M68ICS08GPOM/D
REV. 1.0

M68ICS08GP
In-Circuit Simulator

Operator's Manual

MOTOROLA

List of Sections

Section 1. Introduction 25

Section 2. Hardware Installation 35

Section 3. Software Installation and Initialization 41

Section 4. WinIDE User Interface 47

Section 5. CASM08W Assembler Interface 91

Section 6. ICS08GPW In-Circuit Simulator
User Interface 113

Section 7. Debugging with ICS08GPW. 153

Section 8. PROG08SW FLASH Programmer 163

Section 9. ICD08SW In-Circuit Debugger 171

Section 10. Debugging Command Set 193

Section 11. Example Project. 323

Section 12. Using the MON08 Interface. 329

Appendix A. S-Record Information 333

Appendix B. Technical Reference
and Troubleshooting. 341

Glossary 377

Index. 383

List of Sections

Table of Contents

Section 1. Introduction

1.1 Contents 25

1.2 Overview. 25

1.3 Toolkit Components 26

1.4 Hardware and Software Requirements 27

1.5 Toolset Features 28

1.6 Specifications 29

1.7 Typographic Conventions. 29

1.8 Quick Start Instructions 31

1.9 MC68HC908GP Security Feature 34

Section 2. Hardware Installation

2.1 Contents 35

2.2 Overview. 35

2.3 Configuring the In-Circuit Simulator Board. 36

2.4 Assembling the Hardware Pod 37

2.5 Connecting to a Target System. 38

Section 3. Software Installation and Initialization

3.1 Contents 41

3.2 Overview. 41

3.3 ICS08GP Software Components. 41
3.3.1 WinIDE Editor 42
3.3.2 CASM08W Assembler 42
3.3.3 ICS08GPW In-Circuit Simulator 43
3.3.4 ICD08SW In-Circuit Debugger (ICD). 43
3.3.5 PROG08SW FLASH Programmer 43

3.4 Installing the ICS08GP Software 44
3.4.1 Installation Steps 44
3.4.2 Starting the ICS08GP Software 45
3.4.3 Pod-to-Host Communication 45

Section 4. WinIDE User Interface

4.1 Contents 47

4.2 Overview. 48

4.3 Windows Integrated Development Environment 49

4.4 WinIDE Main Window. 50
4.4.1 Main Window Functions 50
4.4.2 Main Window Components 50

4.5 Getting Started 52
4.5.1 Prerequisites for Starting the WinIDE Editor 52
4.5.2 Starting the WinIDE Editor 52
4.5.3 Opening Source Files 52

4.6 Navigating in the WinIDE Editor 53
4.6.1 Using Markers 54

4.7 Command-Line Parameters 55

4.8 WinIDE Toolbar 56

4.9 WinIDE Menus 58

4.10 WinIDE File Options 59
4.10.1 New File. 60
4.10.2 Open File 60
4.10.3 Save File. 61
4.10.4 Save File As 61
4.10.5 Close File 61
4.10.6 Print File. 62
4.10.7 Print Setup 62
4.10.8 Exit. 62

4.11 WinIDE Edit Options 63
4.11.1 Undo. 63
4.11.2 Redo 64
4.11.3 Cut 64
4.11.4 Copy 65
4.11.5 Paste 65
4.11.6 Delete 65
4.11.7 Select All 65

4.12 WinIDE Environment Options 66
4.12.1 Open Project. 67
4.12.2 Save Project 67
4.12.3 Save Project As 68
4.12.4 Close/New Project 68
4.12.5 Setup Environment. 68
4.12.5.1 General Environment Tab 69
4.12.5.2 General Editor Tab 71
4.12.5.3 Assembler/Compiler Tab 73
4.12.5.4 Executable Tabs: EXE 1-3 78
4.12.6 Setup Fonts. 80

4.13 WinIDE Search Options 81
4.13.1 Find 82
4.13.2 Replace. 83
4.13.3 Find Next 83
4.13.4 Go to Line 84

4.14 WinIDE Window Options. 84
4.14.1 Cascade 85
4.14.2 Tile 86

4.14.3 Arrange Icons 87
4.14.4 Minimize All 88
4.14.5 Split 89

Section 5. CASM08W Assembler Interface

5.1 Contents 91

5.2 Introduction. 92

5.3 CASM08W Assembler User Interface 93

5.4 Assembler Parameters. 94

5.5 Assembler Outputs 95
5.5.1 Object Files 95
5.5.2 Map Files 95
5.5.3 Listing Files 96
5.5.4 Error Files 97
5.5.5 Files from Other Assemblers 97

5.6 Assembler Options 98
5.6.1 Operands and Constants. 98
5.6.2 Comments 99

5.7 Assembler Directives 99
5.7.1 BASE 99
5.7.2 Cycle Adder 100
5.7.3 Conditional Assembly 102
5.7.4 INCLUDE 102
5.7.5 MACRO. 103

5.8 Listing Directives 104
5.8.1 Listing Files 105
5.8.2 Labels. 107

5.9 Pseudo Operations 107
5.9.1 Equate (EQU). 108
5.9.2 Form Constant Byte (FCB) 108
5.9.3 Form Double Byte (FDB) 109
5.9.4 Originate (ORG) 109
5.9.5 Reserve Memory Byte (RMB). 109

5.10 Assembler Error Messages 109

5.11 Using Files from Other Assemblers 111

Section 6. ICS08GPW In-Circuit Simulator User Interface

6.1 Contents 113

6.2 Introduction. 115

6.3 ICS08GPW Description 115
6.3.1 ICS08GPW Simulation Speed 116
6.3.2 System Requirements for ICS08GPW. 116
6.3.3 File Types and Formats 116

6.4 Starting ICS08GPW 118

6.5 ICS08GPW Windows. 120

6.6 Code Windows 121
6.6.1 To Display the Code Windows Shortcut Menus 121
6.6.2 Code Window Shortcut Menu Functions. 122
6.6.3 Code Window Keyboard Commands 123

6.7 Variables Window 124
6.7.1 Displaying the Variables Shortcut Menu. 124
6.7.2 Variables Shortcut Menu Options 124
6.7.3 Variable Window Keyboard Commands. 126

6.8 Memory Window 126

6.9 Status Window 128

6.10 CPU Window 130
6.10.1 Changing Register Values 130
6.10.2 CPU Window Keyboard Commands. 132

6.11 Cycles Window. 132

6.12 Stack Window. 132
6.12.1 Interrupt Stack 133
6.12.2 Subroutine Stack 133

6.13 Trace Window. 134

6.14 Breakpoint Window 135
6.14.1 Adding a Breakpoint 136
6.14.2 Editing a Breakpoint 137
6.14.3 Deleting a Breakpoint 137
6.14.4 Removing All Breakpoints. 138

6.15 Register Block Window 138

6.16 Entering Debugging Commands. 140

6.17 ICS08GPW Toolbar 140

6.18 ICS08GPW Menus 142

6.19 File Options. 143
6.19.1 Load S19 File 144
6.19.2 Reload Last S19 144
6.19.3 Play Macro 145
6.19.4 Record Macro. 145
6.19.5 Stop Macro. 146
6.19.6 Open Logfile 146
6.19.7 Close Logfile 148
6.19.8 Exit. 148

6.20 ICS08GPW Execute Options 148
6.20.1 Reset Processor 149
6.20.2 Step. 149
6.20.3 Multiple Step 149
6.20.4 Go. 149
6.20.5 Stop 150
6.20.6 Repeat Command. 150

6.21 ICS08GPW Window Options. 150
6.21.1 Open Windows. 151
6.21.2 Change Colors 151
6.21.3 Reload Desktop 152
6.21.4 Save Desktop 152

Section 7. Debugging with ICS08GPW

7.1 Contents 153

7.2 Introduction. 153

7.3 ICS08GPW Debugging Command Syntax. 154

7.4 Command Set Summary 155

Section 8. PROG08SW FLASH Programmer

8.1 Contents 163

8.2 Introduction. 164

8.3 Startup and Parameters 165

8.4 Programming Commands 166
8.4.1 BM — Blank-Check Module 166
8.4.2 CM — Choose Module .08P 166
8.4.3 EM — Erase Module 167
8.4.4 PB — Program Bytes 167
8.4.5 PM — Program Module. 167
8.4.6 SM — Show Module 167
8.4.7 SS — Specify S Record 168
8.4.8 UM — Upload Module 168
8.4.9 UR — Upload Range 168
8.4.10 VM — Verify Module 168
8.4.11 VR — Verify Range 168
8.4.12 QU — QUit 169
8.4.13 RE — REset Chip 169
8.4.14 HE — HElp 169

