ICDRS08 In-Circuit Debugger for RS08

Contents:

Plug-Ins:

ICD Software License

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. The software being installed 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 without the express written permission of P&E Microcomputer Systems. Under no conditions may you remove the copyright notices from this software or documentation.

This software requires the use of a license code to operate.

If you have purchased an ICD software license from P&E Microcomputer Systems and been issued a hardware-based license code (a license code that begins with V2), you may (1) install the provided hardware-based ICD license code into a single Cyclone or Multilink unit and (2) install this software on any computer with which the specific Multilink or Cyclone will be used. This gives you the ability to run this software on multiple computers, used by multiple users, with the Multilink or Cyclone hardware which has the hardware license code installed.

If you have purchased an ICD software license from P&E Microcomputer Systems and been issued a legacy computer based license code (a license code that begins with V1), this software is licensed as a single user license which means: (1) This software may be used by one individual user on up to two different computers, provided that the software is never used on the two computers at the same time, (2) P&E Microcomputer Systems 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 Microcomputer Systems for volume discounts and site licensing agreements.

P&E Microcomputer Systems does not assume any liability for the use of this software beyond the original purchase price of the software. 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.

$ K Command Line Parameters

To setup ICDRS08 to run with certain command line parameters, highlight the ICDRS08 icon and select **PROPERTIES** from the Program Manager File Menu.

Syntax:

ICDRS08 [option] ... [option]

[option]	Optional parameters are as follows:
lpt1…lpt3	Chooses lpt1, lpt2, or lpt3. The software will remember the last setting used.
pci1…pci6	Chooses which PCI card to communicate with. The software will remember the last setting used.
pci_delay n	Sets speed of PCI card shift clock, where **n = 0…255**. The equation for the PCI card shift clock frequency is 33 * 10^6 / (5 + 2**n**).
running	Starts ICD with CPU running (see Running help)
io_delay_cnt n	Sets the background debug transfer parameter n where **0<=n<=$7fff**. If you start ICD08 without giving an io_delay_cnt, the software tries to find an appropriate parameter n and shows you what was selected for xtal frequencies less than approximately 1 MHz.
reset_delay n	Causes a delay of '**n**' milliseconds **after** the software pulses the reset line, and **before** the software checks the processor status to make sure that background mode has been entered. Used when reset pulse on the reset line is extended, for example by using a reset driver, which may add several hundred milliseconds to reset.
quiet	Starts the ICD without filling the memory windows and the disassembly window. Can be used for speed reasons or to avoid DSACK errors on startup until windows are positioned or chip selects enabled.
	-or-
path	A DOS path to the directory containing the source code for source level debug or a DOS path to a source file to be loaded at startup (path part is also saved).

Command_Line_Parameters

$ Command Line Parameters

K com;command line;nodtr;parameters;path;quiet;ram_start;running;STARTUP

Note:

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

Examples:

ICDRS08 lpt2	Chooses lpt2, Causes the background debug mode clock to be extended by 2 Cycles.

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

[#] [$] [K] User Interface

- Status Window
- Code Window
- Variables Window
- Memory Window
- Colors Window
- CPU Window

[#] User_Interface

[$] User Interface

[K] User Interface;Windows

[#] [$] [K] Nomenclature

Note the following:

n Any number from 0 to 0FFFFFFFF (hex). The default base is hex. To enter numbers in another base use the suffixes 'T' for base ten, 'O' for base eight or 'Q' for base two. You may also use the prefixes '!' for base ten, '@' for base 8 and '%' for base two. Numbers must start with either one of these prefixes or a numeric character.

Example: **0FF = 255T = 377O = 11111111Q = !255 = @377 = %11111111**

add Any valid address (**default hex**).

[ ] Optional parameter.

PC Program Counter points to the next instruction 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.

[#] NOMENCLATURE

[$]NOMENCLATURE

[K]NOMENCLATURE

$ K Commands

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

# A	Set accumulator A.
ASM [add]	Assemble Instructions to memory.
ASCIIF3 or ASCIIF6	Toggles the memory windows.

B

BELL	Sound Bell
# BF	Block Fill. Same as FILL command.
BR	Set break point.

C

# C	Set/Clear C bit.
# CAPTURE	Open a capture file named '**filename**'. Same as the LOGFILE command.
CAPTUREOFF	Turn off capturing and close the current capture file.
CCR	Set Condition Code Register.
CLEARMAP	Clear Away Map File.

