------------------

7.1 A(x) Command

The A(x) command sets the value of the 32-bit Address Register A(x), where (x) is a value from 0 to 7. A7 is the Stack Pointer.

Syntax:

A(x) [n]

Where:

(x) Value from 0 to 7, corresponding to which A register the user intends to write.

[n] Value to be written to register.

Example:

A3 \$CF03D4AA Writes value of \$CF03D4AA to Address Register A3.

7.2 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:

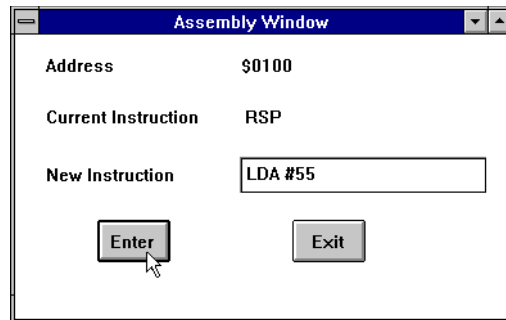
<address> Address where machine code is to be generated. If you do not specify an <address> value, the system checks the address used by the previous ASM command, then uses the next address for this ASM command.

Examples:

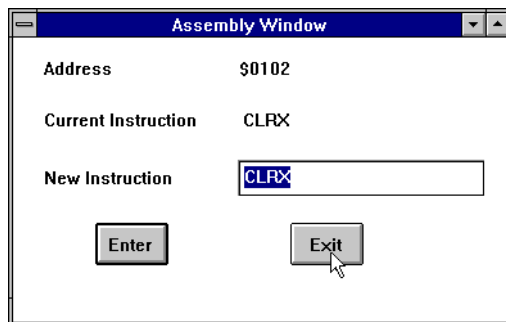
With an address argument:

>ASM 100

The following window appears:



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 clicked the following window will appear:



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.

7.3 ASCIIIF3 and ASCIIIF6

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

ASCIIIF3 toggles memory window 1.

ASCIIIF6 toggles memory window 2.

Syntax:

ASCIIIF3

Example:

>ASCIIIF3 Toggles memory window 1 between displaying [data only] // [data and ASCII characters].

7.4 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).



7.5 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 long 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 C0 CF AA                  Store hex value AA in bytes at addresses C0-CF.
```

```
>FILL.B C0 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
```

7.6 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:

add	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.
```

7.7 C Command

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 X, N, Z, V, C. X is extend, N is negative, Z is zero, V is overflow, and C is carry. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

```
C 0|1
```

Examples:

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

7.8 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".
```

7.9 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.

7.10 CCR - 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 X, N, Z, V, C. X is extend, N is negative, Z is zero, V is overflow, and C is carry. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

CCR <n>

Where:

<n> The new hexadecimal value for the CCR.

Example:

>CCR \$C4 Assign the value C4 to the CCR.

7.11 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.

7.12 CLEARVAR

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

Syntax:

CLEARVAR

Example:

CLEARVAR Removes all the variables from the variables window.

7.13 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.

7.14 Control Command

The CONTROL Command displays the value of the Control register at address add. The user can then enter a new value for the register or a simple carriage return to keep the same value. The addresses used are the same as for the MOVEC instruction. This is used to setup the MBAR, RAMBAR and other Control registers.

Syntax:

```
CONTROL <add>
```

Where:

```
<add>           Address of Control Register to display.
```

Example:

```
CONTROL $0500    Displays Control Register at address $0500.
```

7.15 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.
```

7.16 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.
```

7.17 D(x) Command

The D(x) command sets the value of the 32-bit Data Register D(x), where (x) is a value from 0 to 7.

Syntax:

D(x) [n]

Where:

(x) Value from 0 to 7, corresponding to which register the user intends to write.
 [n] Value to be written to register.

Example:

D3 \$CF03D4AA Writes value of \$CF03D4AA to Data Register D3.

7.18 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.

7.19 DUMP_TRACE

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

Syntax:

Dump_Trace

Example:

>Dump_Trace

7.20 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"
```

7.21 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.
```

7.22 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.
```

7.23 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.

7.24 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.

7.25 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.

7.26 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.

7.27 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.

7.28 I0, I1, I2 Commands

The I0, I1, and I2 commands allow the user to set the interrupt priority mask in the status register. These bits may only be set in supervisor privilege level.

Syntax:

I(x) [n]

Where:

(x) 0, 1, or 2 corresponding to which I register the user intends to set.
[n] 0 or 1, corresponding to the value to which the user wishes to set the register.

Example:

I1 1 Sets the I1 bit to 1.

7.29 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.

7.30 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.

7.31 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.

7.32 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.

7.33 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 100 Loads a binary myfile of bytes starting at hex address 100

7.34 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.