8.5 Programming Example 169

Section 9. ICD08SW In-Circuit Debugger

9.1 Contents 171

9.2 Introduction. 172

9.3 MON08 Debugging Limitations and Tips 172

9.4 Startup and Parameters 175

9.5 User Interface 176
9.5.1 Status Window 176
9.5.1.1 Pop-Up Menu 176
9.5.1.2 Keystrokes 177
9.5.2 Code Window 177
9.5.2.1 Pop-Up Menu 178
9.5.2.2 Keystrokes 179
9.5.3 Variables Window 180
9.5.3.1 Pop-Up Menu 180
9.5.3.2 Keystrokes 181
9.5.4 Memory Window 181
9.5.4.1 Pop-Up Menu 182
9.5.4.2 Keystrokes 182
9.5.5 Colors Window 183
9.5.6 CPU08 Window 184
9.5.6.1 Pop-Up Menu 184
9.5.6.2 Keystrokes 185

9.6 Debugging Commands 185
9.6.1 Syntax and Nomenclature 185
9.6.2 Command Recall 186
9.6.3 Command Set Summary 187

Section 10. Debugging Command Set

10.1 Contents 193

10.2 Command Descriptions. 196
A or ACC Set Accumulator Value 197
ADCLR Clear A/D Input Buffer 198
ADDI Input Data to A/D Converter 199
ASCIIF3
and ASCIIF6 Toggle ASCII Display 200
ASM Assemble Instructions 201
BELL Sound PC Bell 202
BF or FILL Block Fill Memory 203
BR Set Instruction Breakpoint 204
BREAKA Set Accumulator Breakpoint 206

BREAKHX	Set HX Register Breakpoint	208
BREAKSP	Set Stack Pointer Breakpoint	210
C	Set/Clear Carry Bit	212
CAPTURE	Capture Changed Data	213
CAPTUREFILE or CF	Open/Close Capture File	214
CCR	Set Condition Code Register	215
CGMXCLK	Set CGMXCLK to Bus Clock Relationship	216
CHIPMODE	Choose Device for Simulation	218
CLEARMAP	Cear .MAP File	219
CLEARSYMBOL	Clear User Symbols	220
CODE	Show Disassembled Code	221
COLORS	Set Simulator Colors	222
CYCLES or CY	Set Cycles Counter	223
DASM	Disassemble Memory	224
DDRA	Set Port A Direction Register	225
DDRB	Set Port B Direction Register	226
DDRC	Set Port C Direction Register	227
DDRD	Set Port D Direction Register	228
DDRE	Set Port E Direction Register	229
DUMP	Dump Memory to Screen	230
EVAL	Evaluate Expression	231
EXIT or QUIT	Exit/Quit Application	232
G, GO, or RUN	Begin Program Execution	233
GOEXIT	Execute Without Breakpoints/Debugger	234
GONEXT	Execute Past Subroutine/Interrupt	235
GOMACRO	Execute Macro after Break	236
GOTIL	Execute Until Address	237
GOTOCYCLE	Execute to Cycle Counter Value	238
H or HREG	Set/Clear Half-Carry Bit	239
HELP	Open Help	240
HX	Set H:X Index Register Pair	241
I	Set/Clear Interrupt Mask	242
INFO	Display Line Information	243
INPUTA	Set Port A Inputs	244
INPUTB	Set Port B Inputs	245
INPUTC	Set Port C Inputs	246
INPUTD	Set Port D Inputs	247

INPUTE	Set Port E Inputs	248
INPUTS	Show Port Inputs	249
INT or IRQ	Set IRQ Pin State	250
LF or LOGFILE	Open/Close Logfile	251
LISTOFF	Turn Off Step Listing	252
LISTON	Turn On Step Listing	253
LOAD	Load S Records	254
LOADALL	Load S Records and Map File	255
LOADDESK	Load Desktop Settings	256
LOADMAP	Load Map File	257
LOADV	Load and Verify	258
LOAD_BIN	Load Binary File	259
LOADV_BIN	Load and Verify Binary File	260
MACRO	Execute Batch File	261
MACROEND	Stop Saving Commands to Batch File	263
MACROSTART	Save Debug Commands to Batch File	264
MACS	List Macros	265
MAP	Show Information in Map File	266
MD or MD1	Display Memory at Address	267
MD2	Display Memory (Window 2) at Address	268
MM or MEM	Modify Memory	269
NOBR	Remove Breakpoints	272
NOMAP	Clear .MAP File	273
NOSYMBOL	Clear User Symbols	274
PC	Set Program Counter	275
POD	Change Serial Port	276
PORTA or PRTA	Set Port A Output Latches	277
PORTB or PRTB	Set Port B Output Latches	278
PORTC or PRTC	Set Port C Output Latches	279
PORTD or PRTD	Set Port D Output Latches	280
PORTE or PRTE	Set Port E Output Latches	281
QUIET	Toggle Window Refresh	282
R	Use Register Files	283
REG	Show Registers	284
REM	Place Comment in Batch/Macro File	285
RESET	Simulate Processor Reset	286
RESETGO	Reset and Restart MCU	287
SCCLR	Clear SCI I/O Buffers	288

SCDI	Input SCI Data	289
SCDO	View SCI Outputs	290
SAVEDESK	Save Desktop Settings	291
SHOWBREAKS	Display Breakpoint Window	292
SHOWCODE	Display Code at Address	293
SHOWMAP	Show Information in Map File	294
SHOWPC	Display Code at PC Address	295
SHOWTRACE	Display Trace Window	296
SIM08	Switch Simulation Mode	297
SNAPSHOT	Save Window Data to Logfile	298
SOURCEPATH	Set Path for Source Code	299
SP	Set Stack Pointer	300
SPCLR	Clear SPI I/O Buffers.	301
SPDI	Enter SPI Inputs	302
SPDO	View SPI Outputs	303
SPFREQ	Set Input Clock for SPI Slave	304
SS	Execute Source Step(s)	305
ST or STEP or T	Execute Single Step	306
STACK	Show Stack Window	307
STATUS	Show Registers	308
STEPFOR	Step Forever	309
STEPTIL	Step Until Address	310
SYMBOL	Add Symbol	311
TRACE	Enable/Disable Tracing	312
UPLOAD_SREC	Upload S Record to Screen	313
V	Set or Clear V Bit (CCR)	314
VAR	Display Variable	315
VERIFY	Verify S-Record File	316
VERSION or VER	Display Software Version	317
WAIT	Wait for n Cycles	318
WHEREIS	Display Symbol Value	319
X or XREG	Set X Register Value	320
Z	Set/Clear Zero Bit	321

Section 11. Example Project

11.1 Contents 323

11.2 Introduction. 323

11.3 Setting Up a Sample Project 323
11.3.1 Set Up the Environment 324
11.3.2 Create the Source Files 325
11.3.3 Assemble the Project 326

Section 12. Using the MON08 Interface

12.1 Contents 329

12.2 Introduction. 329

12.3 Header Placement and Layout 329

12.4 Connecting to the In-Circuit Simulator. 331

12.5 Disabling the Target-System Interface 332

Appendix A. S-Record Information

A.1 Contents 333

A.2 Introduction. 333

A.3 S-Record Content 334

A.4 S-Record Types. 335

A.5 S-Record Creation. 336

A.6 S-Record Example 336
A.6.1 S0 Header Record 337
A.6.2 First S1 Record. 338
A.6.3 S9 Termination Record 339
A.6.4 ASCII Characters 339

Appendix B. Technical Reference and Troubleshooting

B.1 Contents 341

B.2 Introduction. 341

B.3 Functional Description 342
B.3.1 ICS08GP20 Board 342
B.3.2 SPGMR08 Serial Programmer Base Unit 344