Commands
$Commands
KCommands
A
B
BF
C
CAPTURE

CLEARSYMBOL	Clear Away Symbols.
CLEARVAR	Remove all the variables from the variables window.
CODE	Show disassembled code in the code window starting at address **add**. If you specify an address in the middle of an intended instruction, improper results may occur.
COLORS	Change Debugger Colors

D

# **DASM**	Disassemble Instructions
DUMP	Dump Memory to the Status Window

E

# **EVAL**	Evaluates a numerical term or simple expression, giving the result in hexadecimal, decimal, octal, and binary formats.
EXIT	Terminates the software and closes all windows.

F

# _ # **FILL**	Block Fill. Same as **BF** command.

G

# _ # **G or GO**	Begin Program Execution
# **GOEXIT**	Similar to **GO** command except that the target is left running without any breakpoints and the debugger software is terminated.

D
E
F
FILL
G
GO
GOEXIT

GONEXT	Go from the current PC until the next instruction is reached. Used to execute past a subroutine call or past intervening interrupts.
[#] **GOTIL**	Execute Program until Address

H

[#]**HELP**	Display Help Information
HLOAD	Load ELF/DWARF source files
HLOADMAP	Load DWARF debugging information
HSTEP	Single-step one high-level source line
HSTEPFOR	Continually step high-level instructions
HGO	Start full-speed execution
HREG	Set H Register
HX	Sets both bits of concatenated H:X index register.

I

[#]**INFO**	Displays information about the line that is highlighted in the source window
IRQ or INT	Displays value of IRQ pin.

L

[#] [#] **LF or LOGFILE**	Open/Close Log File. Same as **CAPTURE** command.
LOAD	Load S19 File.

[#] GOTIL

[#] H

[#] I

[#] L

[#] A7UQQB7

# LOADALL	Execute a LOAD and a LOADMAP command.
# LOADDESK	Loads the desktop settings that set the window positions, size, and visibility.
# LOADMAP	Load Debug Map.
LOADV	Execute the LOAD command and then automatically do a VERIFY command with the same file.
LOAD_BIN	Load a binary file of bytes starting at address **add**. The default filename extension is **.BIN**.
LOADV_BIN	Execute the LOAD_BIN command and then do a VERIFY using the same file.

M

# _ # MACRO	Execute Script File
# MACROEND	Stop Recording Script File.
MACROSTART	Start Recording Script File.
MACS	Bring up a window with a list of macros.
MD or MD1	Displays (in the first memory window) the contents of memory locations, beginning at the specified address. Identical to MD1 command.
MD2	Displays (in the second memory window) the contents of memory locations, beginning at the specified address.
# MM or MEM	Memory Modify.

CVITGIS
ATQSFRT
UH255E
M
MACRO
AMT3Z_M
MM

N

#**NOBR**	Clear all break points.

P

# **PC**	Assigns the specified value to the program counter.

Q

# **QUIET**	Turn off (on) refresh of memory based windows.
QUIT	Exit the program.

R

# # **R**	Start Register Viewer/Editor.
# **REG**	Display CPU Registers to Status Window (same as **STATUS** command).
REM	Add Comments to Script File.
# **RESET**	Reset Emulation MCU.
RTVAR	Show address in Variable Window.
RUN	Begin Program Execution

S

# **SAVEDESK**	Saves the desktop settings for the application when it is first opened or for use with the **LOADDESK** command.

N
P
Q
R
HVSXMT
REG
RESET
S

SERIAL	Set up parameters for serial port.
SHOWCODE	Show disassembled code in the code window starting at address **add**. If you specify an address in the middle of an intended instruction, improper results may occur.
SHOWMAP	Enables the user to view information from the current **MAP** file stored in the memory.
SHOWPC	Displays code in the code window starting from the address in the program counter.
SNAPSHOT	Take a snapshot (black and white) of the current screen and send it to the capture file if one exists. Can be used for test documentation and system testing.
SOURCEPATH	Either uses the specified filename or prompts the user for the path to search for source code that is not present in the current directory.
SP	Sets Stack Pointer
STATUS	Display CPU Registers to Status Window (same as **REG** command).
SS	Does one step of source level code. Source must be showing in the code window.
# **STEP or ST or T**	Single Step (Trace).
STEPFOR	Step forever.
STEPTIL	Repeatedly single step until the instruction pointer is equal to the given address.
# **SYMBOL**	Add the given label to a temporary symbol table with the value given. If a map file has been loaded, the values and symbols in that file take precedence. Specifying symbol with no parameters lists the temporary symbols in the debug window.

