

1 Connection To Target

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

You may use this dialog box to connect to your target with PEmicro's Multilink debug probe or Cyclone programmer. It allows you to specify a port for connection. You may also select the CPU type, communications speed, and internal bus frequency. If you wish, you may disable the dialog box so that it will appear only on error.

2 Command Line Parameters

To setup ICD32Z to run with certain command line parameters, highlight the ICD32Z icon and select PROPERTIES from the Program Manager File Menu.

Syntax:

ICD32Z [option] ... [option]

[option] Optional parameters are as follows:

bim_module	Specifies that software is talking to a part with a Burst Integration Module. This module occurs in members of the CPU32X family, such as the 68373 Sabretooth. Default = false.
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:

ICD32Z lpt2 io_delay_cnt 2 bw

Chooses lpt2, Causes the background debug mode clock to be extended by 2 Cycles. Sets display to black and white.

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.

2.1 RUNNING Option

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.

3 Nomenclature

Note the following:

n any number from 0 to 0FFFFFFFF (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 User Interface

This section describes the various windows in the ICD32Z user interface.

4.1 CPU Window

The CPU Window displays the current state of the registers.

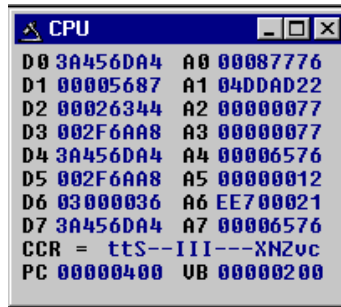


Figure 4-1: CPU Window

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

4.2 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.

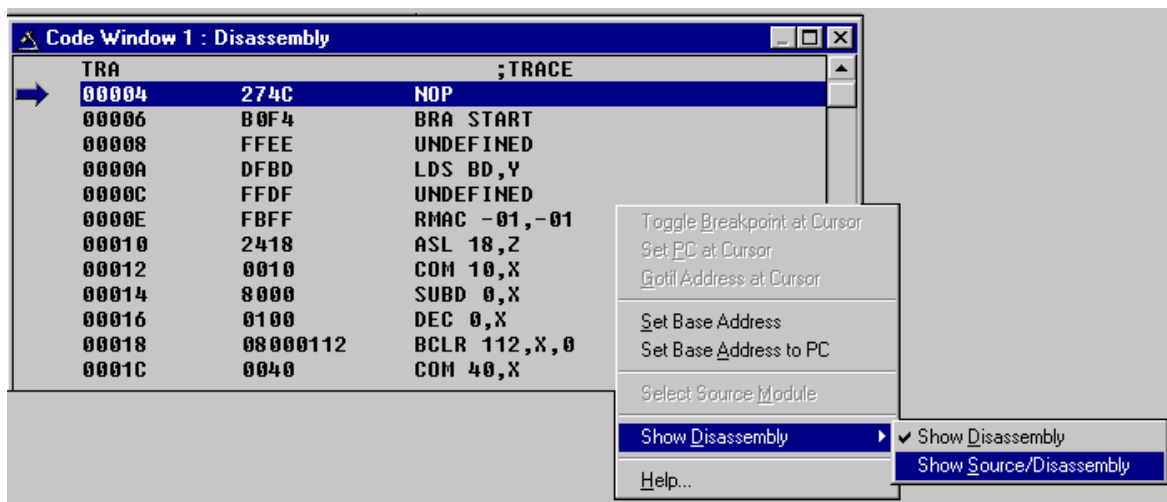


Figure 4-2: Code Window

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

4.3 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.

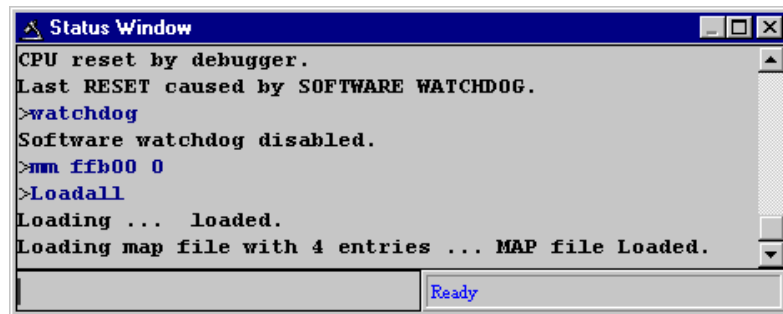


Figure 4-3: Status Window

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.

4.4 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.

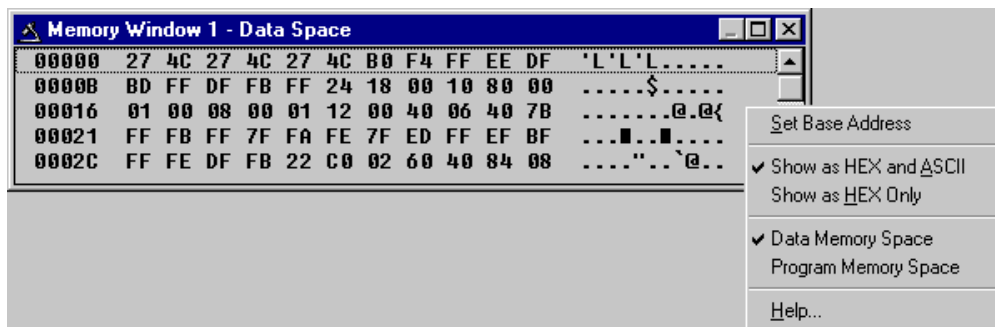


Figure 4-4: Memory Window

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

4.5 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).