B.4 Troubleshooting the Quick Start. 344

B.5 Troubleshooting MON08 Mode 351

B.6 Connector Pin Assignments 355

B.7 Target-Cable Pin Assignments 364

B.8 Parts List 367

B.9 Board Layout and Schematic Diagram. 369

Glossary

Glossary 377

Index

Index 383

List of Figures

Figure	Title	Page
1-1	WinIDE Debugger Toolbar Button	33
1-2	WinIDE Assemble/Compile File Toolbar Button	33
1-3	WinIDE Program Toolbar Button	33
1-4	WinIDE Real-Time Debugger Toolbar Button	34
1-5	MC68HC908GP Security Dialog Window	34
3-1	Pick Device Dialog Window	45
3-2	Can't Contact Board Dialog Window	46
4-1	WinIDE Window Components	49
4-2	WinIDE Status Bar	51
4-3	WinIDE Edit Shortcut Menu	54
4-4	Marker Submenu	55
4-5	WinIDE Toolbar	56
4-6	WinIDE File Menu	59
4-7	Open File Dialog Window	60
4-8	Print Dialog Window	62
4-9	WinIDE Edit Menu	63
4-10	WinIDE Environment Menu	66
4-11	Specify Project File to Open Dialog Window	67
4-12	Specify Project File to Save Dialog Window	68
4-13	Environment Settings Dialog General Environment Tab	69
4-14	Environment Settings Dialog: General Editor Tab	71
4-15	Environment Settings Dialog: Assembler/Compiler Tab	74
4-16	Error Format List	78
4-17	Environment Settings Dialog: EXE Tabs	79
4-18	Setup Fonts Dialog Window	80
4-19	Search Menu	81
4-20	Find Dialog Window	82

Figure Title Page

4-21 Replace Dialog Window 83
4-22 Go to Line Number Dialog Window 84
4-23 WinIDE Window Menu 84
4-24 WinIDE with Subordinate Windows Cascaded 85
4-25 WinIDE with Subordinate Windows Tiled 86
4-26 WinIDE: One Source Window Displayed, Remaining Windows Minimized 87
4-27 WinIDE with Subordinate Windows Minimized 88
4-28 WinIDE Cascaded Windows with Active Window Split 89

5-1 WinIDE with CASM08W Assembler Window Displayed 93

6-1 Can’t Contact Board Dialog Window 119
6-2 ICS08GPW Windows Default Positions 120
6-3 Code Window 2 in Disassembly Mode with Breakpoint Toggled 121
6-4 Code Window Shortcut Menu 121
6-5 Window Base Address Dialog Window 122
6-6 Variables Window with Shortcut Menu 124
6-7 Add Variable Dialog Window 125
6-8 Memory Window with Shortcut Menu 127
6-9 Status Window 128
6-10 Results of Entering the LF Command in the Status Window 129
6-11 Specify Output LOG File! Dialog Window 129
6-12 Logfile Already Exists Window 130
6-13 CPU08 Window with Shortcut Menu 131
6-14 Change CCR Dialog Window 131
6-15 Cycles Window 132
6-16 Stack Window 133
6-17 Trace Window 134
6-18 Breakpoint Window with Shortcut Menu 135
6-19 Edit Breakpoint Dialog Window 136
6-20 Register Block Window 138
6-21 Register Files Window 139
6-22 Register Window 139
6-23 ICS08GPW Toolbar 140

Figure	Title	Page
6-24	ICS08GPW File Menu	143
6-25	Specify S19 File to Load Dialog Window	144
6-26	Specify MACRO File to Execute Dialog Window	145
6-27	Specify MACRO File to Record Dialog Window	145
6-28	Specify Output LOG File Dialog Window	146
6-29	Logfile Already Exists Dialog Window	147
6-30	Sample Output Logfile	147
6-31	ICS08GPW Execute Menu	148
6-32	ICS08GPW Window Menu	150
6-33	Change Window Colors Dialog Window	151
9-1	ICD08SW Status Window	176
9-2	ICD08SW Code Window 2	178
9-3	ICD08SW Variables Window	180
9-4	ICD08SW Memory Window	181
9-5	ICD08SW Change Window Colors Window	183
9-6	ICD08SW CPU08 Window	184
9-7	ICD08SW Set CCR Window	185
10-1	Assembly Window – ASM Command	201
10-2	Modify Memory Dialog Window	269
11-1	CASM08W Window	327
12-1	MON08 Target-System Connector Layout	331
12-2	Target System Stand-Alone Connection	332
B-1	ICS08GP20 Board Layout	369

List of Tables

Table	Title	Page
1-1	M68UCS08GP Specifications	29
2-1	W1 Configuration Header — Oscillator Source	36
2-2	W2 Configuration Header — Target Cable Reset Pin Function.	36
2-3	W3 Configuration Header — Target System V_{DD}	37
3-1	ICS08GP Software Files	44
4-1	Navigating Between External Programs	53
4-2	WinIDE Editor Substitution Parameters	56
4-3	WinIDE Toolbar Buttons	57
4-4	WinIDE Menus and Options Summary	58
5-1	Change Base Prefixes/Suffixes	99
5-2	Assembler Directives	101
5-3	Listing Directives	104
5-4	Listing File Fields	105
5-5	Pseudo Operations Allowed by the CASM08W.	107
5-6	Assembler Error Messages	110
6-1	Base Prefixes and Suffixes	125
6-2	ICS08GPW Toolbar Buttons	141
6-3	ICS08GPW Menus and Options Summary.	142
7-1	Argument Types	155
7-2	ICS08GPW Command Overview	156
9-1	ICD08SW Command Overview	187

List of Tables

Table	Title	Page
12-1	MON08 Target System Connector P1	330
12-2	MON08 Target System Connector P2	330
A-1	S-Record Fields	334
A-2	S-Record Field Contents	335
A-3	S-Record Types	336
A-4	S0 Header Record	337
A-5	S1 Header Record	338
A-6	S-9 Header Record	339
B-1	Target DIP Connector J2	356
B-2	Target A Connector J3	358
B-3	Target B Connector J4	360
B-4	SPGMR Connectors J7 and J8	362
B-5	Auxiliary Port C Logic Analyzer Connector J1	362
B-6	Auxiliary Port D Logic Analyzer Connector J5	362
B-7	MON08 Connector J6	363
B-8	Flex Target Cable (M68CBL05C) and QFP Target Head Adapater (MCTC08GP20FB44)	364
B-9	Target MON08 Cable	367
B-10	ICS08GP20 B0ard Parts List	367

Section 1. Introduction

1.1 Contents

1.2 Overview. 25
1.3 Toolkit Components 26
1.4 Hardware and Software Requirements 27
1.5 Toolset Features 28
1.6 Specifications 29
1.7 Typographic Conventions. 29
1.8 Quick Start Instructions 31
1.9 MC68HC908GP Security Feature 34

1.2 Overview

This chapter provides an overview of the M68ICS08GP in-circuit simulator kit components and a quick start guide to setting up a development project.

The Motorola M68ICS08GP in-circuit simulator kit is a development toolkit for designers who develop and debug target systems that incorporate MC68HC908GP20 and MC68HC908GP32 microcontroller unit (MCU) devices. The toolkit contains all of the hardware and software needed to develop and simulate source code and program Motorola's MC68HC908GP20 and MC68HC908GP32 microcontrollers.

The toolkit consists of two main parts:

- Hardware pod, an ICS08GP20 circuit board mounted on an M68HC08 serial programmer (SPGMR)
- ICS08GP software package

Together, the pod and the ICS08GP software form a complete simulator and non-real-time input/output (I/O) emulator for MC68HC908GP20 and MC68HC908GP32 devices. When the pod is connected to a host PC and target hardware, the actual inputs and outputs of the target system can be used during code simulation.

With the ICD08SW software, the SPGMR can be used as a limited real-time emulator. With the PROG08SW software, the SPGMR can be used to program MCU FLASH memory.

Use the M68ICS08GP toolkit with computers with a serial port based on Windows 95® or later versions.

1.3 Toolkit Components

The complete M68ICS08GP toolkit contains:

- M68ICS08GP hardware:
  - ICS08GP20 in-circuit simulator board
  - Two MC68HC908GP32FB MCUs
  - 40-pin DIP-header ribbon cable for target connection
  - 16-pin ribbon cable for MON08 monitor mode connection
- M68SPGMR08 serial programmer hardware (see note):
  - M68HC08 serial programmer (SPGMR)
  - AC/DC power-supply adapter
  - SPGMR-to-host serial COM cable

NOTE: *If the serial programmer hardware is not present in the toolkit, it must be obtained separately. Order the M68SPGMR08 serial programmer kit by contacting a local Motorola representative.*

- Windows®-optimized software components, collectively referred to as the ICS08GP software, consisting of:
  - WINIDE.EXE, the integrated development environment (IDE) software interface into the MC68HC908GP20- and MC68HC908GP32-based hardware for editing and performing software or in-circuit simulation
  - CASM08W.EXE, the CASM08W command-line cross-assembler
  - CS08GPW.EXE, the in-circuit/stand-alone simulator software for the MC68HC908GP20 or MC68HC908GP32 MCU
  - PROG08SW.EXE, the FLASH memory programming software
  - ICD08SW.EXE, the limited, real-time, in-circuit debugging software
- Documentation is this manual, *M68ICS08GP In-Circuit Simulator Operator's Manual*

1.4 Hardware and Software Requirements

The ICS08GP software requires this minimum hardware and software configuration:

- An IBM-compatible host computer running Windows 95 or later version operating system
- Approximately 640 Kbytes of memory (RAM) and 2 Mbytes free drive space
- A serial port for communications between the M68ICS08GP pod and the host computer

1.5 Toolset Features

The M68ICS08GP toolkit is a low-cost development system that supports editing, assembling, in-circuit simulation, in-circuit emulation, and FLASH memory programming.

