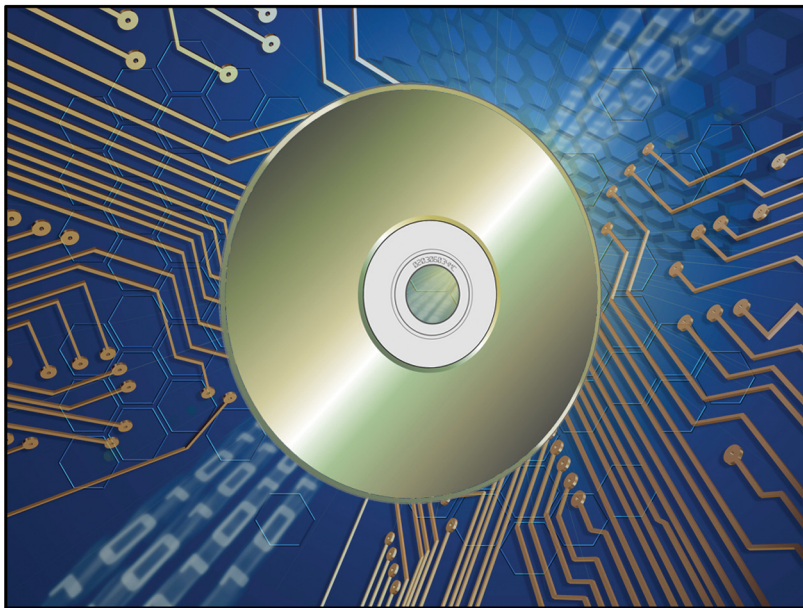




ICDS12ZZ

USER MANUAL



Purchase Agreement

This software and accompanying documentation are protected by United States Copyright law and also by International Treaty provisions. Any use of this software in violation of copyright law or the terms of this agreement will be prosecuted.

All the software in this package is copyrighted by P&E Microcomputer Systems, Inc. Copyright notices have been included in the software.

P&E Microcomputer Systems authorizes you to make archival copies of this software for the sole purpose of back-up and protecting your investment from loss. Under no circumstances may you copy this software or documentation for the purpose of distribution to others. Under no conditions may you remove the copyright notices from this software or documentation.

This software may be used by one person on up to two different computers, provided that the software is never used on the two computers at the same time. P&E expects that group programming projects making use of this software will purchase a copy of the software and documentation for each user in the group. Contact P&E for volume discounts and site licensing agreements.

With respect to the physical media provided within, P&E Microcomputer Systems warrants the same to be free of defects in materials and workmanship for a period of 30 days from the date of receipt. If you notify us within the warranty period, P&E Microcomputer Systems will update the defective media at no cost.

P&E Microcomputer Systems does not assume any liability for the use of this product beyond the original purchase price. In no event will P&E Microcomputer Systems be liable for additional damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use these programs, even if P&E Microcomputer Systems has been advised of the possibility of such damage.

By installing or using this software, you agree to the terms of this agreement. If you do not agree with these terms, you should not install this software.

©2013 P&E Microcomputer Systems, Inc.

MS-DOS & Windows are registered trademarks of Microsoft Corporation. IBM is a registered trademark of IBM corporation.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

P&E Microcomputer Systems, Inc.
98 Galen St.
Watertown, MA 02472
617-923-0053
<http://www.pemicro.com>

Manual version 1.00
October 2013

1	OVERVIEW.....	1
1.1	Features	2
1.2	System Requirements	2
2	USER INTERFACE.....	1
2.1	Button Taskbar	2
2.2	S12Z (CPU) Window	2
2.3	Variables Window.....	5
2.4	Memory Window.....	8
2.5	Code Window	10
2.6	Status Window	16
2.7	Additional Menu- or Command-Accessible Windows	17
3	START-UP CONFIGURATION.....	1
4	ELF/DWARF SUPPORT	1
4.1	Supported Features.....	1
4.2	Getting Started	2
4.3	Displaying Variables.....	3
4.4	ELF Program Headers	5
4.5	Loading An ELF/DWARF 2.0 Application (HLOAD/HLOADMAP).....	5
4.6	Running Until the Start Of Source Code	7
4.7	Stepping Through C Source-Level Instructions.....	8
5	COMMANDS.....	1
5.1	ASCIIF3 and ASCIIF6 - Toggle Memory Window	1
5.2	BELL - Sound Bell	2
5.3	BGND_TIME - Log Time Since Last BGND Instruction	2
5.4	BLOCK FILL or BF- Fill Memory Block.....	3
5.5	BR - Set Break Point	4
5.6	C - Set/Clear C Bit.....	5
5.7	CAPTURE - Open Capture File.....	6
5.8	CAPTUREOFF - Turn Off Capture	6
5.9	CCR - Set Condition Code Register	7
5.10	CLEARMAP - Clear Map File	8



5.11	CLEARSYMBOL - Clear User Symbols	8
5.12	CLEARVAR - Clear Variables Window	9
5.13	CODE - Show Disassembled Code	9
5.14	COLORS - Set Colors of Debugger	10
5.15	D0 – Set Data Register 0	10
5.16	D1 – Set Data Register 1	10
5.17	D2 – Set Data Register 2	11
5.18	D3 – Set Data Register 3	11
5.19	D4 – Set Data Register 4	12
5.20	D5 – Set Data Register 5	12
5.21	D6 – Set Data Register 6	13
5.22	D7 – Set Data Register 7	13
5.23	DASM - Disassemble Memory	13
5.24	DUMP - Dump Data Memory to Screen.....	15
5.25	DUMP_TRACE - Dump Trace Buffer.....	16
5.26	EVAL - Evaluate Expression.....	16
5.27	EXIT or QUIT - Exit Program	17
5.28	G or GO or RUN - Begin Program Execution	18
5.29	GOEXIT - Begin Program Execution & Quit Debugger.....	18
5.30	GONEXT - Execute From PC Until Next Instruction	19
5.31	GOTIL - Execute Program until Address	19
5.32	GOTILROM - Fast Single Step Til Address	20
5.33	HELP - Open Help File	20
5.34	HGO - Begin Program Execution.....	21
5.35	HLOAD - Load ELF/DWARF Object	21
5.36	HLOADMAP - Load Source-Level Debug Map.....	22
5.37	HSTEP - High-Level Language Source Step.....	22
5.38	HSTEPFOR - Step Forever (High-Level Language).....	23
5.39	I - Set/Clear I Bit	23
5.40	INFO - Display Line Information	24
5.41	IPL - Set/Clear IPL Bits	25
5.42	IX - Set Index Register X	26

5.43	IY - Set Index Register Y	26
5.44	LF or LOGFILE - Open / Close Log File	27
5.45	LISTOFF - Do Not Show Info During Steps	28
5.46	LISTON - Show Info during Steps	28
5.47	LOAD - Load S19 and MAP	28
5.48	LOAD_BIN - Load Binary File	29
5.49	LOADV_BIN - Execute LOAD_BIN, Then VERIFY	30
5.50	LOADALL - Execute LOAD, Then LOADMAP	30
5.51	LOADDESK - Load Desktop Settings	31
5.52	LOADMAP - Load Map File	31
5.53	LOADV - Executes LOAD, Then VERIFY	32
5.54	MACRO or SCRIPT - Execute a Batch File	32
5.55	MACROEND - Stop Saving Commands to File	33
5.56	MACROSTART - Save Debug Commands to File	34
5.57	MACS - List Macros	34
5.58	MD or SHOW - Display Memory, Window 1	35
5.59	MD2 - Display Memory, Window 2	35
5.60	MDF3 / MDF6 or SHOWF3 / SHOWF6	36
5.61	MM or MEM - Modify Memory	36
5.62	N - Set/Clear N Bit	38
5.63	PC - Set Program Counter	39
5.64	PDUMP - Dump Program Memory To Screen	40
5.65	QUIET - Toggle Refresh Of Memory-Based Windows	40
5.66	QUIT - Exit Program	41
5.67	R or REG or STATUS - Show Registers (If Applicable)	42
5.68	REM - Place Comment in Macro File	42
5.69	RESET - Reset MCU	42
5.70	RTVAR - Display Variable Contents In Real Time	43
5.71	RUNNING - Re-Enter Debugger With CPU Running	44
5.72	S - Set/Clear S Bit	44
5.73	SAVEDESK - Save Desktop Settings	45
5.74	SERIAL - Set Up Serial Port Parameters	46