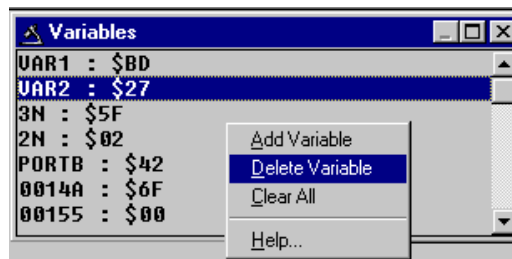


Figure 4-5: Variables Window

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 are 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

4.6 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.

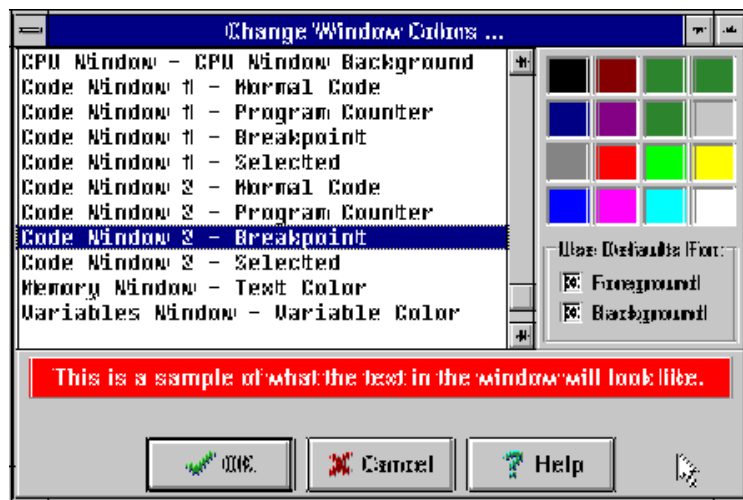


Figure 4-6: Colors Window

5 ICD32Z Commands - Full Descriptions

This section provides full descriptions of the ICD32Z commands, including formatting and examples.

5.1 332_CHIP Command

The 332_CHIP command returns the chip to normal operation after the use of the 340_CHIP or 360_CHIP commands.

Example:

```
>332_CHIP           Returns the chip to normal operation.
```


5.2 340_CHIP Command

A new command, 340_CHIP, has been added to support the 68HC340 chip in the ICD32Z debugger. This command allows for the use of register files if the SIM module is enabled. In addition, it stops the debugger from polling the reset status register unless the SIM module is enabled. The command, 332_CHIP returns the debugger to normal operation.

Note: To enable the SIM module, the debugger needs to talk to the "CPU space". Use the command: DEBUGGER_DFC 7 to accomplish this. Then modify the MBAR register using a MM.L command. Point the debugger back to the SUPERVISOR SPACE with the command DEBUGGER_DFC 5. At this point, you should be able to change the SIM registers. Placing this sequence in the STARTUP.ICD file will automate the process for you.

Example:

At the debug window type:

```
>340_Chip
```

To turn on the SIM space type the following commands

```
>DEBUGGER_DFC 7  
>MM.L address_of_the_MBAR_register value  
>DEBUGGER_DFC 5
```

5.3 360_CHIP Command

A new command, 360_CHIP, has been added to support the 68HC360 chip in the ICD32Z debugger. This command allows for the use of register files if the SIM module is enabled. In addition, it stops the debugger from polling the reset status register unless the SIM module is enabled. The command 332_CHIP returns the debugger to normal operation.

Note:

To enable the SIM module, the debugger needs to talk to the "CPU space". Use the command: DEBUGGER_DFC 7 to accomplish this. Then modify the MBAR register using a MM.L command. Point the debugger back to the SUPERVISOR SPACE with the command DEBUGGER_DFC 5. At this point, you should be able to change the SIM registers. Placing this sequence in the STARTUP.ICD file will automate the process for you.

Example:

At the debug window type:

```
>360_Chip
```

To turn on the SIM space type the following commands:

```
>DEBUGGER_DFC 7  
>MM.L address_of_the_MBAR_register value  
>DEBUGGER_DFC 5
```

5.4 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.

Syntax:

A(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:

A3 \$CF03D4AA Writes value of \$CF03D4AA to Address Register D3.

5.5 ASCIIF3 and ASCIIF6 Commands

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].

5.6 ASM Command - 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:

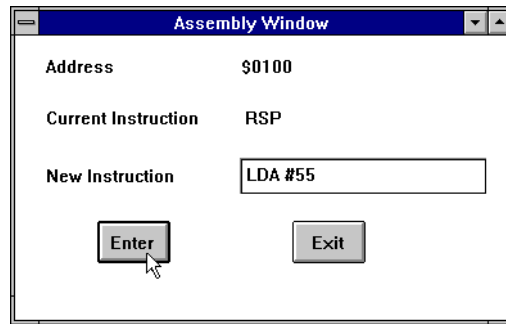
<address> Address where machine code is to be generated. If you do not specify an <address> value, the system checks the address used by the previous ASM command, then uses the next address for this ASM command.

Examples:

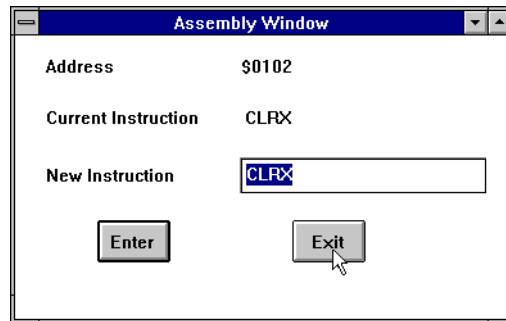
With an address argument:

>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.

5.7 BELL Command - 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.

5.8 BF Command - Block Fill

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] <starange> <endrange> <n>

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

Where:

<starange> 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 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

5.9 BGND_TIME Command

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]

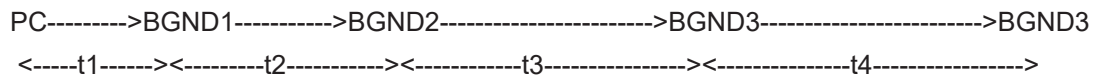
Where:

 n number of points logged

Example:

>BGND_TIME 4

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



5.10 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:

- 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.