Its features include:

- Editing with WinIDE
- Assembling with CASM08
- FLASH memory programming with PROG08SW
- In-circuit and stand-alone simulation of M68HC908GP MCUs with ICS08GPW, including:
  - Simulation of all instructions, memory, and peripherals
  - Optional simulator pin inputs from the hardware
  - Conditional breakpoints, script files, and log files
- Limited real-time emulation and debugging with ICD08SW, including:
  - Loading code into RAM
  - Executing real-time in RAM or FLASH
  - One hardware breakpoint in FLASH
  - Multiple breakpoints in RAM
- On-line help documentation for all software
- Software integrated into the WinIDE environment, allowing hotkey access to all applications
- Emulation connection to the hardware

1.6 Specifications

Table 1-1 summarizes the M68ICS08GP hardware specifications.

Table 1-1. M68UCS08GP Specifications

Characteristic	Specification
Temperature: Operating Storage	 0° to 40°C –40° to +85°C
Relative humidity	0 to 95%, non-condensing
Power requirement	+5 Vdc, from included AC/DC adapter

NOTE: *The procedural instructions in this operator's manual assume that the user is familiar with the Windows interface and selection procedures.*

Figures in this manual show ICS08GPW windows and dialog boxes as they appear in the Windows 95 environment.

1.7 Typographic Conventions

This operator's manual uses special typographical conventions to enhance readability. They are:

- Code, statements, confirmations, data entry, field text, parameters, and strings are indicated in regular Courier:

  `$INCLUDE`
  `"INIT.AS"`

  This option displays an `Exit Application` confirmation message.

  This new filename replaces the `[NONAME#1]` in the title bar.

  `%FILE%`

- Window names and parts of windows are indicated in initial caps, unless the name of the window is capitalized in a unique way:

  Memory and Code windows

  CASM08W window

  WinIDE main window

- For usage in this manual, filenames are not case sensitive. But for consistency, they will always appear in all capital letters in Times:

  SETUP.EXE
  MAP file

- Buttons, icons, functions, and keyboard keys are indicated in Times small caps:

  Press the ENTER key.

  Type CTRL + N or click on the NEW toolbar button.

  The RESET function is an input and output.

- Commands are not case sensitive. But for consistency, they will always appear in all capital letters in Times, unless they contain some peculiarity:

  INPUTx
  UNDO
  LOADMAP

- Menu names, options, and tabs, and dialog, edit, text and lists boxes are indicated in Times bold:

  Do this by checking the **Main File** option in the **Environment Settings** dialog’s **General Options** tab.

  Open the **Open File** dialog.

  Select the filename in the **File Name** list, and use the filename in the **Main filename** edit box.

1.8 Quick Start Instructions

For users experienced in installing Motorola or other development tools, the following steps provide a quick-start installation procedure for the M68ICS08GP hardware and software.

For more complete instructions, refer to **Section 2. Hardware Installation**. If problems occur with the quick start procedures, refer to **Appendix B. Technical Reference and Troubleshooting** for troubleshooting instructions.

1. Install the ICS08GP software package.

   To start the software installation, run the SETUP.EXE program on diskette 1. During installation, follow the instructions in the installation wizard.

2. Assemble and connect the hardware pod (*see Electrostatic Discharge Caution in **2.2 Overview***):

   a. Install the MCU into the ICS08GP20 board.

      Locate socket XU3 on the board. Install the MCU (provided with the M68ICS08GP package) into this socket, observing the pin 1 orientation with the socket's notch. The top (label side) of the MCU package must be visible when looking at the component side of the ICS08GP20 board.

   b. Install the ICS08GP20 board onto the SPGMR base unit.

      Make certain that power is off before performing this step.

      Fit the two 10-pin SIP (single in-line package) headers on back of the board into the sockets on the SPGMR. The Motorola logo on both the board and the SPGMR must be visible and aligned in the same direction. The two LEDs on the SPGMR must be visible.

   c. Connect the SPGMR to the host PC.

      Locate the 9-pin connector labeled HOST on the SPGMR. Using the cable provided, connect it to a serial COM port on the host PC.

d. Apply power to the pod.

   Connect the 5-volt power supply to the round connector on the SPGMR. Plug the power supply into an AC power outlet, using one of the country-specific adapters provided. The system power LED on the SPGMR should light.

3. Start the WinIDE editor and open a project file.

   Start the editor either from the Windows **Start** menu or by double clicking its icon. From the WinIDE **Environment** menu, choose the **Open Project** option, and choose a project file from the **Specify project file** to open dialog. Note that the environment is pre-configured.

   If no project file exists, choose the **New** option from the **File** menu to create a new project file. The desktop and environment settings made in the **Environment Settings** dialog are stored in the WINIDE.INI file and read each time the WinIDE editor is started. The project-specific desktop and environment settings can also be saved in a project file (*.PPF), which is read when the project is opened, allowing it to be saved and used as a general environment as well as custom environments for individual projects.

   To create the project file:

   a. Specify the project-specific desktop and environment settings in the WinIDE editor.

   b. Choose the **Save Project As** option from the WinIDE **Environment** menu to name and save the project to a directory folder.

   For more information about setting up a new project, refer to **Section 11. Example Project**.

4. Run the ICS08GPW simulator.

   If communications are not established with the pod, it may be necessary to select the proper port (COM1 or COM2) and baud rate (9600). When communications are established, the SPGMR's socket power LED will light.

With a project or source file open in the WinIDE main window, click the DEBUGGER button (**Figure 1-1**) on the WinIDE toolbar to start the ICS08GPW debugger and debug the contents of the active source window. For more information about debugging with ICS08GPW, refer to section **6.16 Entering Debugging Commands** and to **Section 7. Debugging with ICS08GPW**.

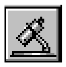


Figure 1-1. WinIDE Debugger Toolbar Button

5. Assemble the code.

   Press the ASSEMBLE/COMPILE FILE button (**Figure 1-2**) on the WinIDE toolbar to assemble the source code in the active WinIDE window. Additional information about the CASM08W assembler can be found in **Section 5. CASM08W Assembler Interface**.

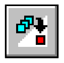


Figure 1-2. WinIDE Assemble/Compile File Toolbar Button

6. Run the PROG08SW programmer.

   Press the PROGRAM button (**Figure 1-3**) on the WinIDE toolbar to start the programmer. Additional information about PROG08SW can be found in **Section 8. PROG08SW FLASH Programmer**.

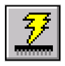


Figure 1-3. WinIDE Program Toolbar Button

7. Run the ICD08SW real-time debugger.

   Press the REAL-TIME DEBUGGER button (**Figure 1-4**) on the WinIDE toolbar to start the ICD08SW in-circuit debugger and emulator. Additional information can be found in **Section 9. ICD08SW In-Circuit Debugger**.

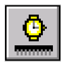


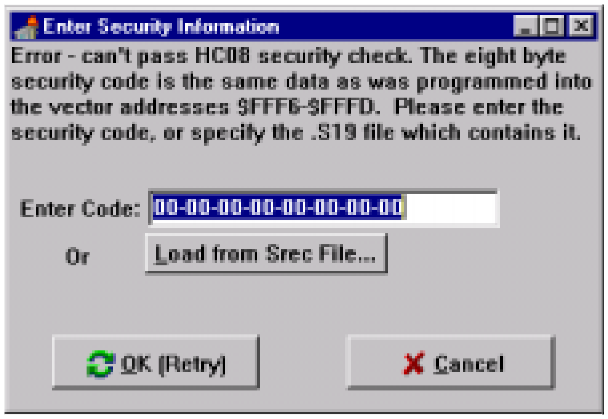
Figure 1-4. WinIDE Real-Time Debugger Toolbar Button

1.9 MC68HC908GP Security Feature

The MC68HC908GP20 and MC68HC908GP32 MCUs contain a security feature based on information that the user programs into the part. Security bytes are specified in addresses $FFF6–$FFFD. The PROG08SW software continually records any changes to these security bytes and stores them in the file SECURITY.INI.