5.75	SERIALON - Turn Communication Window Into Terminal	46
5.76	SERIALOFF - Turn Off Serial Port During GO.....	47
5.77	SHOWCODE - Display Code at Address.....	47
5.78	SHOWMAP or MAP - Show Information in Map File	48
5.79	SHOWPC - Display Code at PC	48
5.80	SHOWTRACE - View Trace Buffer.....	48
5.81	SNAPSHOT - Take Screen Snapshot	49
5.82	SOURCE - Toggle Source/Disassembled Code.....	49
5.83	SOURCEPATH - Path For Source Code	50
5.84	SP - Stack Pointer	50
5.85	SS - Single Step Source-Level Code.....	50
5.86	ST or STEP or T - Single Step.....	51
5.87	STEPFOR - Step Forever.....	52
5.88	STEPTIL - Single Step to Address	52
5.89	SYMBOL - Add Symbol	52
5.90	TRACE - Capture Trace Data	53
5.91	U - Set/Clear U Bit	54
5.92	UPLOAD_SREC - Upload S-Record to Screen.....	55
5.93	V - Set/Clear V Bit.....	56
5.94	VAR - Display Variable	57
5.95	VERSION or VER - Display Software Version.....	58
5.96	VERIFY - Compare Program Memory With S-Record.....	58
5.97	WHEREIS - Display Symbol Value	59
5.98	X - Set/Clear X Bit.....	59
5.99	Z - Set/Clear Z Bit	60

1 OVERVIEW

P&E's ICDS12ZZ software is an in-circuit debugger for Freescale's S12Z microcontrollers. ICDS12ZZ communicates with the target device via one of P&E's hardware interfaces, and uses the S12Z device's background debug mode (BDM) to give the user access to all on-chip resources.

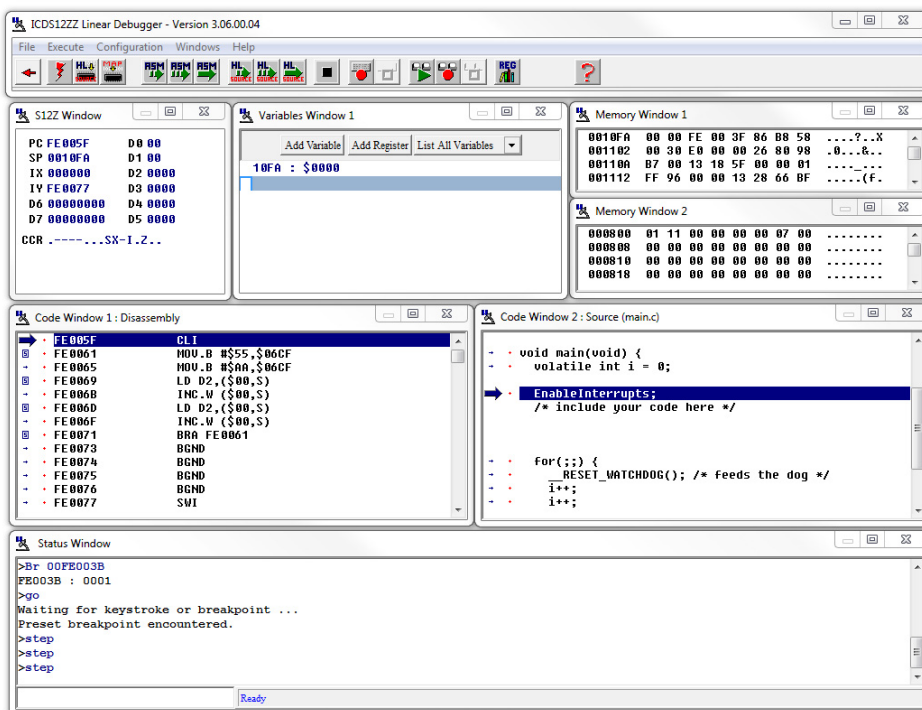


Figure 1-1: ICDS12ZZ Software



1.1 Features

The ICDS12ZZ software includes the following features:

- Breakpoints with counters on the Nth execution
- Variables Window showing bytes, words
- Real-time execution and multiple tracing modes
- Startup and macros files for automating the debug process (useful for creating repeatable testing, calibration, or debug procedures).
- Timing measurement and execution count functions
- Support for symbolic register files
- Full assembly source-level debugging
- C source-level debugging using the ELF/DWARF v.2.0 or ImageCraft v.1.1 debug file format

1.2 System Requirements

ICDS12ZZ requires Windows NT, 2000, XP, Vista, 7 (32-bit or 64-bit), or 8 (32-bit or 64-bit).

2 USER INTERFACE

This chapter discusses each of the Windows and the elements of the Taskbar that comprise the ICDS12ZZ User Interface. Below is a screen snapshot of a typical ICDS12ZZ layout.

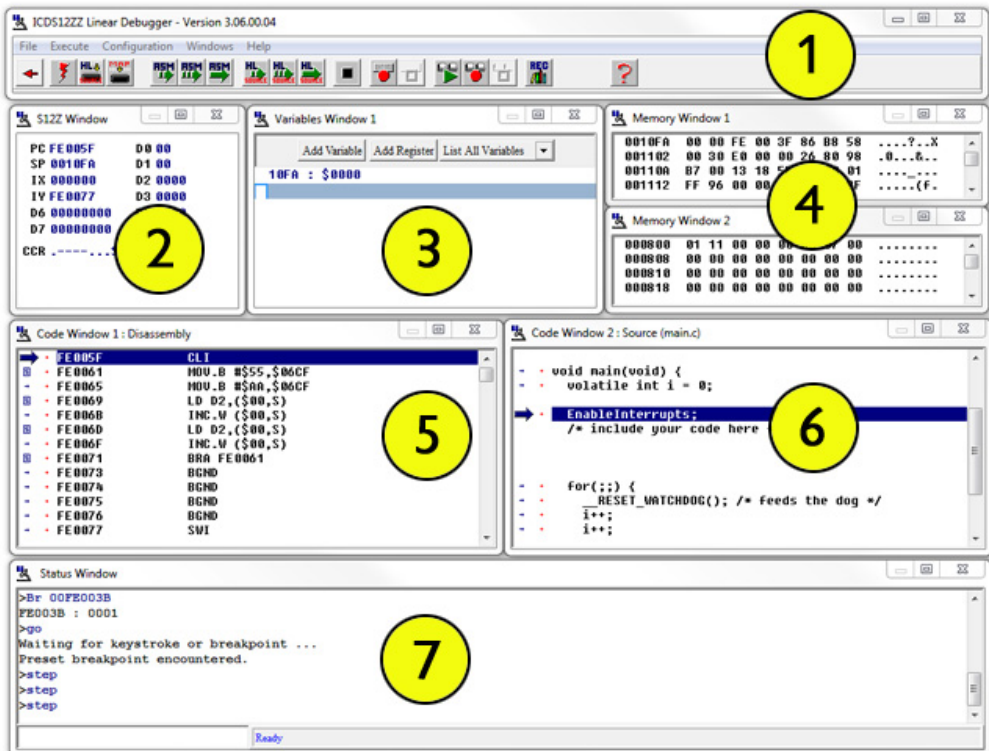


Figure 2-1: ICDS12ZZ User Interface

The labelled areas of the User Interface image above are:

1. [Button Taskbar](#)
2. [S12Z \(CPU\) Window](#)
3. [Variables Window](#)
4. [Memory Window](#)
5. [Code Window](#) - Disassembly
6. [Code Window](#) - Source
7. [Status Window](#)

Below you will find detailed descriptions of each section of the ICDS12ZZ user interface.

2.1 Button Taskbar

ICDS12ZZ features a taskbar with several buttons that execute various functions. They are organized in sets with similar functions. Mouse-over an individual button when using ICDS12ZZ to see that button's function.



Figure 2-1: Button Taskbar

2.2 S12Z (CPU) Window

The CPU Window displays the current state of the S12Z CPU registers. The popup window allows modification of these value.