5.11 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.

5.12 CAPTURE Command

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".

5.13 CAPTUREOFF Command

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.

5.14 CCR (Condition Code Register) Command

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.

5.15 CLEARMAP Command - 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.

5.16 CLEARSYMBOL Command - 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.

5.17 CLEARVAR Command

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

Syntax:

CLEARVAR

Example:

CLEARVAR Removes all the variables from the variables window.

5.18 CODE Command

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.

5.19 COLORS Command - 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.

5.20 COUNT Command

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.

5.21 COUNTER Command

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.

5.22 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.

5.23 D_UPLOAD_SREC Command

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.

5.24 DASM Command - 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
```

5.25 DEBUGGER_DFC Command

Sets the DFC (Destination Function Code) while the user is in the debugger. Default DFC while in the debugger is 5. For setting DFC when the processor is executing, see DFC command.

Syntax:

```
DEBUGGER_DFC [n]
```

Where:

[n] Value for DFC while in debugger.

Example:

```
DEBUGGER_DFC 3 Sets DFC while in debugger to 3.
```

5.26 DEBUGGER_SFC Command

Sets the SFC (Source Function Code) while the user is in the debugger. Default SFC while in the debugger is 5. For setting SFC when the processor is executing, see SFC command.

Syntax:

```
DEBUGGER_SFC [n]
```

Where:

[n] Value for SFC while in debugger.

Example:

```
DEBUGGER_SFC 3    Sets SFC while in debugger to 3.
```

5.27 DFC Command

Sets the DFC (Destination Function Code) to the 3-bit value n. This DFC is valid only while the processor is executing. The DFC is forced to 5 while in the background debug mode using this debug software. Using the DFC command without a parameter n causes the current DFC to be displayed.

Syntax:

```
DFC [n]
```

Where:

[n] Value for DFC when processor is executing.

Examples:

DFC 3	Sets value of DFC to 3.
DFC	Displays current value of DFC.

5.28 DUMP Command - 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.

5.29 DUMP_TRACE Command

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

Syntax:

```
Dump_Trace
```

Example:

```
>Dump_Trace
```

5.30 EVAL Command - 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"
```

5.31 EXIT or QUIT Command - 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.
```

5.32 G or GO or RUN Command - 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.
```

5.33 GOEXIT Command

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.
```

5.34 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.

5.35 GOTIL Command - 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.

5.36 GOTILROM Command

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.

5.37 HELP Command - 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.

5.38 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.

5.39 INFO Command - 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.ASM Line number:6
 Address: \$0100
 Disassembly: START 5F CLRX

5.40 LF or LOGFILE Command - 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.

5.41 LISTOFF Command - 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.

5.42 LISTON Command - 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.

5.43 LOAD Command - 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.

5.44 LOAD_BIN Command

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

5.45 LOADALL Command

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.

5.46 LOADV_BIN Command

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.

5.47 LOADDESK Command - 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.

5.48 LOADMAP Command - 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.

5.49 LOADV Command

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.

5.50 LPTx Command

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

5.51 MACRO or SCRIPT Command - 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.

5.52 MACROEND Command - 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.

5.53 MACROSTART Command - 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

5.54 MACS Command

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

5.55 MD or SHOW Command - 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.

5.56 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.

5.57 MDF3 / MDF6 or SHOWF3 / SHOWF6 Commands

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 Displays code beginning at address \$1000 in memory window F3.

5.58 MM or MEM Command - 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 5-7: Memory Modify 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.

5.59 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.

5.60 NOBR Command

Clears all break points.

Syntax:

NOBR

Example:

- >NOBR Clears all break points.

5.61 PC Command - 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.

5.62 PDUMP Command - 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.

5.63 QUIET Command

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

5.64 QUIT Command

Exit the program.

Syntax:

QUIT

Example:

>QUIT Exit the application

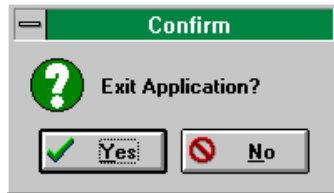


Figure 5-8: Exit Dialog

5.65 R Command - 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.

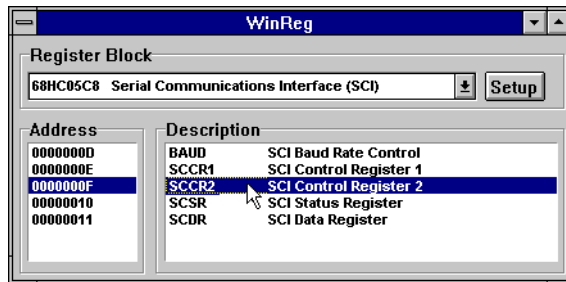


Figure 5-9: WinReg: Register Selection

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.

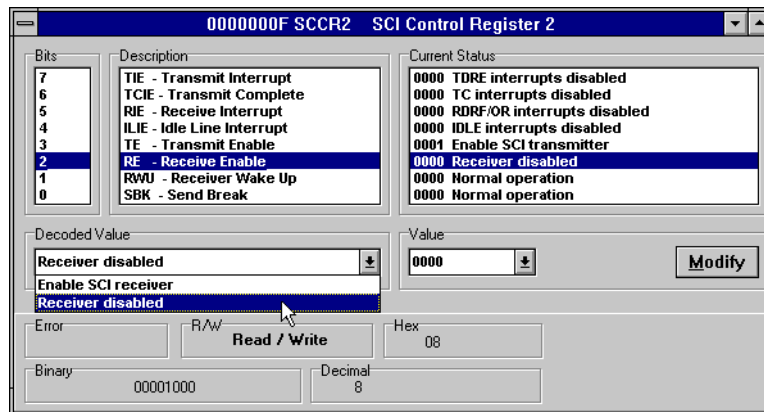


Figure 5-10: WinReg: Register Display

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.

5.66 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.

5.67 REM Command - 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.

5.68 RESET Command - 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.

5.69 S Command

The S command sets the supervisor control bit to either user or supervisor privilege level. Consult the processor reference manual for more details.

Syntax:

S [n]

Where:

[n]0|1

Example:

S 1 Sets the supervisor control bit to 1.

5.70 SAVEDESK Command - 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, et.

Syntax:

SAVEDESK

Example:

>SAVEDESK Save window settings for the application.

5.71 SERIAL Command

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

5.72 SERIALON Command

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
```

5.73 SERIALOFF Command

Turns off serial port use during GO.