7.35 LPTx

Specifies which PC compatible parallel port should be used for the debugger. The port must be fully PC compatible and a full 25-pin cable must be used.

Syntax:

LPTx

Where:

x	1, 2 or 3
---	-----------

Example:

LPT 2 Specifies that the debugger should use parallel port 2

7.36 M Command

The M command allows the user to set/clear the M bit (master/interrupt state).

Syntax:

M 0|1

Example:

M 1 Sets the value of the M bit to 1.

7.37 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.

7.38 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.

7.39 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
```

7.40 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
```

7.41 MD - Memory Display

The MD command displays the contents of 32 emulation memory locations in the first 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:

```
MD <address>
```

Where:

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

Example:

```
>MD 1000 Display the contents of 32 bytes of memory beginning at address 1000.
```

7.42 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.
```

7.43 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> ...]

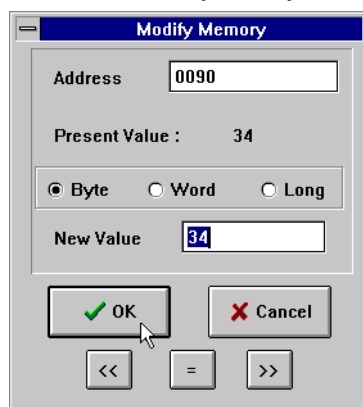
Where:

<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.



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.

7.44 N Command

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 X, N, Z, V, C. X is extend, N is negative, Z is zero, V is overflow, and C is carry. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

N 0|1

Examples:

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

7.45 NOBR

Clears all break points.

Syntax:

NOBR

Example:

>NOBR Clears all break points.

7.46 PC - 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.

7.47 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 Toggles the current debugger state between quiet and not quiet.
- >QUIET OFF Disables Quiet mode (all windows refresh).
- >QUIET ON Enables Quiet mode (windows are blank and do not refresh).

7.48 QUIT

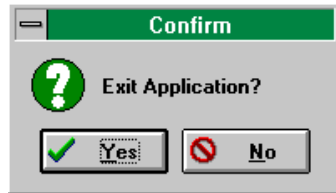
Exit the program.

Syntax:

QUIT

Example:

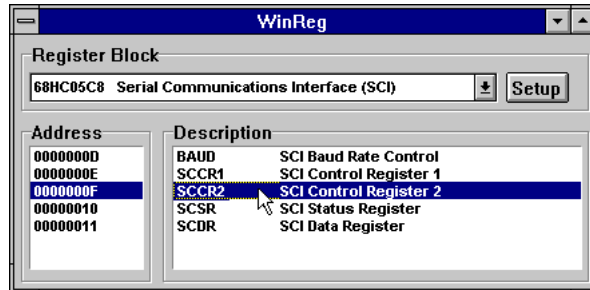
>QUIT Exit the application



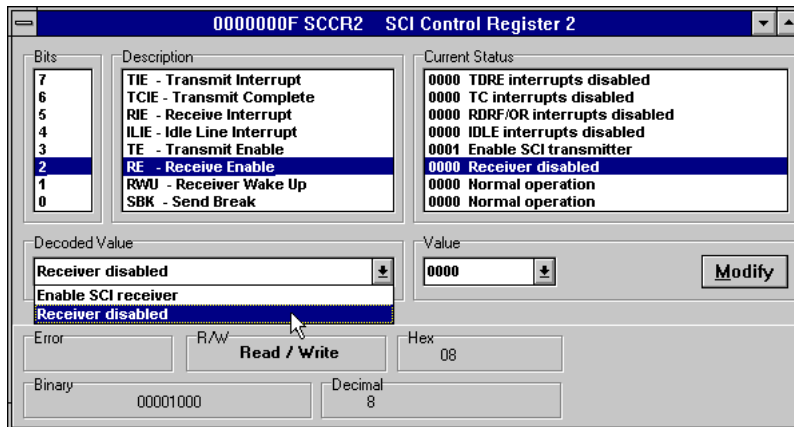
7.49 R - Use Register Files

The R command opens a processor's register files (sold separately by PE micro) 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.



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.



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.

7.50 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.

7.51 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:

>RESETReset the MCU.

7.52 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

7.53 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
```

7.54 SERIALOFF

Turns off serial port use during GO.

Syntax:

SERIALOFF

Example:

>SERIALOFF Turns off serial port use during GO command

7.55 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 snapshot 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.

7.56 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.

7.57 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

7.58 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.
```

7.59 STATUS or REG - Show Registers

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

Syntax:

```
STATUS
```

Example:

```
>STATUS             Displays the contents of the CPU registers.
```

7.60 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.
```

7.61 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.

7.62 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'.

7.63 T1 Command

The T1 command sets/clears the trace enable. When set, the processor will perform a trace exception after every instruction.