T

# **T [n]**	Same as **ST**.

STEP

SYMBOL

T

U

# **UPLOAD_SREC**	Uploads the memory locations between the first and second addresses (add) to the screen in the form of S records.

V

#**VAR**	Displays the specified address and its contents in the variables window for viewing during code execution.
VER	Display the version number of the ICD software. Same as **VERSION**.
VERIFY	Compare the contents of program memory with an S-record file. You will be prompted for the name of the file. The comparisons will stop at the first location with a different value.
VERSION	Display the version number of the ICD software. Same as **VER**.

W

# **WHEREIS**	Give the value for the given symbol, either temporary or MAP file based. See **SYMBOL**.

X

# **X or XREG**	Sets the index register **(X)** to the specified value.

Z

# **Z**	Set/clear Z bit.

U
V
W
X
Z

PEmicro

Technical Support and Customer Service:

Web: Please use our web-based support request system at
http://www.pemicro.com/SRS/main_screen_user.cfm

Email: technical-info@pemicro.com
sales-info@pemicro.com

Phone: 617-923-0053

Fax: 617-923-0808

98 Galen St., 2nd Floor
Watertown, MA 02472

[#] Technical_Support

[$] Technical Support

[K] Contact Information;Sales;Technical Support

[#] [$] [K] Short History Of PEmicro

PEmicro was founded in **1980** and incorporated in **1982** by **Dr. David A. Perreault**. **PEmicro** developed and marketed the first general purpose **EPROM** programmer. It was based on the **Z80** microprocessor and provided the ability to reconfigure pin-outs and programming algorithms to accommodate new and special devices.

The second product developed by **PEmicro** was an **S100** bus based **ROM** emulator. This product pushed forward the capability to rapidly develop microcomputer based systems.

Having developed a large number of microcontroller/microprocessor systems including both hardware and software, **PEmicro** capitalized on this experience and began developing software development tools. Much of this effort was directed toward simulators and cross assemblers for microcontrollers from Motorola, Texas Instruments, Intel, and National Semiconductor.

PEmicro developed the concept of an in-circuit simulator. This device combines the best features of a simulator with the added advantage of real IO. This concept was used in Motorola's very successful promotion for the **68HC05K1** part called the **KICS05**.

More recently, **PEmicro** has developed in circuit emulators and in line **EPROM/EEPROM** programmers for Motorola's **68HC16** and **68HC3xx** series of processors. These products were the first to use the background debug mode on these processors to provide full in-circuit emulation. **PEmicro** worked with Motorola to develop their product introduction kit, the **68HC16EVB**, which Motorola used to introduce this new product to the computer marketplace.

[#] P_E_History

[$] P&E History

[K] P&E History

$ K Command Recall

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

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

[#] Command_Recall

[$] Command Recall

[K] Command Recall

$ K Memory Access

1. When you modify bytes, words, or longs they are read/written using the corresponding background debug mode read/write.

2. Memory window displays are read using word reads.

3. The VAR window is read using the appropriate reads.

4. Code window data is read using word reads.

[#] Memory_Access

[$] Memory Access

[K] Memory Access

$ K IRQ or INT Command

The **IRQ** command displays the state value of the MCU IRQ pin. Same as the **INT** command.

Syntax:

IRQ

Examples:

>**IRQ** Displays value of the IRQ pin.

IRQ_or_INT_Command

$IRQ or INT Command

KINT;IRQ

$ K Using The Register Interpreter

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

Register Interpreter Display

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

{bmc reg_blk.bmp}

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

{bmc reg_sel.bmp}

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

{bmc reg_view.bmp}

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

{bmc regfldop.bmp}

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

{bmc regmodcf.bmp}

Adding Register Field Descriptions To VAR Window

Using_The_Register_Interpreter

$ Using The Register Interpreter

K R command

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

{bmc addregds.bmp}

[#] [$] [K] 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.

[#] G_or_GO_or_RUN_Commands

[$] G or GO or RUN Commands

[K] G;GO;RUN

[#] [$] [K] BR Command

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

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.

[#] BR_Command

[$] BR Command

[K] BR;Breakpoint - Clear;Breakpoint - Set

[#] [$] [K] 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.

[#] LOAD_Command

[$] LOAD Command

[K] LOAD;Map File - Load;S19 - Load

[#] [$] [K] 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.