Syntax:

SERIALOFF

Example:

>SERIALOFF Turns off serial port use during GO command

5.74 SFC Command

Sets the SFC (Source Function Code) to the 3-bit value n. This SFC is valid only while the processor is executing. The SFC is forced to 5 while in the background debug mode using this debug software. Using the SFC command without a parameter n causes the current SFC to be displayed.

Syntax:

SFC [n]

Where:

[n] Value for SFC when processor is executing.

Examples:

SFC 3	Sets value of SFC to 3.
SFC	Displays current value of SFC.

5.75 SHOWCODE Command - 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.

5.76 SHOWMAP or MAP Command - 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.

5.77 SHOWPC Command - 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.

5.78 SHOWTRACE Command

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.

5.79 SNAPSHOT Command

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.

5.80 SOURCE Command

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.

5.81 SOURCEPATH Command

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

5.82 SP (Stack Pointer) Command

The SP command sets the Stack Pointer (A7) 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.

5.83 SS Command

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.

5.84 ST or STEP or T Command - 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.

5.85 STEPFOR Command

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.

5.86 STEPTIL Command - 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.

5.87 SYMBOL Command - 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'.

5.88 T0 and T1 Commands

The T0 and T1 commands allow the user to put the processor in one of two trace modes. Consult the CPU Manual for specific details regarding each tracing mode and its corresponding setting. These bits may only be set in supervisor privilege level.

Syntax:

T(x) [n]

Where:

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

Example:

T1 1 Sets the T1 bit to 1.

5.89 TIME Command

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.

5.90 TRACE Command

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.

5.91 **UPLOAD_SREC Command - 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.

5.92 **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.

5.93 **VAR Command - 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.

5.94 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.

5.95 VERIFY Command

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"

5.96 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.

5.97 WATCHDOG Command

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

5.98 WHEREIS Command - 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.

5.99 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.

5.100 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.

5.101 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.

6 Function Keys

The following are valid function keys for ICD32Z:

F1	Help
F2	Load S19 File
F3	Reload Last S19
F7	Show Trace Buffer
F9	Repeat Last Command

7 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--lll---XNzvC

8 Source Level Debugging

Once a valid map file (generated by CASM32Z) 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 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 information, see the Code Window section.

9 Addressing Modes

Dn	Data Register Direct.
An	Address Register Direct.
(An)	Address Register Indirect.
(An)+	Address Register Indirect with Post-increment.
-(An)	Address Register Indirect with Pre-decrement.
(d16,An)	Address Register Indirect with Displacement.
(d8,An,Xn.Size*Scale)	Address Register Indirect with Index (8-bit Displacement). d8 = -128 to 128, Xn = Dn or An, .Size (Optional) = .W or .L, default = .L, *Scale (Optional) = 1, 2, 4 or 8
(bd,An,Xn.Size*Scale)	Address Register Indirect with Index (Base Displacement). bd (Optional), An (Optional), Xn (Optional) = Dn or An,

.Size (Optional) = .W or .L, default = .L,

*Scale (Optional) = 1, 2, 4 or 8

(xxxx).W	Absolute Short Address, xxxx.W is also allowed.
(xxxx).L	Absolute Long Address, xxxx.L is also allowed.
(xxxx)	If the.W or.L suffix is left off, the size is set to the.W option as long as sizing is left off. If sizing is on and the number xxxx is known in the first pass of assembly and it fits into a 16 bit word, the size is set to .W, otherwise it is interpreted as a.L option. The form xxxx is also allowed.
#xxx	Immediate Data.
(d16,PC)	Address Register Indirect with Displacement. d16 (relative to PC+2)
(d8,PC,Xn.Size*Scale)	Address Register Indirect with Index (8-bit Displacement). d8 = -128 to 128 (relative to PC+2) Xn = Dn or An .Size (Optional) = .W or .L, default = .W *Scale (Optional) = 1, 2, 4 or 8
(bd,xPC,Xn.Size*Scale)	Address Register Indirect with Index (Base Displacement). bd (Optional) (relative to PC+2 unless ZPC is specified) xPC = PC or ZPC (ZPC for program space and program counter not used) Xn (Optional) = Dn or An .Size (Optional) = .W or .L, default = .W *Scale (Optional) = 1, 2, 4 or 8
address	Address for Relative Address. Relative address calculated as given address minus Extension_Word_Address+2. Used in branching instructions.
list	list of registers (r's) for movem instruction
<u>Options:</u>	
#const	user specified bit pattern not used in conjunction with other options
An	address register n (0<=n<=7)
Dn	data register n (0<=n<=7)
An-Am	sequence of address registers
Dn-Dm	sequence of data registers
/	separates multiple options for example A2-A5/D7-D4/A0/D1

Note: Normally, the options are stored in a word in the order A7...A0, D7...D0. However, if the predecrement addressing mode, -(An) is used the given bit pattern is reversed in the word to give A7...A0, D7...D0. This is true even when the #const option is used.