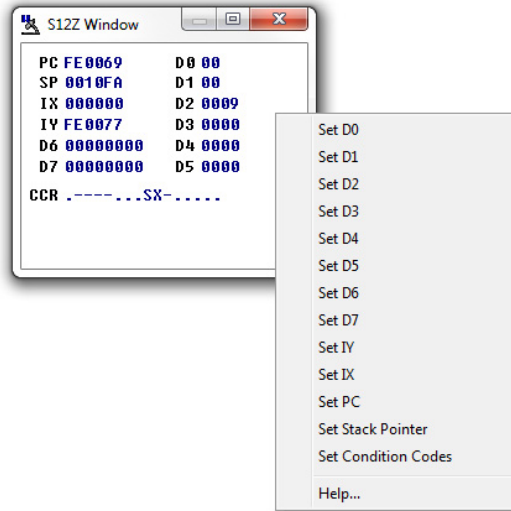


Figure 2-2: CPU Window

2.2.1 Pop-Up Menu

By pressing the RIGHT MOUSE BUTTON while the cursor is over the CPU window, the user is given a pop-up menu which has the following options:

Set Data Register 0..7

Sets the desired data register to a user defined value. Upon selecting this option, the user is prompted for a value.

Set Index Register X

Sets index register X to a user defined value. Upon selecting this option, the user is prompted for a value.

Set Index Register Y

Sets index register Y to a user defined value. Upon selecting this option, the user is prompted for a value.

Set PC

Sets the Program Counter (PC) to a user defined value. Upon selecting this option, the user is prompted for a value.

Set Stack Pointer

Sets the Stack Pointer (SP) to a user defined value. Upon selecting this option, the user is prompted for a value.

Set Condition Codes

Allows the user to toggle bits within the CCR. Upon selecting this option, the CCR Modification Window is displayed.

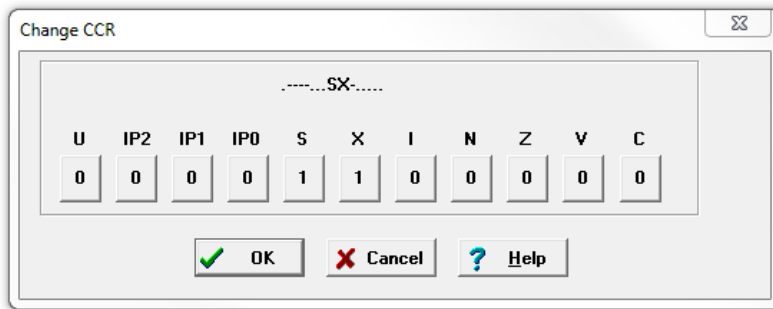


Figure 2-3: CCR Modification Window

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

2.3 Variables Window

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

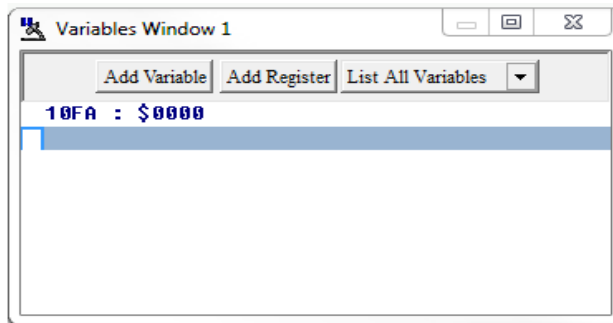


Figure 2-4: Variables Window

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

Note: A second Variable Window (labelled "Variables Window 2") can be opened using the menu.

2.3.1 Adding And Deleting Variables

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

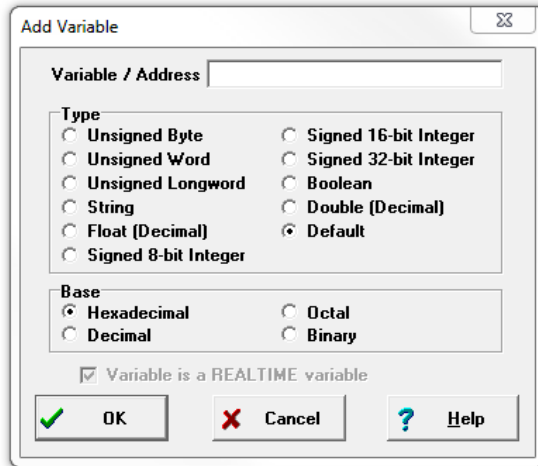


Figure 2-1: Add Variable Dialog

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

2.3.2 Modifying A Variable’s Value

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

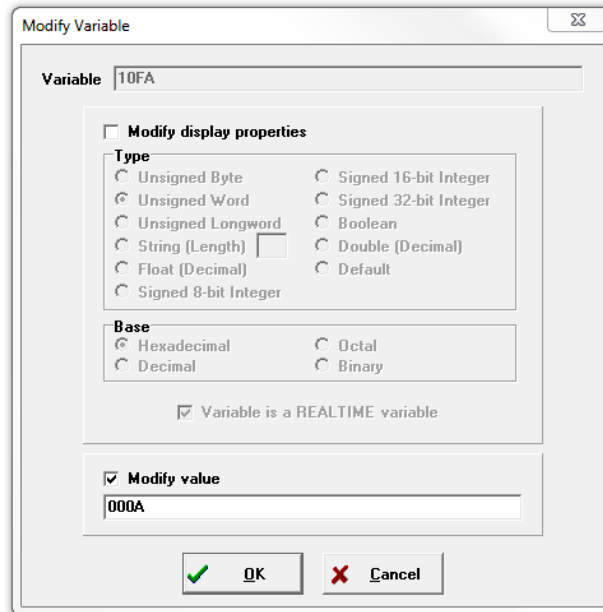


Figure 2-2: Modify Variable Value

2.3.3 Modifying A Variable's Properties

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

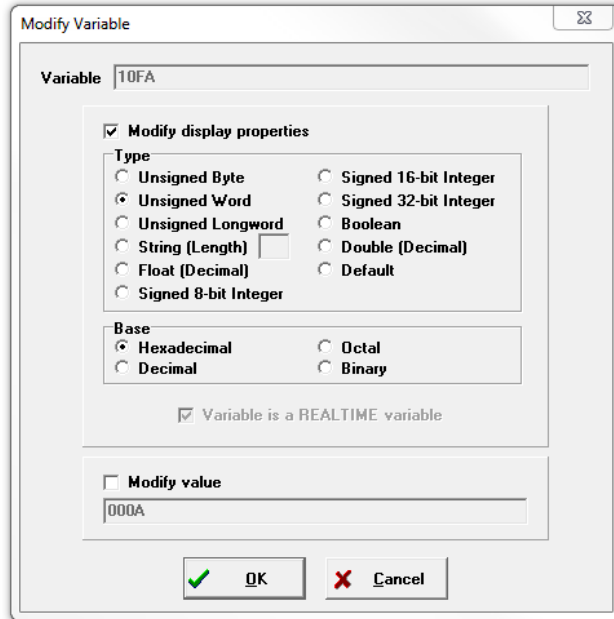


Figure 2-3: Modify Variable Properties

2.4 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.

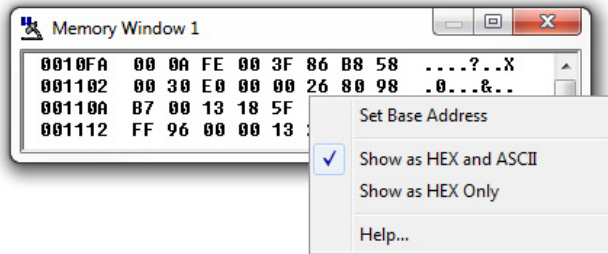


Figure 2-4: Memory Window

2.4.1 Pop-Up 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.

2.4.2 Keystrokes

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

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

2.5 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".