[#] GOTIL_Command

[$] GOTIL Command

[K] GOTIL

[#] [$] [K] 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.

[#] ST_or_STEP_or_T_Commands
[$] ST or STEP or T Commands
[K] Single Step;ST;STEP;T

[#] [$] [K] BLOCK FILL or BF

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

Syntax:

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

Where:

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

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

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

Examples:

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

[#] BLOCK_FILL_Command

[$] BLOCK FILL Command

[K] BF;BF.B;BF.L;BF.W;BLOCK FILL;FILL

[#] [$] [K] TRACE

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

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

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

Syntax:

TRACE <[add1] [add2]>

Where:

add1 Starting address
add2 Ending address

Example:

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

[#] TRACE_Command
[$] TRACE Command
[K] TRACE

[#] [$] SHOWTRACE

After executing the **TRACE** command, which monitors the execution of the CPU and logs the address of (up to) the last 256 instructions that have been executed into an internal array, the **SHOWTRACE** command (or **F7**) allows the user to view this trace buffer.

Syntax:

SHOWTRACE

Example:

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

[#] SHOWTRACE_Command

[$] SHOWTRACE Command

[#] [$] SOURCE

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

Syntax:

SOURCE

Example:

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

[#] SOURCE_Command

[$] SOURCE Command

[#] [$] [K] 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.

[#] STEPTIL_Command

[$] STEPTIL Command

[K] Single Step To Address;STEPTIL

[#] [$] [K] RESET - Reset Emulation MCU

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

Syntax:

RESET

Example:

>RESET Reset the MCU.

[#] RESET_Command

[$] RESET Command

[K] RESET

[#] [$] [K] LPTx

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

Syntax:

LPTx

Where:

x	1, 2 or 3

Example:

LPT 2	Specifies that the debugger should use parallel port 2

[#] LPTx_Command

[$] LPTx Command

[K] LPT1;LPT2;LPT3;LPTx

[#] [$] [K] WATCHDOG

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

Syntax:

WATCHDOG

Example:

>WATCHDOG

[#] WATCHDOG_Command

[$] WATCHDOG Command

[K] WATCHDOG

[#] [$] [K] 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.

[#] STEPFOR_Command

[$] STEPFOR Command

[K] STEPFOR

$ K VERIFY

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

Syntax:

VERIFY

Example:

>LOADALL test.s19

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

[#] VERIFY_Command

[$] VERIFY Command

[K] VERIFY

[#][$][K] NOBR

Clears all break points.

Syntax:

NOBR

Example:

>NOBR Clears all break points.

[#] NOBR_Command

[$] NOBR Command

[K] Breakpoints - Clear All;NOBR

TIME

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

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

Syntax:

TIME <[add1] [add2]>

Where:

add1 Starting address
add2 Ending address

Example:

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

[#] TIME_Command
[$] TIME Command
[K] TIME

[#] [$] [K] LOADALL

Does a **LOAD** and a **LOADMAP** command.

Syntax:

LOADALL [filename]

Where:

filename	Filename of your source code

Example:

LOADALL myprog	This command will load the S19 object file and the P&E Map file.

[#] LOADALL_Command

[$] LOADALL Command

[K] LOADALL

COUNT

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

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

Syntax:

COUNT [add1] [add2]

Where:

add1 Go from first address.

add2 Set breakpoint at second address.

Example:

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

[#] COUNT_Command

[$] COUNT Command

[K] COUNT

[#] [$] [K] COUNTER

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

Syntax:

COUNTER [add]

Where:

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

Example:

>COUNTER 100 Add (or remove) a counter at hex location 100.

>COUNTER Shows all the current internal counters.

[#] COUNTER_Command

[$] COUNTER Command

[K] COUNTER

[#] [$] [K] MACS

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

Syntax:

MACS

Example:

>MACS Brings up a list of MACROS

[#] MACS_Command

[$] MACS Command

[K] MACS

SERIAL

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

Syntax:

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

Where:

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

Example:

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

---

[#] SERIAL_Command

[$] SERIAL Command

[K] SERIAL

[#] [$] [K] SERIALON

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

Syntax:

SERIALON

Example:

```
>SERIAL 2 9600 N 8 1
>SERIALON
>GO
```

[#] SERIALON_Command

[$] SERIALON Command

[K] SERIALON

[#] [$] [K] SERIALOFF

Turns off serial port use during **GO**.

Syntax:

SERIALOFF

Example:

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

[#] SERIALOFF_Command

[$] SERIALOFF Command

[K] SERIALOFF

$ K 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.