Examples:

<u>Source Code</u>	<u>Object Code</u>
movem #1234,(a2)	4892 1234
movem d6-d7,(a2)	4892 00C0
movem a1-d2-a5-d7,(a2)	4892 2284
movem a1-a2-d6-d7,(a2)	4892 06C0
movem #1234,-(a2)	48A2 2C48
movem d6□d7,-(a2)	48A2 0300
movem a1-d2-a5□d7,-(a2)	48A2 2144
movem a1-a2-d6□d7,-(a2)	48A2 0360

Note: Even though address parameters may be optional, the structure is required. Thus, address register indirect with index and a base displacement with the address register omitted, a base displacement of \$3456, and index of D4 and default size and scale is written as (\$3456,,D4).

10 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;

```

Figure 10-1: Code Window

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 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)
S + 00000948 3D200000 LIS R9,0000

```

11 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:

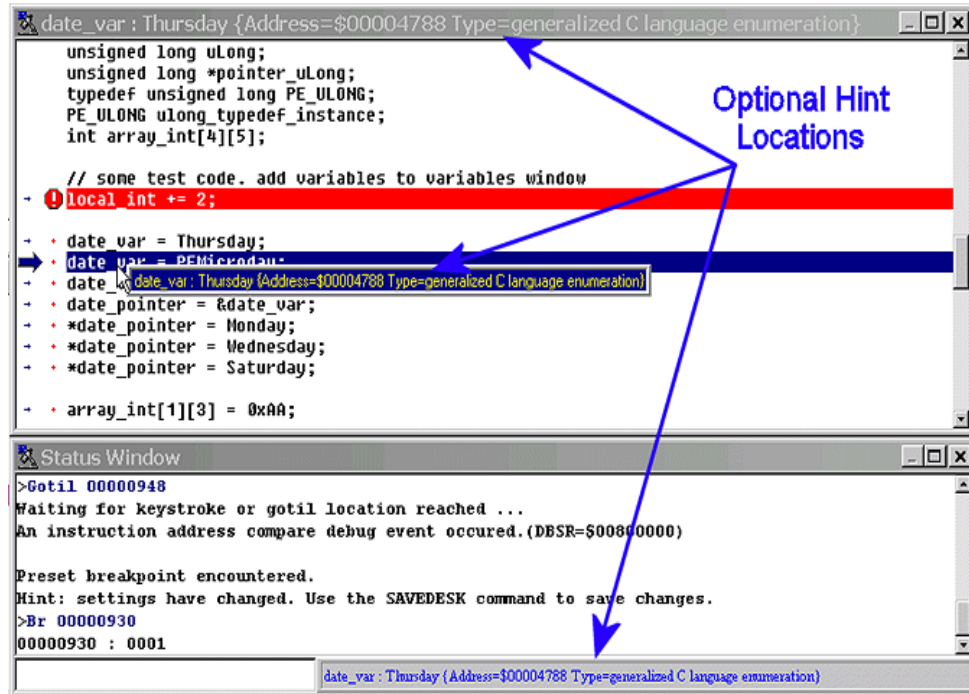


Figure 11-1: Pop-Up Hint

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:

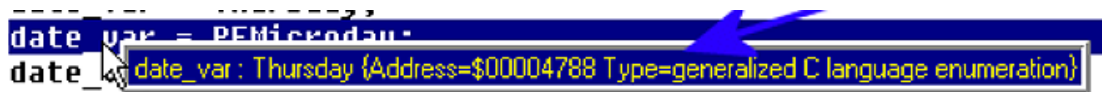


Figure 11-2: Pop-Up Hint

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

12 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 in an internal array the address of every instruction executed (actually, the last 256 instructions).

After execution, you may hit F7 to view the trace buffer.

13 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 CPU32.

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.

14 Instruction Set

ABCD	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	BGND	BKPT	BRA	BSET
BSR	BTST	CHK	CHK2	CLR
CMP	CMPA	CMPI	CMPM	CMP2
DBCC	DBCS	DBEQ	DBF	DBGE
DBGT	DBHI	DBLE	DBLS	DBLT
DBMI	DBNE	DBPL	DBT	DBVC
DBVS	DIVS	DIVSL	DIVU	DIVUL
EOR	EORI	EXG	EXT	EXTB
JMP	JSR	LEA	LINK	LIST
LPSTOP	LSL	LSR	MOVE	MOVEA
MOVEC	MOVEM	MOVEP	MOVEQ	MOVES
MULS	MULU	NBCD	NEG	NEGX
NOP	NOT	OR	ORG	ORI
PEA	RESET	ROL	ROR	ROXL
ROXR	RTD	RTE	RTR	RTS

SBCD	SCC	SCS	SEQ	SF
SGE	SGT	SHI	SLE	SLS
SLT	SMI	SNE	SPL	ST
SVC	SVS	STOP	SUB	SUBA
SUBI	SUBQ	SUBX	SWAP	TAS
TBLS	TBLSN	TBLU	TBLUN	TRAP
TRAPCC	TRAPCS	TRAPEQ	TRAPF	TRAPGE
TRAPGT	TRAPHI	TRAPLE	TRAPLS	TRAPLT
TRAPMI	TRAPNE	TRAPPL	TRAPT	TRAPVC
TRAPVS	TRAPV	TST	UNLK	

15 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 ICD cable 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 in either of the following 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.

16 Memory Access

1. When you modify bytes, words, or longs they are read/written using the corresponding background debug mode read/write.
2. Memory window displays are read using word reads.
3. The VAR window is read using the appropriate reads.
4. Code window data is read using word reads.

17 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.

18 Register Display

Are you tired of looking through manual pages for register descriptions? PEmicro has the ultimate solution for this problem.