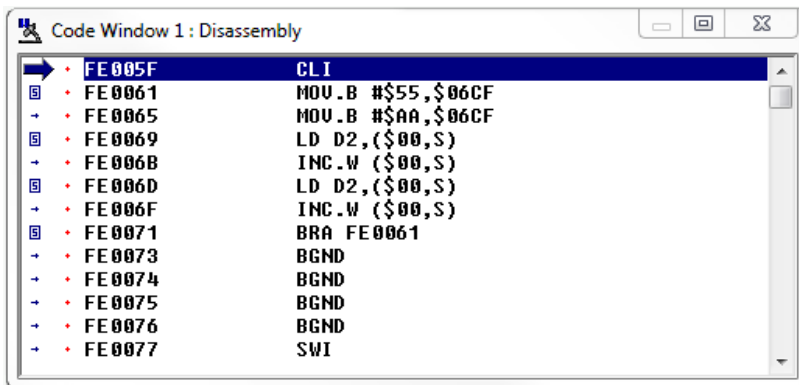
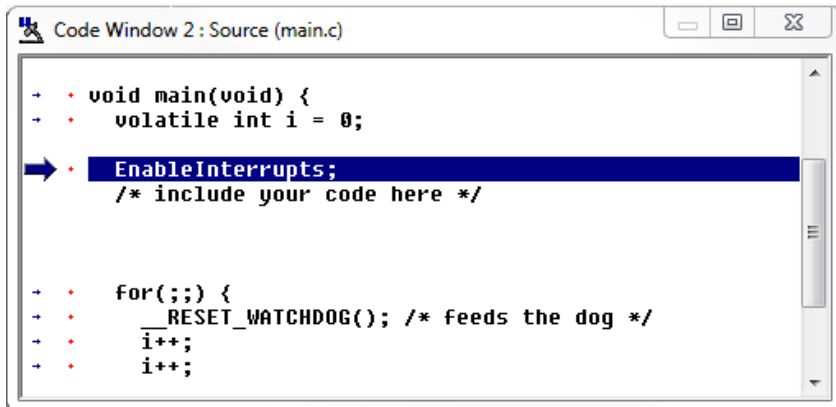


Figure 2-5: Code Window: Disassembly

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).



```

Code Window 2: Source (main.c)
+ * void main(void) {
+ *   volatile int i = 0;
+ *   EnableInterrupts;
+ *   /* include your code here */
+ *
+ *   for(;;) {
+ *     __RESET_WATCHDOG(); /* feeds the dog */
+ *     i++;
+ *     i++;
  
```

Figure 2-6: Code Window: Source

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.

2.5.1 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:

```

+ . For(;;) {
+ .     RESET WATCHDOG(); /* feeds the dog */
+ .     i++;
+ .     i++;
+ . } /* loop forever */
+ . /* please make sure that you never leave main */
+ . }
    
```

Figure 2-7: Example: Breakpoint Set

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.



Figure 2-8: HGO Button

If the debugger encounters a breakpoint, execution will stop on this source line. If a breakpoint is not encountered, execution will continue until the user presses a key or uses the stop button on the debugger button bar. To remove a breakpoint, double-click the large red stop sign.

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

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

➔	• FE0069	LD D2,(\$00,S)
+	• FE006B	INC.W (\$00,S)
☐	• FE006D	LD D2,(\$00,S)
-	• FE006F	INC.W (\$00,S)
☐	• FE0071	BRA FE0061

Figure 2-9: Example: Source-Level Start Indicator

2.5.2 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:

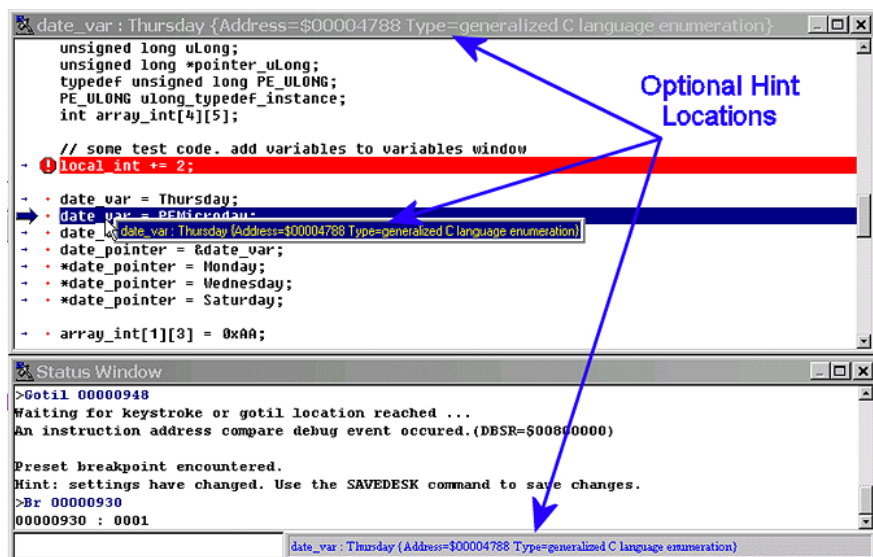


Figure 2-10: Hint Locations

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 2-11: Close-Up: Hint Box

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

2.5.3 Pop-Up Menu

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

Toggle Breakpoint at Cursor

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

Set PC at Cursor

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

Gotil Address at Cursor

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

Set Base Address

This option allows the code window to look at different locations in the user's code, or anywhere in the memory map. The user will be prompted to enter an address or label to set

the code window's base address. This address will be shown as the top line in the Code Window. This option is equivalent to the SHOWCODE command.

Set Base Address to PC

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

Select Source Module

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

Show Disassembly or Show Source/Disassembly

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

Help

Displays this help topic.

2.5.4 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

2.6 Status Window

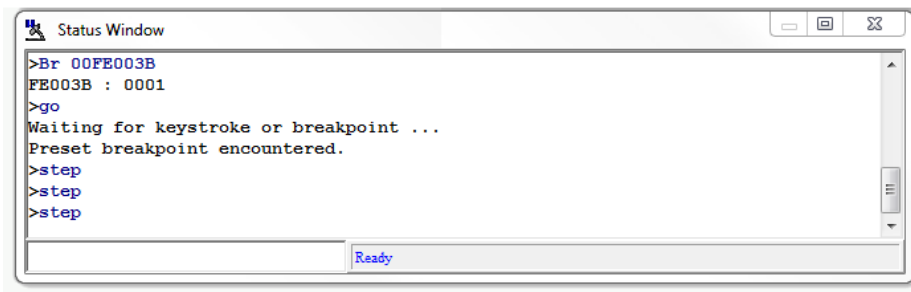


Figure 2-12: Status Window

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.

2.6.1 Command Recall

You can use the PgUp and PgDn keys to scroll through the past 30 commands issued in the status 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.

2.6.2 Pop-Up 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.

2.6.3 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.

2.7 Additional Menu- or Command-Accessible Windows

2.7.1 Breakpoint Window

The Breakpoint Window allows you to view the address of where your breakpoints are set, the count, line number and source path for each.



Figure 2-13: Breakpoint Window

2.7.2 Change Window Colors Window

The Change Window 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.

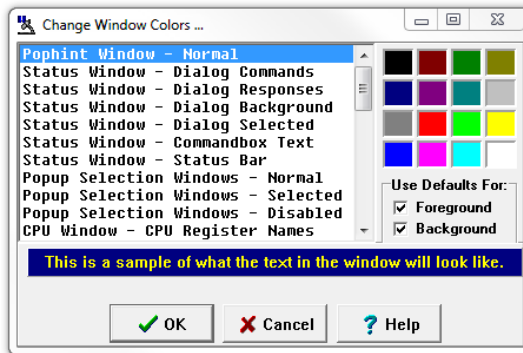


Figure 2-14: Colors Window

2.7.3 Debug Options

The Debug Options Window allow you to set various parameters for the debugger operation, including whether devices are auto-detected and which type of breakpoints to set.

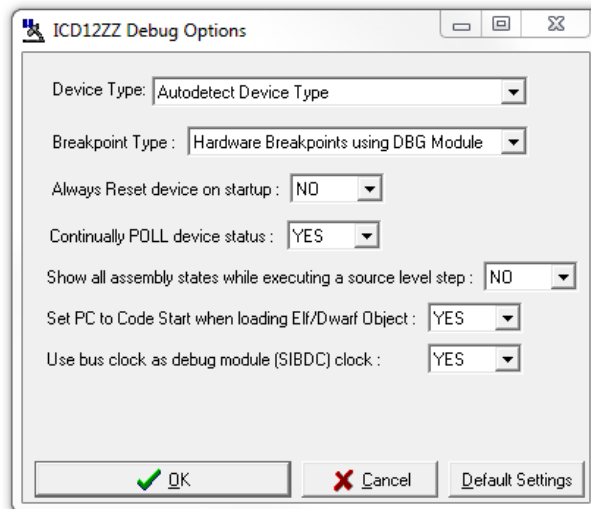


Figure 2-15: Debug Options Window

2.7.4 Stack Window

The Stack Window shows values that have been pushed on the stack, the stack pointer value, along with CPU results if a RTI or RTS instruction is executed at that time.

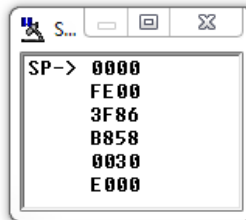


Figure 2-16: Stack Window

During an interrupt, the stack window also shows the interrupt stack: the top fifteen values of the stack, plus the values of the condition code register (CCR), accumulator (A) and index register (X). This information indicates the restored state of the stack upon the return from the interrupt. During execution of a subroutine, the stack window also shows the subroutine stack, which indicates the restored state of the stack upon return from a subroutine.