The information in this file is also shared with the ICS08GPW in-circuit simulator and the ICD08SW in-circuit debugger software. This allows the user to reset the device and still have access to the monitor mode. The ICS08GPW software automatically attempts to access the part by trying the default (nothing written) and up to the last 10 sequences of bytes that have been written to the part.

If, after trying each of the sequences of bytes stored in the SECURITY.INI file, the ICS08GPW software is unable to access the part, a security dialog box is displayed (**Figure 1-5**). This dialog allows the user to enter the security bytes manually or to read the information from the S19 file from which the MCU was last programmed.



Enter Security Information

Error - can't pass HC08 security check. The eight byte security code is the same data as was programmed into the vector addresses $FFF6-$FFFD. Please enter the security code, or specify the .S19 file which contains it.

Enter Code: 00-00-00-00-00-00-00-00

Figure 1-5. MC68HC908GP Security Dialog Window

Section 2. Hardware Installation

2.1 Contents

2.2 Overview 35
2.3 Configuring the In-Circuit Simulator Board. 36
2.4 Assembling the Hardware Pod 37
2.5 Connecting to a Target System. 38

2.2 Overview

This chapter explains how to:

- Configure the ICS08GP20 in-circuit simulator board
- Assemble the ICS08GP20 board and the M68HC08 serial programmer (SPGMR) to form the hardware pod and connect the pod to a host PC
- Connect the pod to a target system

In interactive mode, the pod is connected to the serial port of a host PC. The actual inputs and outputs of a target system can be used during simulation of source code.

In stand-alone mode, the pod is not connected to the PC. The ICS08GPW software can be used as a stand-alone simulator.

ESD CAUTION: *Ordinary amounts of static electricity from clothing or the work environment can damage or degrade electronic devices and equipment. For example, the electronic components installed on printed circuit boards are extremely sensitive to electrostatic discharge (ESD). Wear a grounding wrist strap whenever handling any printed circuit board. This strap provides a conductive path for safely discharging static electricity to ground.*

2.3 Configuring the In-Circuit Simulator Board

Three configuration headers provide for jumper-selectable hardware options. **Table 2-1**, **Table 2-2**, and **Table 2-3** describe these settings.

NOTE: *The factory-default settings should be used when following the quick start procedure described in* ***1.8 Quick Start Instructions****.*

Table 2-1. W1 Configuration Header — Oscillator Source

Pin	Direction	Signal Name	Description
1	Out	SP_OSC	4.9152-MHz SPGRM oscillator output
2	In or out	OSC1	OSC1 on sockets and target connectors

Jumper on pins 1-2, default — The SPGMR oscillator is selected.

Jumper off — Allows using an oscillator on the target system or injecting a different clock rate at W1 pin 2.

Table 2-2. W2 Configuration Header — Target Cable Reset Pin Function

Pin	Direction	Signal Name	Description
1	Out	RST_$\overline{\text{OUT}}$	Reset signal to target system — 0 to +5 Vdc output reflecting state of MCU $\overline{\text{RST}}$ signal
2	In or out	TGT_$\overline{\text{RST}}$	To/from target $\overline{\text{RST}}$ pins
3	In	RST_$\overline{\text{IN}}$	Reset signal from target system — 0 to +5 Vdc input to control state of MCU $\overline{\text{RST}}$ signal

Jumper on pins 1 and 2, default — The target system's $\overline{\text{RESET}}$ is not allowed to reset the ICS08GP20 MCU.

Table 2-3. W3 Configuration Header — Target System V_{DD}

Pin	Signal Name	Description
1	VDD	System power
2	TGT_VDD	To target V_{DD}, V_{DDA}, and V_{DDAD} pins

Jumper on pins 1 and 2, default — ICS08GP20 system power is applied to the target cable V_{DD}, V_{DDA}, and V_{DDAD} pins.

Jumper off — Allows using a separate power supply for the target system

2.4 Assembling the Hardware Pod

Before beginning, locate these items:

- Two 10-pin SIP connectors on the solder side of the ICS08GP20 in-circuit simulator board
- 9-pin RS-232 serial connector labeled HOST on the SPGMR
- 5-volt circular power-input connector on the SPGMR

To assemble the pod and prepare it for use with a host PC:

1. Install the MCU into the ICS08GP20 board.

   Locate socket XU3 on the board. Install the MCU (provided with the M68ICS08GP package) into this socket upside down, observing the pin 1 orientation with the socket's notch. The top (label side) of the MCU package must be visible when looking at the component side of the ICS08GP20 board.

2. Install the ICS08GP20 board onto the SPGMR base unit.

   Make certain that power is off before performing this step.

   Fit the two 10-pin SIP headers on back of the board into the sockets on the SPGMR. The Motorola logo on both the board and the SPGMR must be visible and aligned in the same direction. The two LEDs on the SPGMR must be visible.

3. Connect the SPGMR to the host PC.

   Locate the 9-pin connector labeled HOST on the SPGMR. Using the cable provided, connect it to a serial COM port on the host PC.

4. Apply power to the pod.

   Connect the 5-volt power supply to the round connector on the SPGMR. Plug the power supply into an AC power outlet, using one of the country-specific adapters provided. The system power LED on the SPGMR should light.

2.5 Connecting to a Target System

The two ways to connect the ICS08GP20 simulator board to a target system are:

1. Using the MCU on the ICS08GP20, break its processor signals out to the target system.

   This method allows the ICS08GP20's MCU to control the target system's hardware. Either an MC68HC908GP20 or MC68HC908GP32 MCU must be installed on the ICS08GP20 board. The target system's MCU must be removed.

   The processor signals can be routed to the target system in two ways:

   a. A 40-pin DIP emulation cable is provided with the kit for use with the connector labeled DIP on the ICS08GP20 board. Attach the cable to an equivalent connector on the target system. The pin assignments for this connector are given in **Appendix B. Technical Reference and Troubleshooting**.

   b. Connectors A and B on the ICS08GP20 board may be used with a flex emulation cable and target head adapter, which are available separately. Target head adapters are available for the QFP (quad flat pack), PLCC (plastic-leaded chip carrier), and DIP (dual in-line package) footprints on the target board.

2. Use the MON08 debug interface for communication with the target system's MCU.

   This method allows in-circuit FLASH programming and debugging of the target system's MCU. Either an MC68HC908GP20 or MC68HC908GP32 MCU must be installed in the target system. The ICS08GP20's MCU must be removed.

   Connect the ICS08GP20's MON08 connector with a compatible MON08 connector on the target system. Complete instructions for constructing this interface on the target board are found in **Section 12. Using the MON08 Interface**.

Section 3. Software Installation and Initialization

3.1 Contents

3.2 Overview 41

3.3 ICS08GP Software Components. 41
3.3.1 WinIDE Editor 42
3.3.2 CASM08W Assembler 42
3.3.3 ICS08GPW In-Circuit Simulator 43
3.3.4 ICD08SW In-Circuit Debugger (ICD). 43
3.3.5 PROG08SW FLASH Programmer 43

3.4 Installing the ICS08GP Software 44
3.4.1 Installation Steps 44
3.4.2 Starting the ICS08GP Software 45

3.4.3 Pod-to-Host Communication 45

3.2 Overview

This chapter summarizes and explains how to install and initialize the ICS08GP software.

3.3 ICS08GP Software Components

The ICS08GP software consists of these components:

- WINIDE.EXE — Windows integrated development environment editor
- CASM08W.EXE — 68HC08 cross assembler
- ICS08GPW.EXE — In-circuit simulator optimized for the HC08 Family of Motorola microcontrollers
- ICD08SW.EXE — Real-time debugger and emulator
- PROG08SW.EXE — FLASH memory programmer

3.3.1 WinIDE Editor

The WinIDE editor is a text-editing application that allows use of several different programs from within a single development environment. Use the WinIDE editor to edit source code, launch a variety of compatible assemblers, compilers, debuggers, or programmers, and configure the environment to read and display errors from such programs.

If error detection options are selected in the **Environment Settings** dialog, the WinIDE editor will highlight errors in the source code and display the error messages from the compiler or assembler in the editor.

Because the WinIDE editor is modular, the user may, for example, choose to substitute a third-party C compiler or other assembler for the CASM08W cross assembler provided in the toolkit.

3.3.2 CASM08W Assembler

CASM08W is a cross assembler that creates Motorola S19 object files and MAP files from assembly files containing 68HC08 instructions.

To debug source code in the simulator or debugger code window, load compatible source-level map files. CASM08W produces such map files as an output by default.

The CASM08W assembler supports all 68HC08 instructions and addressing modes. It can produce .S19 object files, .MAP files, and .LST absolute listing files. The listing files can be configured to show cycle counts.