The R command lets you look at and modify the registers and the register fields in both a symbolic and numeric format. When you type the R command, it searches the directory which contains the ICD program for any files with the .REG extension (REGister files). Each of these files describes a block of registers. The block name (1st line of .reg) file is displayed in a pick window. If you select a register block, each of the registers in that block has it's address, name, and description listed in a pick window. If you select a register, it's current value is read from the device and displayed both symbolically and numerically. You may then edit the register contents.

If you modified the displayed register contents and exit the register display, you are asked if you want to write the new value back to the device register. You should be careful of any "WRITE ONLY" registers, since they can not be read but will be written.

The .REG files for most NXP modules are available at nominal cost from PEmicro. These files are in ASCII and it is possible to create your own files for devices not supported by PEmicro.

19 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).

20 '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.

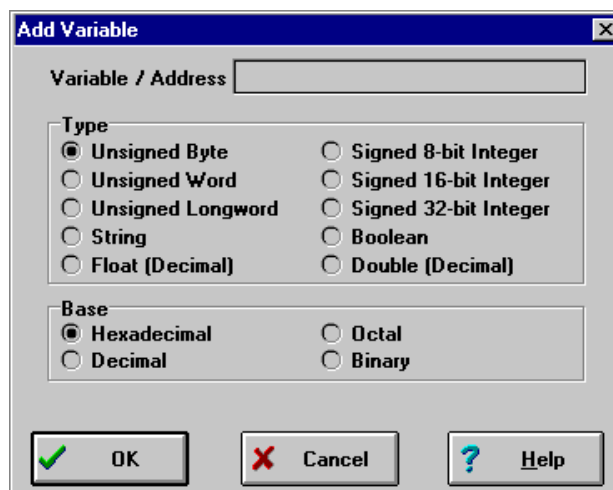


Figure 20-1: Add Variable Dialog

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 be displayed.

21 ELF/DWARF Support

The In-Circuit Debugger supports loading files of type ELF/DWARF, which provides C source-level debugging. The ICD will process files that adhere to the following specifications:

Executable and Linking Format (ELF)

System V Application Binary Interface, edition 4.1

Debug With Arbitrary Record Format (DWARF)

DWARF Debugging Information Format Revision 2.0/3.0

Note that the debugger will load DWARF version 2.0/3.0 information only. No other DWARF version is compatible with the ICD.

Some processors have ELF/DWARF supplemental specifications in addition to the general specifications above. The ICD adheres to all available processor supplements.

21.1 Supported Features

- Source-Level Code View
 - Disassembly-Level Code View
 - Single-Stepping Source
 - Running and Stopping Source
 - Loading Object Data to MCU Memory:
 - Base Type Variable View
 - Array Variable View
 - Structure/Union Variable View
 - Enumerated Type Variable View
 - Typedef Variable View
 - Global Variable View
 - Location Relative (e.g. Stack Relative) Variable View
 - Register Based Variable View
 - Variable Scoping:
 - Auto-typing of Variables in VAR Window
 - Modifying Variables
- Currently, the ICD does not support C macro information.

21.2 Getting Started

The HLOAD command loads ELF/DWARF source files.

The HLOADMAP command loads DWARF debugging information only. No executable object code is loaded.

The HSTEP command, or SS, single-steps one high-level source line. HSTEPALL performs this same action (HSTEP) on each core of a multi-core device.

The HSTEPFOR command continually steps high-level instructions until the user aborts by

pressing a key.

The HGO command starts full-speed execution and attempts to stop on a high level instruction when aborted by the user. HGOALL performs this same action (HGO) on each core of a multi-core device.

The HSTEP/HSTEPALL, HSTEPFOR, and HGO/HGOALL commands are identical to the STEP/STEPALL, STEPFOR, and GO/GOALL commands with the following exceptions:

(1) While assembly instructions are executed between the high-level language lines, the Variable, Memory, and Source Code windows are not updated. The Disassembly and CPU windows are updated for every low-level instruction. At the next high-level instruction boundary, all windows are updated (unless in GO mode).

(2) When the user aborts the current execution command, the debugger executes up to 20 steps while trying to find the next high-level language instruction boundary. The debugger will attempt to show source automatically. If, in 20 steps, it can not find the boundary of a high-level source code instruction, it will stop and show disassembly. To see source again, use the HSTEP/HSTEPALL command or right click on the Source Code window and select Show Source Module.

After loading the code, depending upon the software application, you may have to set the program counter (PC or IP) to the reset vector of your code. There is not usually high-level source code at this location. Instead, there is often compiler code used to initialize your variables, heap, and so forth. The reset code should call the "main" function. Use the command "GOTIL main" to execute code to the beginning of the "main" function. At this point, you should see source code. It is not correct to set the PC directly to the main routine, as this would skip the compiler's initialization.

21.3 Displaying Variables

The debugger Variable window will show global and static variables as well as location relative variables. A location relative variable is typically a local variables on the application stack. However, the compiler may indicate other variable types as location relative and may use many different location schemes for variables. Some variables may change location depending on the value of the PC. The ICD supports all of the DWARF 2.0 location possibilities.

The debugger will show variables whose location is a register. Compilers will often store temporary variables in CPU registers of the processor. If you attempt to look at the address of a register variable by adding the symbol &l (where l is the variable) to the Variables window, the debugger should indicate the register in which the variable is stored.

The ICD supports scoping of variables. If you put a variable name in the Variables window, the debugger will show the variable of the same name that is currently in scope. If you had a global integer variable, temp, and a local float variable, temp, within the routine init_port, the float would be shown while you step through the init_port routine. Otherwise, the Variable window would display the integer variable. When a variable value equates to [Not Accessible], this means the variable specified is either out of scope or doesn't exist.

The following symbols may be added to a variable name in the Variables window. Note that pointer variables are displayed in red.