[#] RUNNING_Parameter

[$] RUNNING Parameter

[K] running

[#] [$] [K] GOEXIT

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

Syntax:

GOEXIT [add]

Where:

add Starting address of your code.

Example:

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

[#] GOEXIT_Command

[$] GOEXIT Command

[K] GOEXIT

[#] [$] [K] MD - Memory Display

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

Syntax:

MD <address>

Where:

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

Example:

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

[#] Memory_Display

[$] Memory Display

[K] MD;Memory Display

[#] [$] [K] MD2 Command

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

Syntax:

MD2 <address>

Where:

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

Example:

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

[#] MD2_Command

[$] MD2 Command

[K] MD2 Command

[#] [$] [K] LOAD_BIN

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

Syntax:

LOAD_BIN [filename] [add]

Where:

filename	Name of the binary file
add	Starting address

Example:

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

[#] LOAD_BIN_Command

[$] LOAD_BIN Command

[K] Binary File - Load;LOAD_BIN

[#] [$] [K] LOADV_BIN

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

Syntax:

LOADV_BIN [filename] [add]

Where:

filename	Name of the binary file
add	Starting address

Example:

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

[#] LOADV_BIN_Command

[$] LOADV_BIN Command

[K] Binary File - Load and Verify;LOADV_BIN

[#] [$] [K] REG or STATUS - Show Registers

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

Syntax:

REG

Example:

>REG Displays the contents of the CPU registers.

[#] REG_or_STATUS_Commands

[$] REG or STATUS Commands

[K] REG;Show Registers;STATUS

[#] [$] [K] BGND_TIME

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

Syntax:

BGND_TIME [n]

Where:

n number of points logged

Example:

>BGND_TIME 4

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

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

[#] BGND_TIME_Command

[$] BGND_TIME Command

[K] BGND_TIME

QUIET

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

Syntax:

QUIET

Example:

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

QUIET_Command

$QUIET Command

Kquiet;Refresh of Memory-Based Windows - Toggle

[#][$][K] DUMP_TRACE

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

Syntax:

Dump_Trace

Example:

>Dump_Trace

[#] DUMP_TRACE_Command

[$] DUMP_TRACE Command

[K] DUMP_TRACE;Trace Buffer

[#][$][K] SS

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

Syntax:

SS

Example:

>SS Does one step of source level code.

[#] SS_Command

[$] SS Command

[K] Single Step;SS

GONEXT

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

Syntax:

GONEXT

Example:

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

[#] GONEXT_Command

[$] GONEXT Command

[K] GONEXT

[#] [$] [K] 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.

[#] VERSION_or_VER_Commands

[$] VERSION or VER Commands

[K] VER;VERSION

[#] [$] [K] CAPTURE

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

Syntax:

CAPTURE <filename>

Where:

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

Example:

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

[#] CAPTURE_Command

[$] CAPTURE Command

[K] CAPTURE

[#] [$] [K] CAPTUREOFF

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

Syntax:

CAPTUREOFF

Example:

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

[#] CAPTUREOFF_Command

[$] CAPTUREOFF Command

[K] CAPTUREOFF

[#] [$] [K] ASCIIF3 and ASCIIF6

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

ASCIIF3 toggles memory window 1.

ASCIIF6 toggles memory window 2.

Syntax:

ASCIIF3

Example:

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

[#] ASCIIF3_and_ASCIIF6_Commands

[$] ASCIIF3 and ASCIIF6 Commands

[K] ASCIIF3;ASCIIF6;Memory Windows - ASCII Toggle

[#] [$] 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.

[#] MDF3_and_MDF6

[$] MDF3 and MDF6

[#] [$] [K] PC - Program Counter

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

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

Syntax:

PC <address>

Where:

<address>	The new PC value.

Example:

>**PC 0500**	Sets the PC value to 0500.

[#] Program_Counter___Link

[$] Program Counter - Link

[K] PC;Program Counter

$ K 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.

{bml lcd00004.BMP}

POPUP MENU

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

Help...

Displays this help topic.

KEYSTROKES

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

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

STATUS_WINDOW

$STATUS_WINDOW

KSTATUS_WINDOW

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

[#] [$] [K] 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_Base_Address

[$] Code Window Base Address

[K] Code Window Base Address

[#] [$] [K] Code Window

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

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

- **Using Code Window Quick Execution Features**
- **Using Code Window Popup Debug Evaluation Hints**

GOTIL and BREAKPOINT Shortcuts

You may execute a GOTIL to a particular line of code by double-clicking the blue arrow to the left of that line. You may also set Breakpoint at a particular line of code by double-clicking the red dot to the left of that line.