The assembler also supports macros and conditional assembly. **Section 5. CASM08W Assembler Interface** provides full information about the assembler options and how to use them.

3.3.3 ICS08GPW In-Circuit Simulator

ICS08GPW is an in-circuit simulator for HC908GP series microcontrollers that can get inputs and outputs (I/O) for the device when the external M68ICS08GP hardware pod (for example, the assembled ICS08GP20 board and SPGMR serial programmer) is attached to the host computer. I/O from a target board can be used by attaching the pod to the target through the extension cable that comes with the toolkit.

The simulator can also work in stand-alone mode without the pod attached to the host computer. In this case, simulator inputs can be specified using the INPUTx commands.

Start or move to the ICS08GPW in-circuit simulator software from the WinIDE editor. The ICS08GPW software also can be started using standard Windows techniques and run independently of the WinIDE editor.

The ICS08GPW simulator accepts standard Motorola S19 object code files as input for object code simulation and debugging. When using a third-party assembly or C language compiler, the compiler must be capable of producing source-level MAP files to allow source-level debugging.

3.3.4 ICD08SW In-Circuit Debugger (ICD)

ICD08SW allows limited real-time debugging of the 68HC908GP20 MCU. Unlike the simulator, the ICD allows reading, writing, and controlling execution of the actual processor. Code can be loaded into RAM or FLASH memory and executed in real time or in single steps. For debugging in RAM, multiple software breakpoints are available. For debugging in FLASH memory, one hardware breakpoint is available.

3.3.5 PROG08SW FLASH Programmer

PROG08SW allows erasing, programming, and verification of the MCU's FLASH memory. Individual bytes may be programmed, or an .S19 object file may be used as the source.

3.4 Installing the ICS08GP Software

The ICS08GP software is supplied on three 3.5-inch diskettes. Diskette 1 contains a setup program that automatically installs the software into the host PC's hard drive.

3.4.1 Installation Steps

To install the software on the host computer's hard drive, follow these steps:

1. Insert diskette 1 into the 3.5-inch disk drive. For Windows 95: From the **Start** menu, select the **Run** option.
2. In the **Run** dialog, enter **Setup** or click the BROWSE button to select a different drive and/or directory and press **OK**.
3. In the ICS08GP setup wizard, follow the instructions appearing on the screen.

Table 3-1 lists the files and directories required to control the ICS08GP program modules.

Table 3-1. ICS08GP Software Files

Directory	Filename	Description
CASM08W	CASM08W.EXE CASM08W.HLP	Windows cross assembler for 68HC08 Help for CASM08W
ICD08SW	ICD08SW.EXE ICD08SW.HLP	Windows in-circuit debugger Help for ICD08SW
ICS08GPW	ICS08GPW.EXE ICS08GPW.HLP	Windows in-circuit simulator Help for ICS08GPW
PROG08SW	PROG08SW.EXE PROG08SW.HLP	Windows FLASH programmer Help for PROG08SW
WINIDE	WINIDE.EXE WINIDE.HLP	Windows integrated development environment (WinIDE) Help for WinIDE

3.4.2 Starting the ICS08GP Software

Depending on the operating system used, choose the appropriate method for starting the WinIDE software:

- From the Windows 95 **Start** menu, select the WINIDE and/or ICS08GPW icon(s).

The other ICS08GP programs may be started alone or from within the WinIDE editor. If CASM08W is started alone, a list of command line options appears.

The first time an attempt is made to connect to the pod after installing the ICS08GPW simulator software, the user is prompted to select the chip from the **Pick Device** dialog (**Figure 3-1**):

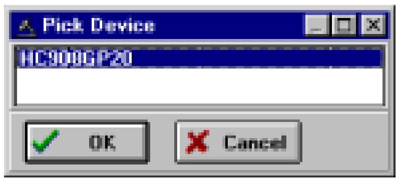
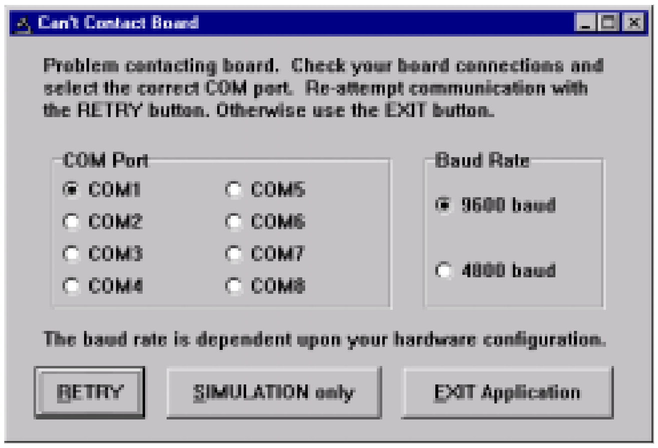


Figure 3-1. Pick Device Dialog Window

3.4.3 Pod-to-Host Communication

When the ICS08GP software is started, it attempts to communicate with the pod using the specified COM port, baud rate, and default parameters. When connection with the pod is established, the status bar contains the message `Contact with pod established`.

If the pod is not installed, or if the software cannot establish communications with the pod through the specified COM port, the `Can't Contact Board` dialog appears (**Figure 3-2**), with options for changing the COM port or baud rate and retrying the connection or running the simulator in stand-alone mode (with no input or output from the pod).



Problem contacting board. Check your board connections and select the correct COM port. Re-attempt communication with the RETRY button. Otherwise use the EXIT button.

COM Port: [✓ COM1] COM2 COM3 COM4 COM5 COM6 COM7 COM8

Baud Rate: [✓ 9600 baud] 4800 baud

The baud rate is dependent upon your hardware configuration.

Figure 3-2. Can't Contact Board Dialog Window

NOTE: *The COM port assignment defaults to COM 1 unless another port is specified in the startup command.*

Section 4. WinIDE User Interface

4.1 Contents

4.2 Overview . . . 48

4.3 Windows Integrated Development Environment . . . 49

4.4 WinIDE Main Window . . . 50
4.4.1 Main Window Functions . . . 50
4.4.2 Main Window Components . . . 50

4.5 Getting Started . . . 52
4.5.1 Prerequisites for Starting the WinIDE Editor . . . 52
4.5.2 Starting the WinIDE Editor . . . 52
4.5.3 Opening Source Files . . . 52

4.6 Navigating in the WinIDE Editor . . . 53
4.6.1 Using Markers . . . 54

4.7 Command-Line Parameters . . . 55

4.8 WinIDE Toolbar . . . 56

4.9 WinIDE Menus . . . 58

4.10 WinIDE File Options . . . 59
4.10.1 New File . . . 60
4.10.2 Open File . . . 60
4.10.3 Save File . . . 61
4.10.4 Save File As . . . 61
4.10.5 Close File . . . 61
4.10.6 Print File . . . 62
4.10.7 Print Setup . . . 62
4.10.8 Exit . . . 62

4.11 WinIDE Edit Options 63
4.11.1 Undo. 63
4.11.2 Redo 64
4.11.3 Cut 64
4.11.4 Copy 65
4.11.5 Paste 65
4.11.6 Delete 65
4.11.7 Select All 65

4.12 WinIDE Environment Options 66
4.12.1 Open Project. 67
4.12.2 Save Project 67
4.12.3 Save Project As 68
4.12.4 Close/New Project 68
4.12.5 Setup Environment. 68
4.12.5.1 General Environment Tab 69
4.12.5.2 General Editor Tab 71
4.12.5.3 Assembler/Compiler Tab 73
4.12.5.4 Executable Tabs: EXE 1-3 78
4.12.6 Setup Fonts. 80

4.13 WinIDE Search Options 81
4.13.1 Find 82
4.13.2 Replace. 83
4.13.3 Find Next 83
4.13.4 Go to Line 84

4.14 WinIDE Window Options. 84
4.14.1 Cascade 85
4.14.2 Tile 86
4.14.3 Arrange Icons. 87
4.14.4 Minimize All 88
4.14.5 Split 89

4.2 Overview

This chapter is an overview of the WinIDE windows, menus, toolbars, dialogs, options, and procedures for using each of them.