Note: S12Z MCUs store information in the stack (1) during an interrupt or (2) during execution of a subroutine. The stack window shows both these possible interpretations of stack data. You must know whether program execution is in an interrupt or in a subroutine, to know which stack data interpretation is valid.

2.7.5 Trace Window

The Trace Window allows the user to view the contents of the trace buffer. After executing a TRACE command, you may use the SHOWTRACE command or hit F7 to view the trace buffer.

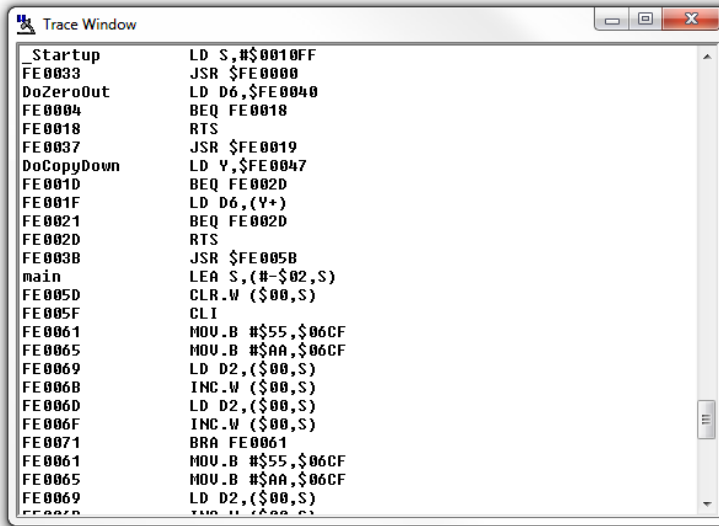


Figure 2-17: Trace Buffer Window

3 START-UP CONFIGURATION

To setup ICDS12ZZ to run with certain command line parameters, highlight the ICDS12ZZ icon and select PROPERTIES from the Program Manager File Menu.

Note: Everything on a command line after and including the semicolon “;” character is considered a comment. This helps in documenting macro (script) files.

Syntax:

ICDS12ZZ [option] ... [option]

Optional parameters [option] are as follows:

running Starts ICD with CPU running (see Running help)

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:

ICDS12ZZ running quiet Starts ICD with CPU running and without filling the Memory Windows or the Disassembly (Source) Window.

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

4 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 2.0.0

Note that the debugger will load DWARF version 2.0 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.

4.1 Supported Features

The following are C source-level (ELF/DWARF) features supported by the ICD software:

- 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.

4.2 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.

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.

The HSTEP, HSTEPFOR, and HGO commands are identical to the STEP, STEPFOR, and GO 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 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.

4.3 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 &I (where I 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

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

4.4 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 P&E 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.

4.5 Loading An ELF/DWARF 2.0 Application (HLOAD/HLOADMAP)

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

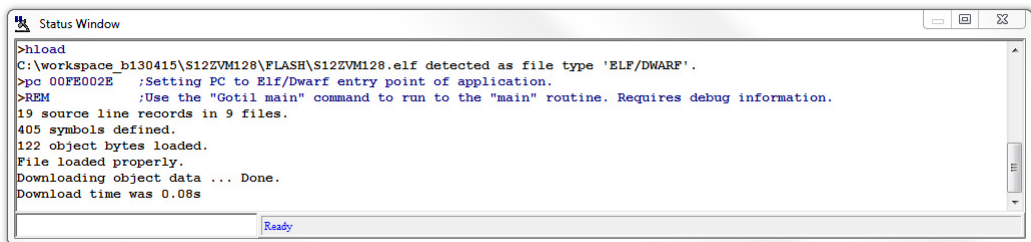
(1) Elf Binary Image : This contains all the instructions and data which make up the application which will run on the target microprocessor. This part of the image will eventually reside in the flash memory of the target, but during debug may also be loaded into the RAM of the target.

(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. P&E'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:



```

>hload
C:\workspace_b130415\S12ZVM128\FLASH\S12ZVM128.elf detected as file type 'ELF/DWARF'.
>pc 00FE002E ;Setting PC to Elf/Dwarf entry point of application.
>REM ;Use the "Got1 main" command to run to the "main" routine. Requires debug information.
19 source line records in 9 files.
405 symbols defined.
122 object bytes loaded.
File loaded properly.
Downloading object data ... Done.
Download time was 0.08s

```

Ready

Figure 4-1: Status Window

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.

4.6 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:

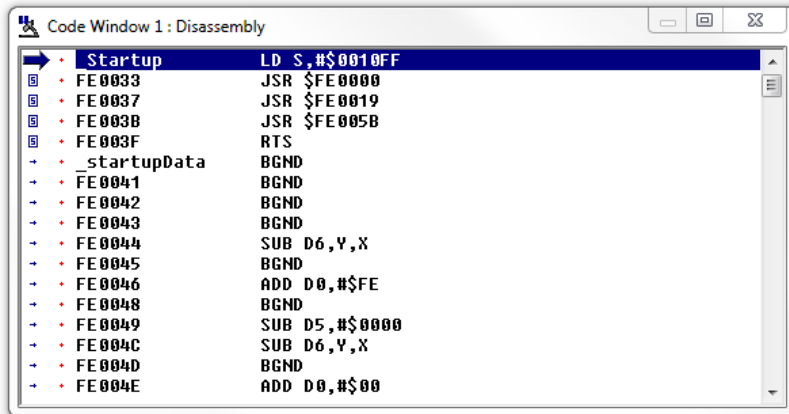
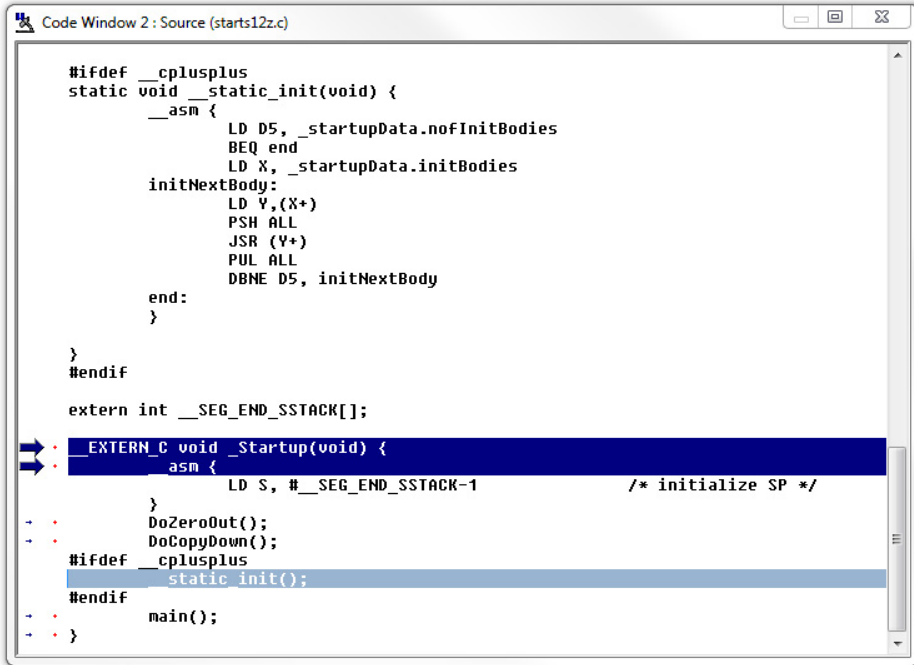


Figure 4-2: Code Window - Disassembly Shown

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:



```

Code Window 2 : Source (starts12z.c)

#ifdef __cplusplus
static void __static_init(void) {
    __asm {
        LD D5, _startupData.nofInitBodies
        BEQ end
        LD X, _startupData.initBodies
    initNextBody:
        LD Y,(X+)
        PSH ALL
        JSR (Y+)
        PUL ALL
        DBNE D5, initNextBody
    end:
    }
}
#endif

extern int __SEG_END_SSTACK[];

EXTERN_C void _Startup(void) {
    __asm {
        LD S, #__SEG_END_SSTACK-1          /* initialize SP */
    }
    DoZeroOut();
    DoCopyDown();
#ifdef __cplusplus
    static init();
#endif
    main();
}

```

Figure 4-3: Code Window - Source Code Shown

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.

4.7 Stepping Through C Source-Level Instructions

The P&E 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 MPC5554 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 at 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.