{bmc cwind08.bmp}

Code Window: Disassembly

{bmc cwins08.bmp}

Code Window: Source

POPUP MENU

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

Toggle Breakpoint at Cursor

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

[#] CODE_WINDOW

[$] CODE WINDOW

[K] Code_Window

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.

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

$ K 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:

{bml varwin1.bmp}

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.

- **Adding And Deleting Variables**
- **Modifying A Variable's Value**
- **Modifying A Variable's Properties**

[#] Variables_Window

[$] Variables_Window

[K] Variables_Window

[#] [$] [K] 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.

{bml lcd00007.BMP}

POPUP MENU

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

Set Base Address

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

Show Memory and ASCII

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

Show Memory Only

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

Help...

Shows this help topic.

KEYSTROKES

[#] MEMORY_WINDOW

[$] MEMORY WINDOW

[K] Memory_Window

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

[#] [$] [K] Colors Window

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

{bmc Icd00008.BMP}

[#] COLORS_WINDOW

[$] COLORS WINDOW

[K] Colors_Window

$ K 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:

{bmc varaddel.bmp}

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.

Adding_And_Deleting_Variables

$ Adding And Deleting Variables

K Add_Variable

[#] [$] [K] 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.

[#] BELL_Command

[$] BELL Command

[K] BELL Command

$ K 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.

[#] CLEARMAP_Command

[$]CLEARMAP Command

[K]CLEARMAP Command

[#] [$] [K] CLEARVAR

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

Syntax:

CLEARVAR

Example:

CLEARVAR Removes all the variables for the variables window.

[#] CLEARVAR_Command

[$] CLEARVAR Command

[K] CLEARVAR Command

[#] [$] [K] ASM - Assemble Instructions

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

Syntax:

ASM [<address>]

Where:

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

Examples:

With an address argument:

>**ASM 100**

The following window appears:

{bmc Icd00010.BMP}

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

---

[#] ASM_Command

[$] ASM Command

[K] ASM Command

{bmc Icd00011.BMP}

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

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.

{bmc lcd00012.BMP}

[#] MM_or_MEM_Command

[$] MM or MEM Command

[K] MM or MEM Command

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**.

[#] [$] [K] 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.

[#] EXIT_or_QUIT_Command

[$] EXIT or QUIT Command

[K] EXIT or QUIT Command

[#] [$] [K] 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.

[#] CLEARSYMBOL_Command

[$] CLEARSYMBOL Command

[K] CLEARSYMBOL Command

[#] [$] [K] 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.

[#] LOGFILE_Command

[$] LOGFILE Command

[K] LOGFILE Command

[#] [$] [K] 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_Command

[$] MACRO Command

[K] MACRO Command

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

[#] [$] [K] 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**

[#] MACROSTART_Command

[$] MACROSTART Command

[K] MACROSTART Command

[#] [$] [K] 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.

[#] DASM_Command

[$] DASM Command

[K] DASM Command

n The optional parameter **n** determines the number of Bytes, Words, or Longwords which are written on one line.

Examples:

>**DASM 300**

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

>**DASM 400 408**

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

[#] [$] [K] 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.

[#] LOADDESK_Command

[$] LOADDESK Command

[K] LOADDESK Command

[#] [$] [K] LOADMAP - Load Map File

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

Syntax:

LOADMAP [<filename>]

Where:

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

Examples:

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

[#] LOADMAP_Command

[$] LOADMAP Command

[K] LOADMAP Command

[#] [$] [K] 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.

[#] LISTON_Command

[$] LISTON Command

[K] LISTON Command

[#] [$] [K] 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.

[#] DUMP_Command

[$] DUMP Command

[K] DUMP Command

[#] [$] 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.

[#] PDUMP_Command

[$] PDUMP Command

[#] [$] [K] 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.

[#] LISTOFF_Command

[$] LISTOFF Command

[K] LISTOFF Command

[#] [$] [K] 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"

[#] EVAL_Command

[$] EVAL Command

[K] EVAL Command

[#] [$] [K] 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.

[#] HELP_Command
[$]HELP Command
[K]HELP Command

[#] [$] [K] 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

[#] INFO_Command
[$] INFO Command
[K] INFO Command

[#] [$] [K] MD or SHOW - Display Memory at Address

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

The **MD** and **SHOW** commands are identical.