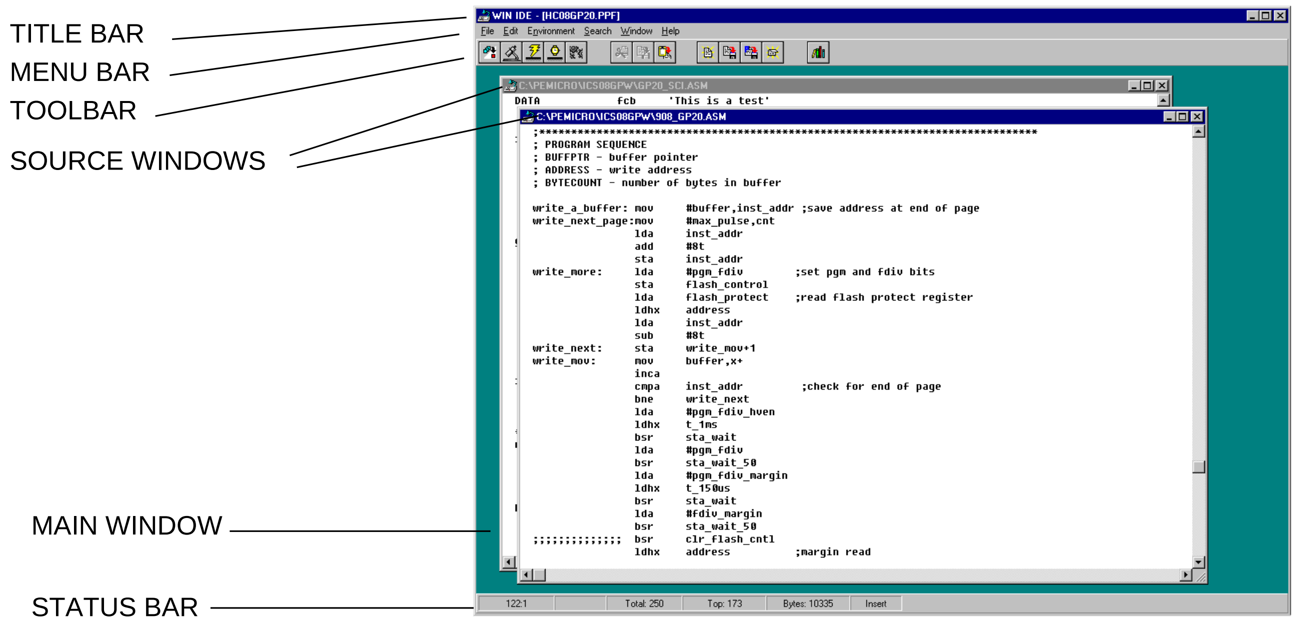
4.3 Windows Integrated Development Environment

The Windows integrated development environment (WinIDE editor) is a graphical interface for editing, compiling, assembling, and running these external ICS08 programs:

- CASM08W assembler
- ICS08GPW in-circuit simulator
- PROG8SW FLASH programmer
- ICD08SW real-time in-circuit debugger

The WinIDE interface consists of standard Windows title and menu bars, a WinIDE toolbar, a main window containing any open source or project file windows, and a status bar.

The WinIDE components are labeled in **Figure 4-1** and described in **4.4.2 Main Window Components**.



```
DATA            fcb     'This is a test'
```

```
;********************************************************************************
; PROGRAM SEQUENCE
; BUFFPTR - buffer pointer
; ADDRESS - write address
; BYTECOUNT - number of bytes in buffer

write_a_buffer: mov     #buffer,inst_addr ;save address at end of page
write_next_page:mov     #max_pulse,cnt
                lda     inst_addr
                add     #8t
                sta     inst_addr
write_more:     lda     #pgm_fdiv        ;set pgm and fdiv bits
                sta     flash_control
                lda     flash_protect    ;read flash protect register
                ldhx    address
                lda     inst_addr
                sub     #8t
write_next:     sta     write_mov+1
write_mov:      mov     buffer,x+
                inca
                cmpa    inst_addr         ;check for end of page
                bne     write_next
                lda     #pgm_fdiv_hven
                ldhx    t_1ms
                bsr     sta_wait
                lda     #pgm_fdiv
                bsr     sta_wait_50
                lda     #pgm_fdiv_margin
                ldhx    t_150us
                bsr     sta_wait
                lda     #fdiv_margin
                bsr     sta_wait_50
;;;;;;;;;;;;;;  bsr     clr_flash_cntl
                ldhx    address          ;margin read
```

Figure 4-1. WinIDE Window Components

4.4 WinIDE Main Window

4.4.1 Main Window Functions

When first starting the WinIDE editor, the Main window opens without any source or project files. As source files are opened or a project is created, they appear as subordinate windows in the Main window. Using standard Windows techniques and the WinIDE window menu options, subordinate windows can be moved, sized, and arranged.

Use the WinIDE Main window to:

- Open, create, edit, save, or print source (*.ASM, *.LST, *.MAP, and *.S19) or project (*.PPF) file.
- Configure the desktop and environment settings for the editor, assembler, compiler, debugger, and other programs.
- Launch the in-circuit simulator, compiler, debugger, programmer, or another program.

4.4.2 Main Window Components

Figure 4-1 shows how the WinIDE Main window might look during a typical editing project and labels the standard window components:

- Title bar — The title bar appears at the top edge of the Main window and contains:
  - Application title
  - Name of the currently open project file
  - Windows control buttons for closing, minimizing, or maximizing the window
- Menu bar — The menu bar appears immediately below the title bar and contains the names of the WinIDE menus.

- Toolbar — The WinIDE toolbar appears just below the menu bar and contains shortcut buttons for frequently used menu options.
- Main window — The Main window area is the inside portion of the Main window which contains the open subordinate windows that can be resized, repositioned, minimized, or maximized using standard Windows techniques or Window menu options.
- Status Bar — The status bar (**Figure 4-2**) appears along the bottom edge of the Main window and contains a number of fields (depending on the project) that show:
  – Source-file line and column numbers of the blinking insertion point cursor
  – System status or progress of the current window; for example, when the window is edited, the status will be modified
  – Total number of lines in the active window
  – Top — Current line position in the file of the top of the active window
  – Bytes — Displays the total number of bytes in the active window
  – Insert/Overwrite mode — Indicates the current typing mode

The status fields expand and contract as client area contents change and files become active.

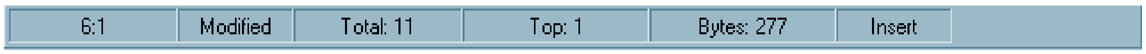


Figure 4-2. WinIDE Status Bar

4.5 Getting Started

4.5.1 Prerequisites for Starting the WinIDE Editor

Before starting the WinIDE editor, the Windows operating environment must be running and the ICS08GP software package must be installed on the host computer.

NOTE: *Remember that for the M68ICS08GP in-circuit simulator to run in simulation mode, the asynchronous communications cable must connect the M68ICS08GP hardware pod to the host computer, and the power to the pod must be on.*

4.5.2 Starting the WinIDE Editor

To start the editor, select the WinIDE icon by selecting the ICS08GPW PROGRAM GROUP icon from the Windows 95 **Start** menu.

4.5.3 Opening Source Files

When the WinIDE editor opens, the Main window is empty. To build the environment for the project, choose the **Open** option from the **File** menu (or click the FILE button on the WinIDE toolbar). In the **Open File** dialog, choose the files that will make up the project:

1. Select the drive containing the files from the **Drives** list.
2. Select the directory folder containing the files from the **Folders** list.
3. Use the **Filename** text box to specify a filename or a wildcard/ extension to filter the list of filenames (or choose a file type from the **List files of type** list). The default file type is .ASM, but these are additional choices:

   *.c — Source code files

   *.lst — Listing files

   *.txt — Text files

   . — All files

To add more filter types, change the environment settings with the **General Editor** tab in the **Main** menu.

When all of the project files have been selected, click the OK button to open the files in the WinIDE Main window.

4.6 Navigating in the WinIDE Editor

To navigate among the several subwindows in which the project files are displayed in the WinIDE Main window:

1. Choose the subordinate window's filename from the Window menu or click on the file's title bar to bring it to the front of the cascaded stack.
2. When using a large screen or a few project files, choose the **Tile** option from the Window menu to lay out all of the subwindows so that all are visible. Or, choose the **Cascade** option to arrange all windows so that only the top window is entirely visible.
3. Regardless of how the windows are arranged, the title bars of all windows are visible.

To move between the WinIDE editor and the external ICS08GP programs, use the toolbar buttons or hotkeys shown in **Table 4-1**. To switch back to the editor from a program, click the program's BACK TO EDITOR toolbar button.

For a complete description of all the WinIDE toolbar buttons, see **Table 4-3**

Table 4-1. Navigating Between External Programs

Switch To	Toolbar Button	EXE Tab	Hotkey
CASM08W assembler	▣	—	F4
ICS08GPW simulator	🔨	EXE1	F6
PROG08SW programmer	⚡	EXE2	F7
ICD08SW debugger	⏱	EXE3	F8

4.6.1 Using Markers

Markers provide a convenient way to mark multiple points in a file for navigating among frequently visited locations while editing. As many as 10 markers can be set in source files in the WinIDE editor. A marker appears in the file as a small button labeled with the marker number.