& dereference

* reference

. access to union or structure member

-> pointer access to union or structure member

[] array subscript

For example:

```
int TintGlobal;
```

```
int *ptrTint;
```



```
int TmultiArray[3][3][3];
```

```
struct Tstruct {  
    int ii;  
    int jj;  
    short ll;  
} TstructInstance;
```

```
union Tunion {  
    unsigned long TuLongUnion;  
    struct Tstruct TstructUnion  
} TunionInstance;
```

```
union Tunion *TunionPointer;
```

ICD commands:

```
var TintGlobal  
The value of TintGlobal
```

```
var &TintGlobal  
The address of TintGlobal
```

```
var TmultiArray[0][1][2]  
The value of this element of the array
```

```
var TstructInstance.ii  
The value of this member of the structure
```

```
var TunionPointer->TuLongUnion  
The value of this union member, the union pointed to by TunionPointer
```

21.4 Examining A Variable

You may view the entire contents of a structure or array variable by double-clicking the entry in the Variables Window. Alternatively, right-click on an entry in the Variables Window and select "Examine Variable" in its popup menu.

The Examine Variable dialog is available for ELF/DWARF2.0 debugging for the following variable types:

- Structures - Displays the members of the structure.
- Arrays - Displays elements of non-dynamic arrays.
- Pointer - Displays dereferenced pointer to non-dynamic variable.
- Other types - Dialog is not available.

In the Examine Variable window, click the plus sign to expand the variable.

Figure 21-1: Simple Structure

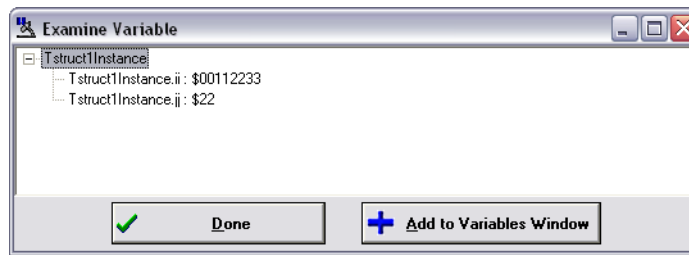
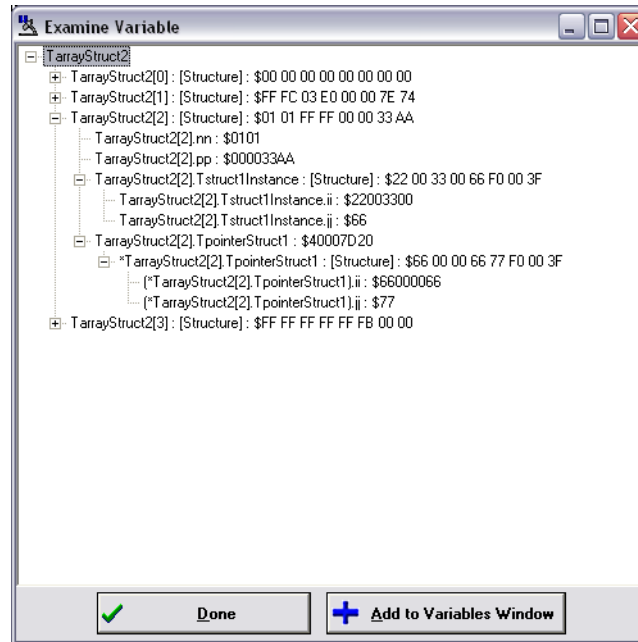


Figure 21-2: More Complex Structure



Add an array element or structure member to the Variables Window by double-clicking on the entry, or use the "Add to Variables Window" button.

21.5 ELF Program Headers

Program headers, included in every executable ELF/DWARF file, describe how the application object code is to be loaded to the target. Two values in each program header entry, defined by the System V ABI as `p_paddr` and `p_vaddr`, are available to provide a load address for a particular group of code. For executable files, the `p_vaddr` field is typically used to provide the load address. However, some compilers, such as the GNU compiler, may utilize the `p_paddr` field instead.

When the PEmicro debugger loads the ELF/DWARF file, it may detect the use of the non-standard `p_paddr` field in the ELF program header. In this instance, the debugger will display a dialog box that will ask the user what to do. Generally, when using the GNU tools, click "Yes" in the dialog box to load code using the non-standard `p_paddr` field.

For a complete description of ELF Program Headers, see the System V ABI (ELF) specification.

21.6 Loading An ELF/DWARF 2.0/3.0 Application (HLOAD/HLOADMAP)

The Elf/Dwarf 2.0 file contains two types of information:

- (1) Elf Binary Image : This contains all the instructions and data which make up the application which will run on the target microprocessor. This part of the image will eventually reside in the flash memory of the target, but during debug may also be loaded into the RAM of the target.
- (2) Dwarf Debug Information : The debug information is used by the debugger to allow the user to debug the binary image. This contains source file line number information, variable names, variable addresses, and so forth. This information is loaded into the debugger only and is not

presented to the target microcontroller.

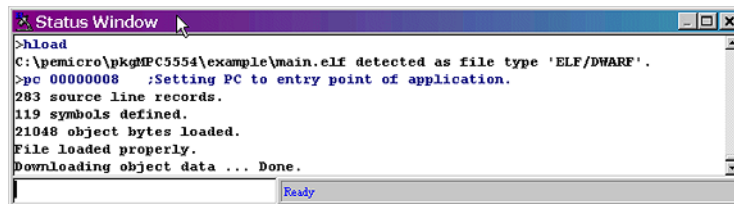
The two most common configurations for loading an Elf/Dwarf 2.0 file are:

(1) Binary image to be loaded into FLASH : If the binary image is to be loaded into flash, it must be done prior to entering ICD which does not program flash. PEmicro's PROG flash programmer may be used to program the image into flash. Upon entering the ICD, with the binary image already resident in flash, the user would use the HLOADMAP command to load the debug information portion of the Elf/Dwarf file into the debugger.