Syntax:

MD <address>

Where:

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

Examples:

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

[#] MD_Command

[$] MD Command

[K] MD Command

[#] [$] [K] 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.

[#] REM_Command

[$] REM Command

[K] REM Command

[#] [$] [K] 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.

[#] MACROEND_Command

[$] MACROEND Command

[K] MACROEND Command

[#] [$] [K] 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.

[#] SAVEDESK_Command

[$] SAVEDESK Command

[K] SAVEDESK Command

[#] [$] [K] 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.

[#] WHEREIS_Command

[$] WHEREIS Command

[K] WHEREIS Command

[#] [$] [K] UPLOAD_SREC - Upload S-Record to Screen

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

Note:

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

Syntax:

UPLOAD_SREC <startrange> <endrange>

Where:

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

Example:

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

[#] UPLOAD_SREC_Command

[$] UPLOAD_SREC Command

[K] UPLOAD_SREC Command

[#] [$] D_UPLOAD_SREC

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

Note:

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

Syntax:

D_UPLOAD_SREC <startrange> <endrange>

Where:

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

Example:

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

[#] D_UPLOAD_SREC

[$] D_UPLOAD_SREC

[#] [$] [K] 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**'.

[#] SYMBOL_Command

[$] SYMBOL Command

[K] SYMBOL Command

[#] [$] [K] 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 defualt)
* Word (**.W**): hexadecimal
* Long (**.L**): hexadecimal
* String (**.S**): ASCII characters

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

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

Syntax:

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

Where:

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

Examples:

>**VAR C0**	Show byte value of address **C0** (hex and binary)
>**VAR.B D4**	Show byte value of address **D4** (hex and binary)

[#] VAR_Command

[$] VAR Command

[K] VAR Command

>**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**.

[#] [$] [K] 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.**

[#] SHOWCODE_Command

[$] SHOWCODE Command

[K] SHOWCODE Command

[#] [$] [K] 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.

[#] SHOWMAP_Command

[$] SHOWMAP Command

[K] SHOWMAP Command

[#] [$] [K] COLORS - Set Colors of Simulator

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

Syntax:

COLORS

Example:

>**COLORS** Open the colors window.

[#] COLORS_Command

[$] COLORS Command

[K] COLORS Command

[#] [$] [K] 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.

[#] SHOWPC_Command

[$] SHOWPC Command

[K] SHOWPC Command

[#] [$] 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**.

[#] CODE_Command

[$] CODE Command

[#] [$] [K] GOTILROM

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

Syntax:

GOTILROM [add]

Where:

add Starting address of your code.

Example:

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

[#] GOTILROM_Command

[$] GOTILROM Command

[K] GOTILROM Command

LOADV

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

Syntax:

LOADV [filename]

Where:

filename	Filename of your source code

Example:

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

[#] LOADV_Command

[$] LOADV Command

[K] LOADV Command

[#][$][K] QUIT

Exit the program.

Syntax:

QUIT

Example:

>QUIT Exit the application

{bmc Icd00015.BMP}

[#] QUIT_Command

[$] QUIT Command

[K] QUIT Command

[#] [$] [K] 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.

[#] SNAPSHOT_Command

[$] SNAPSHOT Command

[K] SNAPSHOT Command

[#] [$] [K] SOURCEPATH

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

Syntax:

SOURCEPATH filename

Where:

filename Name of the source file

Example:

>SOURCEPATH d:\mysource\myfile.asm

[#] Sourcepath_Command

[$] Sourcepath Command

[K] Sourcepath Command

$ K PC - Program Counter

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

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

Syntax:

PC <address>

Where:

<address>	The new PC value.

Example:

>**PC 0500**	Sets the PC value to 0500.

[#] PC___Program_Counter

[$] PC - Program Counter

[K] PC;Program Counter

[#][$][K] 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.

[#] Stack_Pointer

[$] Stack Pointer

[K] Stack Pointer

[#] [$] [K] 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 ZC (**Z** is zero and **C** is carry). A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

Z 0|1

Examples:

>**Z 0** Clear the Z bit of the CCR.

>**Z 1** Set the Z bit of the CCR.

[#] Z_Command

[$] Z Command

[K] Z

[#] [$] [K] A or ACC - Set Accumulator Value

The **ACC** command sets the accumulator to a specified value. The ACC and A commands are identical.