5 COMMANDS

The following are valid commands for the ICDS12ZZ debugger. Please note the following regarding nomenclature:

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 to be fetched (Equals IP+6).

str ASCII string.

; Everything on a command line after and including the “;” character is considered a comment. This helps in documenting macro (script) files.

5.1 ASCIIIF3 and ASCIIIF6 - Toggle Memory Window

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

ASCIIIF3 toggles memory window 1.

ASCIIIF6 toggles memory window 2.

Syntax:

ASCIIF3

Example:

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

5.2 BELL - Sound Bell

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

Syntax:

BELL [<n>]

Where:

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

Example:

```
>BELL 3      Ring PC bell 3 times.
```

5.3 BGND_TIME - Log Time Since Last BGND Instruction

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 This command will give you four time differences (t1,t2,t3,t4).

```
PC----->BGND1----->BGND2----->BGND3----->BGND4
<-----t1-----><-----t2-----><-----t3-----><-----t4----->
```

5.4 BLOCK FILL or BF- Fill Memory Block

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/word/lon value to be stored in the specified block.

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

.B Each byte of the block receives the value.

.W Each word of the block receives the value.

.L Each long of the block receives the value.

Examples:

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

5.5 BR - Set Break Point

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.

You may also add/remove breakpoints by double-clicking on the red dot to the left of a particular line of code in the Code Window.

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.

5.6 C - Set/Clear C Bit

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

Note:

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

- U — User/Supervisor mode status/control, CCR bit 15
- IPL — Interrupt Priority Level, CCR bits 10~8
- S — Stop mode enable, CCR bit 7
- X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6
- I — I Interrupt mask, CCR bit 4
- N — Negative status flag, CCR bit 3
- Z — Zero status flag, CCR bit 2

V — Two's complement overflow status flag, CCR bit 1

C — Carry/borrow status flag, CCR bit 0A 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 Clears the C bit of the CCR.

>C 1 Sets the C bit of the CCR.

5.7 CAPTURE - Open Capture File

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".

5.8 CAPTUREOFF - Turn Off Capture

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.

5.9 CCR - Set Condition Code Register

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

Note:

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

- U — User/Supervisor mode status/control, CCR bit 15
- IPL — Interrupt Priority Level, CCR bits 10~8
- S — Stop mode enable, CCR bit 7
- X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6
- I — I Interrupt mask, CCR bit 4
- N — Negative status flag, CCR bit 3
- Z — Zero status flag, CCR bit 2
- V — Two's complement overflow status flag, CCR bit 1
- C — Carry/borrow status flag, CCR bit 0 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 01 Assign the value 01 to the CCR. This makes the binary pattern 00000001; the C bit set, other bits clear.

5.10 CLEARMAP - Clear Map File

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

Syntax:

CLEARMAP

Example:

>CLEARMAP Clears symbol and source information.

5.11 CLEARSYMBOL - Clear User Symbols

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

Note:

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

Syntax:

CLEARSYMBOL

Example:

>CLEARSYMBOL Clears user defined symbols.

5.12 CLEARVAR - Clear Variables Window

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

Syntax:

CLEARVAR

Example:

CLEARVAR Removes all the variables for the variables window.

5.13 CODE - Show Disassembled Code

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

Syntax:

CODE <add>

Where:

<add> Address where your code begins.

Example:

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

5.14 COLORS - Set Colors of Debugger

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.

5.15 D0 – Set Data Register 0

The D0 command sets the data register 0 to the specified value. The data register 0 is 8 bits wide.

Syntax:

D0 <value>

Where:

<value> The new value for the D0 register.

Example:

>D0 01 Set the data register 0 value to 01.

5.16 D1 – Set Data Register 1

The D1 command sets the data register 1 to the specified value. The data register 1 is 8 bits wide.

Syntax:

D1 <value>

Where:

<value> The new value for the D1 register.

Example:

>D1 02 Set the data register 1 value to 02.

5.17 D2 – Set Data Register 2

The D2 command sets the data register 2 to the specified value. The data register 2 is 16 bits wide.

Syntax:

D2 <value>

Where:

<value> The new value for the D2 register.

Example:

>D2 0102 Set the data register 2 value to 0102.

5.18 D3 – Set Data Register 3

The D3 command sets the data register 3 to the specified value. The data register 3 is 16 bits wide.

Syntax:

D3 <value>

Where:

<value> The new value for the D3 register.



Example:

>D3 0103 Set the data register 3 value to 0103.

5.19 D4 – Set Data Register 4

The D4 command sets the data register 4 to the specified value. The data register 4 is 16 bits wide.

Syntax:

D4 <value>

Where:

<value> The new value for the D4 register.

Example:

>D4 0104 Set the data register 4 value to 0104.

5.20 D5 – Set Data Register 5

The D5 command sets the data register 5 to the specified value. The data register 5 is 16 bits wide.

Syntax:

D5 <value>

Where:

<value> The new value for the D5 register.

Example:

>D5 0105 Set the data register 5 value to 0105.

5.21 D6 – Set Data Register 6

The D6 command sets the data register 6 to the specified value. The data register 6 is 32 bits wide.

Syntax:

D6 <value>

Where:

<value> The new value for the D6 register.

Example:

>D6 040A0106 Set the data register 6 value to 040A0106.

5.22 D7 – Set Data Register 7

The D7 command sets the data register 7 to the specified value. The data register 7 is 32 bits wide.

Syntax:

D7 <value>

Where:

<value> The new value for the D7 register.

Example:

>D7 040A0107 Set the data register 7 value to 040A0107.

5.23 DASM - Disassemble Memory

The DASM command disassembles machine instructions, displaying the addresses and their contents as disassembled instructions in the status window. The memory locations

between the first and second addresses (add) are uploaded to the screen in the form of Bytes, Words, or Longwords. The first address must be on an even boundary for Words or Longwords. If the capture feature is active, the lines of dumped data are also sent to the capture file. Data is read as Bytes, Words, or Longwords from the data space.

- If the command includes an address value, one disassembled instruction is shown, beginning at that address.
- If the command is entered without any parameter values, the software finds the most recently disassembled instruction then shows the next instruction, disassembled.
- If the command includes startrange and endrange values, the software shows disassembled instructions for the range.

Note:

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

Syntax:

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

Where:

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

Examples:

>DASM 300

0300	A6E8	LDA #0E8
0302	B700	STA PORTA
0304	A6FE	LDA #FE

>DASM 400 408

0400	5F	CLR X
0401	A680	LDA #80
0403	B700	STA PORTA
0405	A6FE	LDA #FE
0407	B704	STA DDRA

5.24 DUMP - Dump Data Memory to Screen

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

Note:

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

Syntax:

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

Where:

<startrange> Beginning address of the data memory block.

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

<n> Optional number of bytes, words, or longs to be written on one line.

Examples:

>DUMP C0 CF Dump array of RAM data memory values, in bytes.

>DUMP.W 400 47F Dump ROM code from data memory hex addresses 400 to 47F in words.

>DUMP.B 300 400 8 Dump contents of data memory hex addresses 300 to 400 in rows of eight bytes.

5.25 DUMP_TRACE - Dump Trace Buffer

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

Syntax:

Dump_Trace

Example:

>Dump_Trace

5.26 EVAL - Evaluate Expression

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

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

Syntax:

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

Where:

- <n> Alone, the numerical term to be evaluated. Otherwise either numerical term of a simple expression.
- <op> The arithmetic operator (+, -, *, /, &, or ^) of a simple expression to be evaluated.

Examples:

>EVAL 45 + 32

004DH 077T 000115O 0000000001001101Q "w"

>EVAL 100T

0064H 100T 000144O 0000000001100100Q "d"

5.27 EXIT or QUIT - Exit Program

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

Syntax:

EXIT

Example:

>EXIT Finish working with the program.

5.28 G or GO or RUN - Begin Program Execution

The G or GO or RUN command starts execution of code in the Debugger at the current Program Counter (PC) address, or at an optional specified address. When only one address is entered, that address is the starting address. When a second address is entered, execution stops at that address. The G or GO or RUN commands are identical. When only one address is specified, execution continues until a key or mouse is pressed, a breakpoint set with a BR command occurs, or an error occurs.

Syntax:

GO [<startaddr> [<endaddr>]]

Where:

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

<endaddr> Optional execution ending address.

Examples:

>GO Begin code execution at the current PC value.

>GO 346 Begin code execution at hex address 346.

>G 400 471 Begin code execution at hex address 400. End code execution just before the instruction at hex address 471.

>RUN 400 Begin code execution at hex address 400.