Syntax:

T1 0|1

Example:

T1 0 Assigns value of 0 to T bit in Status Register.

7.64 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.

7.65 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.

7.66 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.

7.67 V Command

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 X, N, Z, V, C. X is extend, N is negative, Z is zero, V is overflow, and C is carry. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

V 0|1

Examples:

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

7.68 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 hexadecimal, 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.

7.69 VBR Command

The VBR command allows the user to set the vector base register. The VBR contains the base address of the exception vector table in memory. Same as VB command.

Syntax:

VBR [n]

Where:

[n] Value to assign to the vector base register, between \$00000000-\$FFFFFFFF.

Example:

VBR \$006D219B Assigns \$006D219B to VBR.

7.70 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"
```

7.71 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
```

7.72 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.
```

7.73 X Command

The X command sets or clears (that is, assigns 0 or 1 to) the X 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 X, N, Z, V, C. X is extend, N is negative, Z is zero, V is overflow, and C is carry. A letter in these designators

means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

X 0|1

Examples:

>X 0 Clear the X bit of the CCR.

>X 1 Set the X bit of the CCR.

7.74 Z Command

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 X, N, Z, V, C. X is extend, N is negative, Z is zero, V is overflow, and C is carry. A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

Z 0|1

Examples:

>Z 0 Clear the Z bit of the CCR.

>Z 1 Set the Z bit of the CCR.

8 Source Level Debugging

Once a valid map file (generated by CASM12Z) is loaded into the ICD via the LOADMAP or LOADALL command, source level debugging is enabled. When the PC 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.

9 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 device.

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.

10 Instruction Set

ADD	ADDA	ADDI	ADDQ	ADDX	AND
ANDI	ASL	ASR	BCC	BCS	BEQ
BGE	BGT	BHI	BLE	BLS	BLT
BMI	BNE	BPL	BVC	BVS	BCHG
BCLR	BRA	BSET	BSR	BTST	CLR
CMP	CMPA	CMPI	EOR	EORI	EXT
EXTB	HALT	JMP	JSR	LEA	LINK
LPSTOP	LSL	LSR	MOVE	MOVEA	MOVEC
MOVEM	MOVEQ	MULS	MULU	NEG	NEGX
NOP	NOT	OR	ORI	PEA	PULSE
RTE	RTS	SCC	SCS	SEQ	SF
SGE	SGT	SHI	SLE	SLS	SLT
SMI	SNE	SPL	ST	SVC	SVS
STOP	SUB	SUBA	SUBI	SUBQ	SUBX
SWAP	TRAP	TRAPF	TST	UNLK	WDDATA
WDEBUG					

11 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 to not 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 debug probe 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 debug probe without removing power connections.**
3. Remove power from debug probe.
4. Do whatever...
5. Connect power to debug probe.
6. Start the ICD software using the RUNNING option.
7. Connect debug probe to your target.

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

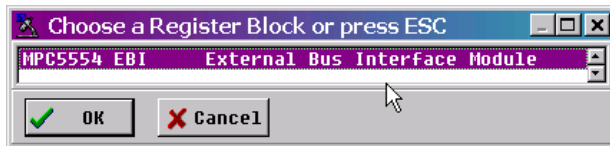
- a. Cut the Vdd lead in the debug probe and use a separate power supply or jumper the debug probe power back to your target. If you use a separate supply, you should maintain a common ground.