Syntax:

ACC <n>

Where:

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

Example:

>A 10 Set the accumulator to $10.

[#] A_Command

[$] A Command

[K] A

[#] [$] [K] HX - Set H:X Index Register Pair

The **HX** command sets both bytes of the concatenated index register (H:X) to the specified value.

Syntax:

HX <value>

Where:

<value>	The new value for the X register.

Example:

>**HX 0400**	Set the H:X index register value to $0400.

[#] HX_Command

[$] HX Command

[K] H:X Index Register;HX

[#] [$] [K] HREG - Set H Register

The **HREG** command sets the high byte of the H:X index register to a specified value.

Syntax:

HREG <value>

Where:

<value>	The new value for the X register.

Example:

>**HREG 05**	Set the H index register value to 05.

[#] HREG_Command

[$] HREG Command

[K] H Index Register;HREG

[#] [$] [K] X or XREG - Set X Index Register

The **X** command sets the index register (X) to the specified value. The **X** command is identical to the XREG command.

Syntax:

X <value>

Where:

<value>	The new value for the X register.

Example:

>X 05	Set the index register value to 05.
>XREG F0	Set the index register value to F0.

[#] X_Command

[$] X Command

[K] X

[#] [$] [K] 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.

[#] PC_Command

[$] PC Command

[K] PC

[#] [$] [K] 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 ZC (**Z** is zero and **C** is carry). A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

CCR <n>

Where:

<n>	The new hexadecimal value for the CCR.

Example:

>CCR 01	Assign the value 01 to the CCR. This makes the binary pattern 00000001; the C bit set, other bits clear.

[#] CCR_Command

[$] CCR Command

[K] CCR

[#] [$] [K] 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 V11HINZC (**V** is overflow, **H** is half-carry, **I** is IRQ interrupt mask, **N** is negative, **Z** is zero and **C** is carry). A letter in these designators means that the corresponding bit of the CCR is set; a period means that the corresponding bit is clear.

Syntax:

C 0|1

Examples:

>C 0	Clears the C bit of the CCR.
>C 1	Sets the C bit of the CCR.

[#] C_Command

[$] C Command

[K] C

[#] [$] [K] CPU Window

The CPU Window displays the current state of the RS08 CPU registers. The popup window allows modification of these value.

{bmc lcd00016.BMP}

POPUP MENU

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

Set Accumulator

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

Set X Index Register

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

Set Shadow Program Counter

This option is disabled and is only shown for convention.

Set PC

Sets the Program Counter (PC) 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.

{bmc lcd00017.BMP}

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

[#] CPU_Window

[$] CPU Window

[K] CPU Window

$ K 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.

{bmc lcd00018.BMP}

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:

RS08 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.

[#] Stack_Window

[$] Stack Window

[K] Stack Window

$ + K RTVAR Command

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_Command

$ RTVAR Command

+ ICD08Z:0

K RTVAR Command

>**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**.

$ + K @ 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.

{bmc var_mod.bmp}

MODIFYING_A_VARIABLE_S_VALUE

$ MODIFYING A VARIABLE'S VALUE

+ ICD08Z:0

K MODIFYING A VARIABLE S VALUE

@ Status|0|||0||||||

$ + K @ 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.

{bmc var_modp.bmp}

MODIFYING_A_VARIABLE_S_PROPERTIES

$ MODIFYING A VARIABLE'S PROPERTIES

+ ICD08Z:0

K MODIFYING A VARIABLE S PROPERTIES

@ Status|0|||0||||||

$ + K @ 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:

{bmc CODw_dot.bmp}

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 {bmc HGO_BTTN.bmp} on the debugger button bar, execution will begin in real-time. If the debugger encounters a breakpoint, execution will stop on this source line. If a breakpoint is not encountered, execution will continue until the user presses a key or uses the stop button on the debugger button bar. To remove a breakpoint, double-click the large red stop sign.

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

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

{bmc CODwdot2.bmp}

Using_Code_Window_Quick_Execution_Features

$ Using Code Window Quick Execution Features

+ ICD08Z:0

K Using Code Window Quick Execution Features

@ Status|0|||0||||||

$ + K @ 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:

{bmc pophint1.bmp}

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:

{bmc pophint2.bmp}

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

Using_Code_Window_Popup_Debug_Evaluation_Hints

$ Using Code Window Popup Debug Evaluation Hints

+ ICD08Z:0

K Using Code Window Popup Debug Evaluation Hints

@ Status|0|||0||||||