5.29 GOEXIT - Begin Program Execution & Quit Debugger

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

Syntax:

GOEXIT [add]

Where:

addStarting 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.

5.30 GONEXT - Execute From PC Until Next Instruction

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.

5.31 GOTIL - Execute Program until Address

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

You may also execute a GOTIL to a particular line of code by double-clicking on the blue arrow to the left of that line of code in the Code Window.

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.

5.32 GOTILROM - Fast Single Step Til Address

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.

5.33 HELP - Open Help File

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

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

Syntax:

HELP [<topic>]

Where:

<topic> a debug command or assembly instruction

Examples:

>HELP Open the help system

>HELP GO Open GO command help information.

5.34 HGO - Begin Program Execution

The HGO command starts execution of code in the debugger at the current program counter (PC) address. Execution continues until a key or mouse is pressed, a breakpoint set with a BR command occurs, or an error occurs. If a key is pressed, real-time execution will stop and the debugger will stop the processor until it reaches the next source instruction.

Syntax:

HGO

Examples:

> HGO Step one source instruction.

5.35 HLOAD - Load ELF/DWARF Object

The HLOAD command allows the user to load an ELF/DWARF, S19, or P&E map file.

Syntax:

HLOAD [<filename>]

Where:

<filename>The name of the object or debug file to be loaded.

Examples:

> HLOAD MAIN.ELF Load the object and debug information in the ELF/DWARF file MAIN.ELF.

5.36 HLOADMAP - Load Source-Level Debug Map

The HLOADMAP command loads a map file that contains source level debug information into the debugger. This command only loads debug info, it does not load object and debug info.

Syntax:

HLOADMAP [<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:

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

5.37 HSTEP - High-Level Language Source Step

The HSTEP command allows the user to step one high-level language source instruction. This is accomplished by rapidly single-stepping the processor on the assembly level.

Syntax:

HSTEP

Examples:

> HSTEP Step one source instruction.

5.38 HSTEPFOR - Step Forever (High-Level Language)

HSTEPFOR 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:

HSTEPFOR

Example:

>HSTEPFOR Step through instructions continuously

5.39 I - Set/Clear I Bit

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

Note:

The CCR bit designators are at the bottom of the CPU window. The CCR pattern is the following:

- U — User/Supervisor mode status/control, CCR bit 15
- IPL — Interrupt Priority Level, CCR bits 10~8
- S — Stop mode enable, CCR bit 7
- X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6
- I — I Interrupt mask, CCR bit 4
- N — Negative status flag, CCR bit 3
- Z — Zero status flag, CCR bit 2
- V — Two's complement overflow status flag, CCR bit 1

C — Carry/borrow status flag, CCR bit 0

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:

I 0|1

Examples:

>I 0 Clear the I bit of the CCR.

>I 1 Set the I bit of the CCR.

5.40 INFO - Display Line Information

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

Syntax:

INFO

Example:

>INFO Display information about the cursor line.

Shows:

Filename: PODTEST.ASMLine number:6

Address: \$0100

Disassembly: START 5F CLRX

5.41 IPL - Set/Clear IPL Bits

The IPL command sets or clears (that is, assigns 0 or 1 to) the IPL bits in the condition code register (CCR). The Interrupt Priority Level [2:0] consists of 3 bits.

Note:

The CCR bit designators are at the bottom of the CPU window. The CCR pattern is the following:

- U — User/Supervisor mode status/control, CCR bit 15
- IPL — Interrupt Priority Level, CCR bits 10~8
- S — Stop mode enable, CCR bit 7
- X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6
- I — I Interrupt mask, CCR bit 4
- N — Negative status flag, CCR bit 3
- Z — Zero status flag, CCR bit 2
- V — Two's complement overflow status flag, CCR bit 1
- C — Carry/borrow status flag, CCR bit 0

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:

IPL <value>

Where:

<value> The new value for the IPL bits.

Examples:

>IPL 7 Set the IPL[2:0] value to 7.

5.42 IX - Set Index Register X

The IX command sets the index register (X) to the specified value. The index register X is 24 bits wide.

Syntax:

IX <value>

Where:

<value> The new value for the IX register.

Example:

>IX 180105 Set the index register value to 180105.

5.43 IY - Set Index Register Y

The IY command sets the index register Y to the specified value. The index register Y is 24 bits wide.

Syntax:

IY <value>

Where:

<value> The new value for the IY register.

Example:

>IY 180106 Set the index register Y value to 180106.

5.44 LF or LOGFILE - Open / Close Log File

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

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

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

The command interpreter does not assume a filename extension.

Syntax:

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

Where:

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

Examples:

- | | |
|----------------|--|
| >LF TEST.LOG R | Start logging. Overwrite file TEST.LOG (in the current directory) with all lines that appear in the status window. |
| >LF TEMP.LOG A | Start logging. Append to file TEMP.LOG (in the current directory) all lines that appear in the status window. |
| >LOGFILE | (If logging is enabled): Disable logging and close the log file. |

5.45 LISTOFF - Do Not Show Info During Steps

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

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

Syntax:

LISTOFF

Example:

>LISTOFF Do not show step information.

5.46 LISTON - Show Info during Steps

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

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

Syntax:

LISTON

Example:

>LISTON Show step information.

5.47 LOAD - Load S19 and MAP

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

Syntax:

LOAD [<filename>]

Where:

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

Examples:

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

5.48 LOAD_BIN - Load Binary File

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

5.49 LOADV_BIN - Execute LOAD_BIN, Then VERIFY

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.

5.50 LOADALL - Execute LOAD, Then LOADMAP

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 P&E Map file.

5.51 LOADDESK - Load Desktop Settings

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

Syntax:

LOADDESK

Example:

>LOADDESK Get window settings from desktop file.

5.52 LOADMAP - Load Map File

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

Syntax:

LOADMAP [<filename>]

Where:

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

Examples:

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

>LOADMAP PROG1 Load map file PROG1.MAP into the host computer.

>LOADMAP A: Displays the names of the .MAP files in drive A:

>LOADMAP Display the names of the .MAP files in the current directory.

5.53 **LOADV - Executes LOAD, Then VERIFY**

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.

5.54 **MACRO or SCRIPT - Execute a Batch File**

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

Note:

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

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

Syntax:

MACRO <filename>

Where:

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

Examples:

>MACRO INIT.MAC Execute commands in file INIT.MAC.

>SCRIPT * Display names of all .MAC files (then execute the selected file).

>MACRO A:* Display names of all .MAC files in drive A (then execute the selected file).

>MACRO Display names of all .MAC files in the current directory, then execute the selected file.

5.55 MACROEND - Stop Saving Commands to File

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

Syntax:

MACROEND

Example:

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

5.56 MACROSTART - Save Debug Commands to File

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

Syntax:

MACROSTART [<filename>]

Where:

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

Example:

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

5.57 MACS - List Macros

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

5.58 MD or SHOW - Display Memory, Window 1

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

Syntax:

MD <address>

Where:

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

Example:

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

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

5.59 MD2 - Display Memory, Window 2

The MD2 command displays the contents of 32 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.

5.60 MDF3 / MDF6 or SHOWF3 / SHOWF6

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

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

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

Syntax:

MDF3 add

Example:

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

5.61 MM or MEM - Modify Memory

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

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

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

Syntax:

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

Where:

<address> The address of the first memory location to be modified.

<n> The value(s) to be stored (optional).

Examples:

With only an address:

>MM 90 Start memory modify at address \$90.

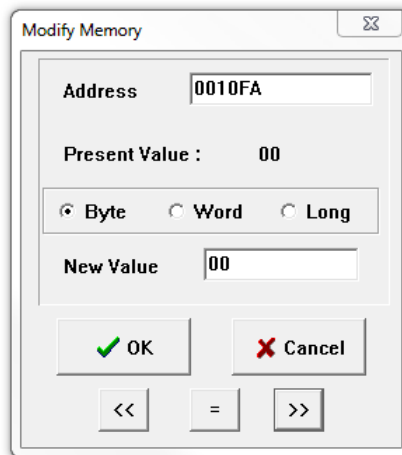


Figure 5-4: Memory Modify Window

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.

5.62 N - Set/Clear N Bit

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

Note:

The CCR bit designators are at the bottom of the CPU window. The CCR pattern is the following:

- U — User/Supervisor mode status/control, CCR bit 15
- IPL — Interrupt Priority Level, CCR bits 10~8
- S — Stop mode enable, CCR bit 7
- X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6
- I — I Interrupt mask, CCR bit 4
- N — Negative status flag, CCR bit 3
- Z — Zero status flag, CCR bit 2
- V — Two's complement overflow status flag, CCR bit 1
- C — Carry/borrow status flag, CCR bit 0

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.