- b. Attach a second insulation displacement connector to the debug probe and jumper from it to the target power supply.

12 Using The Register Interpreter

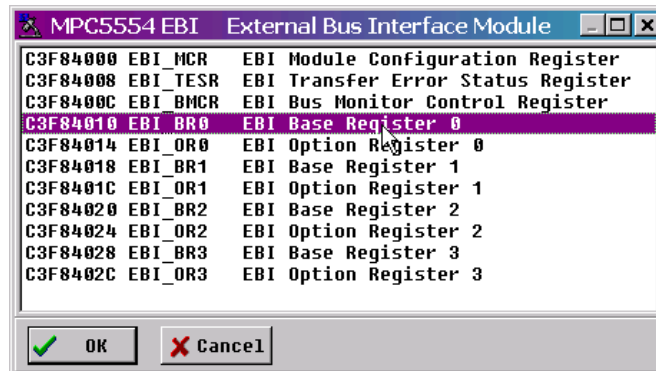
The register interpreter allows descriptive display/modification of bit fields within the processor's peripheral registers. This capability allows the user to quickly check the current state of a peripheral and easily check their configuration of the device. When displayed from the debugger, the register interpreter reads the current value of the peripheral register, decodes it, and displays it for the user. When displayed from WinIDE, all the fields are initialized to 0, and when the user hits the Enter key the value is written into the editor at the current cursor location (as opposed to being written back into memory as the debugger does).

12.1 Register Interpreter Display

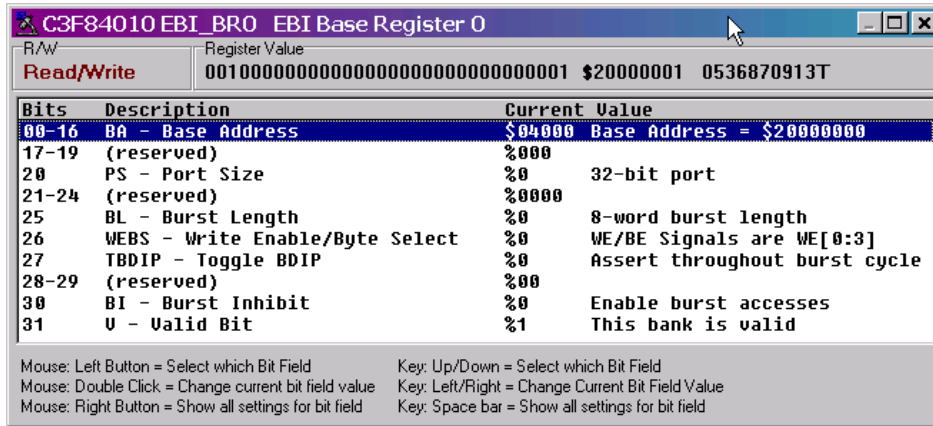
In the debugger, use either the "R" command or click the view/edit register button on the main button bar. To display in the WinIDE, use the "SHIFT-F1" keystroke combination or the register files button on the main buttonbar. In either application, a window will appear allowing the user to select a specific peripheral block to choose (this image shows only one available block):



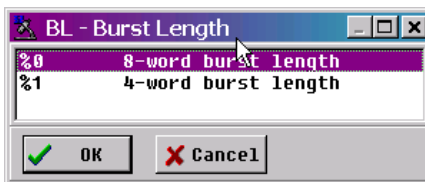
Double clicking the module of choice will bring up the register selection window:



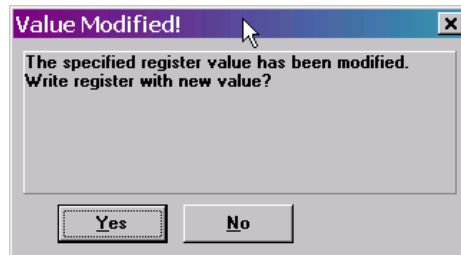
Double clicking a specific register will bring up the edit/display window for that register:



The keystrokes and mouse actions are listed in the windows which allow the user to modify the values of each of the fields. By right clicking on a specific field, the user is shown all options for that particular field:

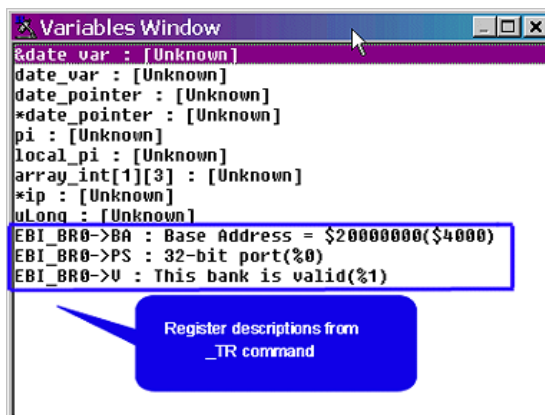


When in the debugger, when the user quits the register view/edit window by hitting the ESC key, if they have modified the register's value, they will be given the opportunity to write the new value into the register as shown in the following window:



12.2 Adding Register Field Descriptions To VAR Window

Bit fields defined within a register description may be added to the variables window via the “_TR” command in the debugger. After selecting a field, this field is added to the variables window and will be continually updated with all the other information in the variables window. A variables window with some fields from one of the chip selects is shown here:



13 CPU Values & Names

CPU Values:

Any on screen location with an alphabetic name (A0, PC, etc.) may be changed by entering the name of the location followed by a value and <Enter> key will result as :

>A0 44

This will change the value of A0 register to hex value 44.

>D7 FF

This will change the value of D7 register to hex value FF

>PC 100

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

>S 1

This will set the supervisor mode.

CPU Names:

REGISTERS	FLAGS in CCR
D0 ... D7	T1 and T0 - for TT
A0 ... A7	I2, I1 and I0 - for III
PC	X
VB	N
SFC	Z
DFC	V
CCR	C
	S

Note: The CCR is displayed with alphabetic characters. An upper case character is used when the bit is a 1 and a lower case character is used when the bit is a 0. A dash denotes a bit that never changes, in this case they are read as 0.

CCR = 1010010100011001 = TtS--□i□---XNzvC