(2) Binary image needs to be loaded into RAM : If the binary image of the application is to be loaded into RAM on the target, the HLOAD command is used. The HLOAD command loads both the binary image into the target microprocessor's RAM as well as loads the dwarf debug information into the debugger. Before loading the binary image, the user should make sure that the RAM is turned on at the proper address.

The STATUS window will display the amount of debug and object information loaded from the Elf/Dwarf file, in a manner similar to the following window:

Figure 21-3: Status Window



```

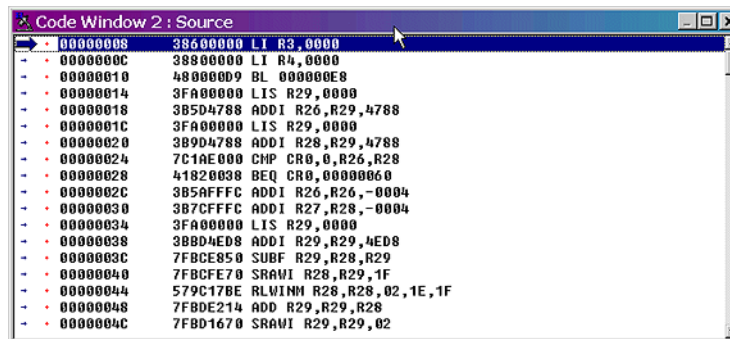
>hload
C:\pemicro\pkghPC5554\example\main.elf detected as file type 'ELF/DWARF'.
>pc 00000008 :Setting PC to entry point of application.
283 source line records.
119 symbols defined.
21048 object bytes loaded.
File loaded properly.
Downloading object data ... Done.
  
```

By default, when an Elf/Dwarf object is loaded, the program counter (PC) is set to point to the start of the demonstration application code.

21.7 Running Until The Start Of Source Code

After loading an Elf/Dwarf file, it may be that the code window still points to disassembly and does not initially show the loaded applications source code:

Figure 21-4: Code Window: Source



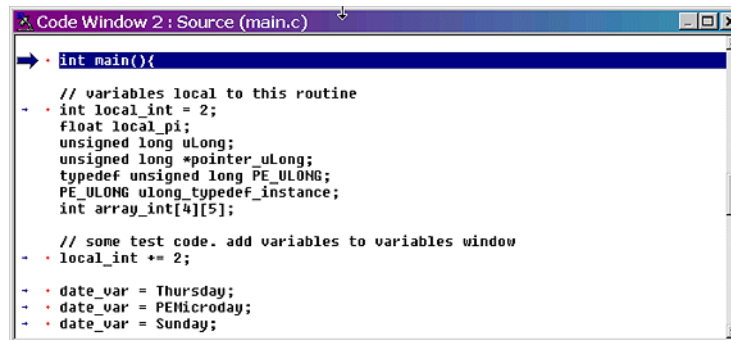
```

00000008 38600000 LI R3,0000
- 0000000C 38800000 LI R4,0000
- 00000010 48000009 BL 000000E8
- 00000014 3FA00000 LIS R29,0000
- 00000018 3B504788 ADDI R26,R29,4788
- 0000001C 3FA00000 LIS R29,0000
- 00000020 3B904788 ADDI R28,R29,4788
- 00000024 7C1AE000 CMP CR0,0,R26,R28
- 00000028 41820038 BEQ CR0,00000060
- 0000002C 3B5AFFFC ADDI R26,R26,-0004
- 00000030 3B7CFFFC ADDI R27,R28,-0004
- 00000034 3FA00000 LIS R29,0000
- 00000038 3BBD4E08 ADDI R29,R29,4E08
- 0000003C 7FBCE850 SUBF R29,R28,R29
- 00000040 7FBCE700 SRAVI R28,R29,1F
- 00000044 579C17BE RLWINH R28,R28,02,1E,1F
- 00000048 7FBDE214 ADD R29,R29,R28
- 0000004C 7FBD1670 SRAVI R29,R29,02
  
```

This is because, before running the user's main() function, the compiler must first execute some initialization code which does not have corresponding debug information. To run past the compiler initialization code, issue the "gotil main" command in the status window. Note that the labels are case sensitive and that the main label should be all lowercase. This will set a breakpoint at the beginning of the main() routine in main.c and start the processor running. Execution should stop almost immediately and the PC should be pointing to valid source code. This source code will

appear in the source code window, similar to the following:


Figure 21-5: Code Window: Source (main.c)



```
Code Window 2 : Source (main.c)
- int main()
- // variables local to this routine
- int local_int = 2;
- float local_pi;
- unsigned long uLong;
- unsigned long *pointer_uLong;
- typedef unsigned long PE_ULONG;
- PE_ULONG ulong_typedef_instance;
- int array_int[4][5];
- // some test code. add variables to variables window
- local_int += 2;
- date_var = Thursday;
- date_var = PEMicroday;
- date_var = Sunday;
```

Also, instead of using the GOTIL command, the user could have stepped through the initialization code (using the STEP or HSTEP commands) and would have eventually reached the main() function.

21.8 Stepping Through C-Level Instructions

The PEmicro debugger implements a high-level language source step command, which may be executed by using the HSTEP command in the status window or by clicking the high-level step button  on the debugger button bar. Each time the high-level language source is stepped, the debugger will rapidly single step assembly level instructions until the next source instruction is encountered, at which point execution will cease. While the debugger is fast single-stepping, the only on-screen value which will be updated is the PC (by default). When the debugger reaches the next source instruction, all visible windows will be updated with data read from the target. Note that using the HSTEP command does not run code in real-time. Real-time execution is described in the next section.

The user should step several source-level instructions as this point. Note that some instructions will take longer to step than others, because each C level instruction may consist of a greater or fewer number of underlying assembly instructions than others.