5.62.1 NOBR - Clear All Breakpoints

Clears all break points.

Syntax:

NOBR

Example:

>NOBR Clears all break points.

5.63 PC - Set Program Counter

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

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

Syntax:

PC <address>

Where:

<address> The new PC value.

Example:

>PC 0500 Sets the PC value to 0500.

5.64 PDUMP - Dump Program Memory To Screen

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

Note:

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

Syntax:

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

Where:

<startrange> Beginning address of the program memory block.

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

<n> Optional number of bytes, words, or longs to be written on one line.

Examples:

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

5.65 QUIET - Toggle Refresh Of Memory-Based Windows

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

Syntax:

QUIET

Example:

>QUIET Toggles the current debugger state between quiet and not quiet.

>QUIET OFF Disables Quiet mode (all windows refresh).

>QUIET ON Enables Quiet mode (windows are blank and do not refresh).

5.66 QUIT - Exit Program

Exit the program.

Syntax:

QUIT

Example:

>QUIT Exit the application

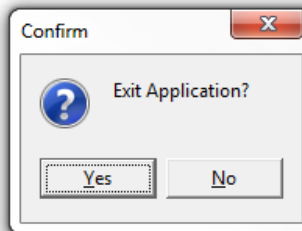


Figure 5-5: Quit Command Dialog

5.67 R or REG or STATUS - Show Registers (If Applicable)

The REG command displays the contents of the CPU registers in the status window (if applicable). 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.

5.68 REM - Place Comment in Macro File

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

Syntax:

REM <text>

Where:

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

Example:

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

5.69 RESET - Reset 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.

5.70 RTVAR - Display Variable Contents In Real Time

The RTVAR command displays the specified address and its contents in the Variables Window for viewing during code execution and while the part is running (real time). 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 RTVAR 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:

RTVAR [.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:

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

5.71 RUNNING - Re-Enter Debugger With CPU 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.

5.72 S - Set/Clear S Bit

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

Note:

The CCR bit designators are at the bottom of the CPU window. The CCR pattern is the following:

- U — User/Supervisor mode status/control, CCR bit 15
- IPL — Interrupt Priority Level, CCR bits 10~8
- S — Stop mode enable, CCR bit 7
- X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6

- I — I Interrupt mask, CCR bit 4
- N — Negative status flag, CCR bit 3
- Z — Zero status flag, CCR bit 2
- V — Two's complement overflow status flag, CCR bit 1
- C — Carry/borrow status flag, CCR bit 0

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:

S 0|1

Examples:

>S 0 Clear the S bit of the CCR.

>S 1 Set the S bit of the CCR.

5.73 SAVEDESK - Save Desktop Settings

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

Syntax:

SAVEDESK

Example:

>SAVEDESK Save window settings for the application.

5.74 SERIAL - Set Up Serial Port Parameters

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

5.75 SERIALON - Turn Communication Window Into Terminal

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
```

5.76 SERIALOFF - Turn Off Serial Port During GO

Turns off serial port use during GO.

Syntax:

```
SERIALOFF
```

Example:

```
>SERIALOFF      Turns off serial port use during GO command
```

5.77 SHOWCODE - Display Code at Address

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

Syntax:

```
SHOWCODE <address>
```

Where:

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

Example:

```
>SHOWCODE 200      Show code starting at hex location 200.
```

5.78 SHOWMAP or MAP - Show Information in Map File

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

Syntax:

SHOWMAP

Example:

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

5.79 SHOWPC - Display Code at PC

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

Syntax:

SHOWPC

Example:

>SHOWPC Show code from the PC address value.

5.80 SHOWTRACE - View Trace Buffer

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.

5.81 **SNAPSHOT - Take Screen Snapshot**

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

Syntax:

SNAPSHOT

Example:

>LOGFILE SNAPSHOT This command will open a file by the name SNAPSHOT.LOG and stores all the command at the status window.

>SNAPSHOT This command will take a snapshot of all the open windows of ICD and store it in SNAPSHOT.LOG file.

>LF This command will close SNAPSHOT.LOG file

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

5.82 **SOURCE - Toggle Source/Disassembled Code**

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.

5.83 SOURCEPATH - Path For Source Code

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

5.84 SP - Stack Pointer

The SP command sets the Stack Pointer to a specified value

Syntax:

SP <n>

Where:

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

Example:

>SP FF Set the Stack Pointer to hex FF.

5.85 SS - Single Step Source-Level Code

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.

5.86 ST or STEP or T - Single Step

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

Syntax:

STEP <n>

or

ST <n>

or

T <n>

Where:

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

Example:

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

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

5.87 STEPFOR - Step Forever

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

Syntax:

STEPFOR

Example:

>STEPFOR Step through instructions continuously.

5.88 STEPTIL - Single Step to Address

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

Syntax:

STEPTIL <address>

Where:

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

Example:

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

5.89 SYMBOL - Add Symbol

The SYMBOL command creates a new symbol, which can be used anywhere in the debugger, in place of the symbol value. If this command is entered with no parameters, it

will list the current user defined symbols. If parameters are specified, the SYMBOL command will create a new symbol.

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

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

Syntax:

SYMBOL [<label> <value>]

Where:

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

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

Examples:

>SYMBOL Show the current user-defined symbols.

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

5.90 TRACE - Capture Trace Data

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 U - Set/Clear U Bit

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

Note:

The CCR bit designators are at the bottom of the CPU window. The CCR pattern is the following:

U — User/Supervisor mode status/control, CCR bit 15

IPL — Interrupt Priority Level, CCR bits 10~8

S — Stop mode enable, CCR bit 7

X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6

I — I Interrupt mask, CCR bit 4

N — Negative status flag, CCR bit 3

Z — Zero status flag, CCR bit 2

V — Two's complement overflow status flag, CCR bit 1

C — Carry/borrow status flag, CCR bit 0

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:

U 0|1

Examples:

>U 0 Clear the U bit of the CCR.

>U 1 Set the U bit of the CCR.

5.92 UPLOAD_SREC - Upload S-Record to Screen

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

Note:

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

Syntax:

UPLOAD_SREC <startrange> <endrange>

Where:

<starange> 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.

5.93 V - Set/Clear V Bit

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

Note:

The CCR bit designators are at the bottom of the CPU window. The CCR pattern is the following:

U — User/Supervisor mode status/control, CCR bit 15

IPL — Interrupt Priority Level, CCR bits 10~8

S — Stop mode enable, CCR bit 7

X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6

I — I Interrupt mask, CCR bit 4

N — Negative status flag, CCR bit 3

Z — Zero status flag, CCR bit 2

V — Two's complement overflow status flag, CCR bit 1

C — Carry/borrow status flag, CCR bit 0

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.

5.94 VAR - Display Variable

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

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

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

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

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

Syntax:

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

Where:

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

Examples:

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

5.95 **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.

5.96 **VERIFY - Compare Program Memory With S-Record**

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"

5.97 WHEREIS - Display Symbol Value

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

Syntax:

WHEREIS <symbol> | <address>

Where:

<symbol> A symbol listed in the symbol table.

<address> Address for which a symbol is defined.

Examples:

>WHEREIS START Display the symbol START and its value.

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

5.98 X - Set/Clear X Bit

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 bottom of the CPU window. The CCR pattern is the following:

U — User/Supervisor mode status/control, CCR bit 15

IPL — Interrupt Priority Level, CCR bits 10~8

S — Stop mode enable, CCR bit 7

X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6

I — I Interrupt mask, CCR bit 4

N — Negative status flag, CCR bit 3

Z — Zero status flag, CCR bit 2

V — Two's complement overflow status flag, CCR bit 1

C — Carry/borrow status flag, CCR bit 0

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.

5.99 Z - Set/Clear Z Bit

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

Note:

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

U — User/Supervisor mode status/control, CCR bit 15

IPL — Interrupt Priority Level, CCR bits 10~8

S — Stop mode enable, CCR bit 7

X — X Interrupt mask (pseudo non-maskable interrupt), CCR bit 6

I — I Interrupt mask, CCR bit 4

N — Negative status flag, CCR bit 3

Z — Zero status flag, CCR bit 2

V — Two's complement overflow status flag, CCR bit 1

C — Carry/borrow status flag, CCR bit 0

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.