When the project is saved, the WinIDE editor saves the markers for all open edit files as well, so that when the project is opened again, the markers are still set.

To set a marker anywhere in the file:

1. Place the cursor on the line where the marker should be.
2. Press CNTL + SHIFT + N, where N is a value from 0 to 9 indicating the marker number. A marker appears at the far left of the line.

To move to a marker, press CNTL + N, where N is denotes a marker number between 0 and 9. This feature is useful when editing a large file.

Markers can also be set, changed, navigated to, or cleared using options on the **Edit Shortcut** menu (**Figure 4-3**). Open the **Edit Shortcut** menu by clicking the right mouse button in any edit window.

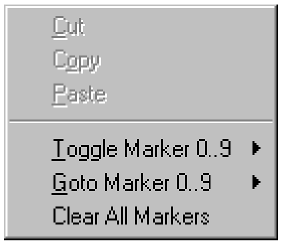


Figure 4-3. WinIDE Edit Shortcut Menu

To set or clear a marker using the **Edit Shortcut** menu options:

1. With the cursor in any editing window, click the right mouse button to open the **Shortcut** menu.
2. Position the cursor on the line where the marker should appear. Click the right mouse button to display the **Shortcut** menu.
3. Click the **Toggle Marker 0–9** option to open the list of markers.
4. Click once on the marker to toggle. When the marker number is checked, it is toggled on; when the marker number is unchecked, it is toggled off.

To move to a marker number using the **Shortcut** menu options:

1. With the cursor anywhere in the edit file, click the right mouse button to open the **Edit Shortcut** menu.
2. Click on the **Go To Marker 0–9** option to open the **Marker** submenu (**Figure 4-4**), and choose the marker number to move to.

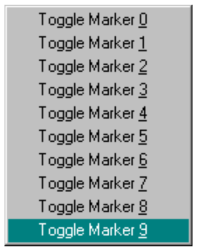


Figure 4-4. Marker Submenu

Many WinIDE menu options can be executed using either keyboard commands or toolbar buttons. For example, to move to a marker, press the CTRL + SHIFT + N key combination, where N is the marker number.

4.7 Command-Line Parameters

The WinIDE editor lets the user specify command-line options to pass to each executable program. The name of the currently edited file, or some derivative thereof, can be passed within these options. To pass the current filename, specify a parameter `%FILE%`. The WinIDE editor substitutes this string with the current filename at execution time. The extension of the passed filename can also be changed by specifying it within the `%FILE%` parameter. For example, to specify an .S19 extension on the current filename, the user would specify a `%FILE.S19%` parameter.

As an example, **Table 4-2** shows the parameters if the current filename being edited is MYPDA.ASM.

Table 4-2. WinIDE Editor Substitution Parameters

Parameters Specified	Parameters Passed to Program
`%FILE%   S   L   D`	`MYPDA.ASM   S   L   D`
`%FILE.S19% 1 @2`	`MYPDA.S19   1   @2`

Although it is by default, the currently edited filename that is used in the `%FILE%` parameter substitution, the environment can be configured always to pass the same filename. Do this by checking the **Main File** option in the **Environment Settings** dialog's **General Options** tab. This technique is useful when passing a specific filename to the external program without regard to what is being edited.

4.8 WinIDE Toolbar

The WinIDE toolbar (**Figure 4-5**) provides a number of convenient shortcut buttons that duplicate the function of the most frequently used menu options. A tool tip or label pops up when the mouse button lingers over a toolbar button, identifying the button's function.

Table 4-3 identifies and describes the WinIDE toolbar buttons and hotkeys.

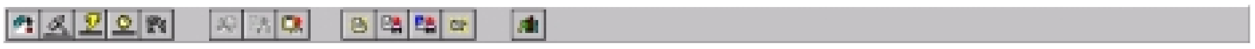


Figure 4-5. WinIDE Toolbar

Table 4-3. WinIDE Toolbar Buttons

Icon	Button Label	Button Function
	External program 1 hotkey F6	Call the external program 1 specified in the **Environment Settings** dialog's **EXE 1** tab. This could be a third-party assembler, debugger, or compiler. Default: ICS08GPW
	External program 2 hotkey F7	Call the external program 2 specified in the **Environment Settings** dialog's **EXE 2** tab. This could be a third-party assembler, debugger, or compiler. Default: PROG08SW
	External program 3 hotkey F8	Call the external program 3 specified in the **Environment Settings** dialog's **EXE 3** tab. This could be a third-party assembler, debugger, or compiler. Default: ICD08SW
	External program 4 hotkey F9	Call the external program 4 specified in the **Environment Settings** dialog's **EXE 4** tab. This could be a third-party assembler, debugger, or compiler.
	Assemble/compile file hotkey F4	Assemble or compile the active source window using CASM08W.
	Cut	Cut the selected text from the active source window. This button is a shortcut for the **Edit-Cut** menu option.
	Copy	Copy the selected text in the active source window to the Windows clipboard. This button is a shortcut for the **Edit-Copy** menu option.
	Paste	Paste the contents of the Windows clipboard at the insertion-point location in the active source window.This button is a shortcut for the **Edit-Paste** menu option.
	Open file	Close the active source window. This button is a shortcut for the **File-Open** menu option.
	Save file	Save the file in the active source window. This button is a shortcut for the **File-Save** menu option.
	Save project all files and setup	Save the active project. This button is a shortcut for the **Environment-Save Project As** menu option.
	Close file	Close the active source window. This button is a shortcut for the **File-Close** menu option.
	View register files	Set up peripheral registers interactively.

4.9 WinIDE Menus

Table 4-4 summarizes WinIDE menu titles and options.

Table 4-4. WinIDE Menus and Options Summary

Menu Title	Option	Description
File	New file	Open a new file window (no name).
	Open file	Display the **Open File** dialog to choose a file to open.
	Save file	Save the current file.
	Save file as	Open the **Save As** dialog to choose a directory and filename in which to save the current file.
	Close file	Close the current file.
	Print	Open the **Print** dialog to print the current file.
	Print setup	Open the **Print Setup** dialog to choose printer options.
	Exit	Close the WinIDE editor.
Edit	Undo	Undo the last action.
	Redo	Redo the last action.
	Cut	Cut the selection to the clipboard.
	Copy	Copy the selection to the clipboard.
	Paste	Paste the contents of the clipboard.
	Delete	Delete the selection.
	Select all	Select all text in the current window.
Environment	Open project	Open the **Specify Project File** to **Open** dialog.
	Save project	Save the current project.
	Save project as	Open the **Specify Project File** to **Save** dialog.
	Close/new project	Close the current project file or open a new project file if no current file.
	Set up environment	Open the **Environment Settings** dialog to change settings for: – General Environment, External EXE 2 – General Editor, External EXE 3 – Assembler/Compiler, External EXE 4 – External EXE 1
	Set up font	Open the **Font** dialog to specify font options for text in the current file.
Search	Find	Open the **Find** dialog to enter a search string.
	Replace	Open the **Replace** dialog to enter a search and replacement string.
	Find next	Go to the next occurrence of the search string.
	Go to line	Open the **Go to Line Number** dialog and enter a line number to go to in the current file.

Table 4-4. WinIDE Menus and Options Summary (Continued)

Menu Title	Option	Description
Window	Cascade	Cascade open windows with active window on top.
	Tile	Tile open windows with active window on top.
	Arrange icons	Arrange minimized window icons along the bottom edge of the main window.
	Minimize all	Minimize all open windows.
	Split	Toggle a split window in the active file.
	Windows by name	Itemize the open and minimized windows by name in order of opening.
Help	Contents	Opens the WinIDE Help Contents Page of the Help File.
	About	Displays the WinIDE About Window.

4.10 WinIDE File Options

This section describes the WinIDE **File** menu options for managing and printing source files or exiting the WinIDE editor.

To select a **File** option, click once on the **File** menu title to open the **File** menu (**Figure 4-6**). Click on an option to perform the operation. Use accelerator or shortcut keystrokes to execute the option.

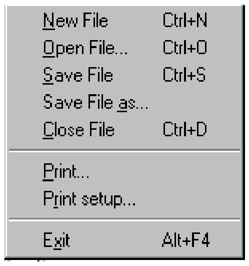


Figure 4-6. WinIDE File Menu

4.10.1 New File

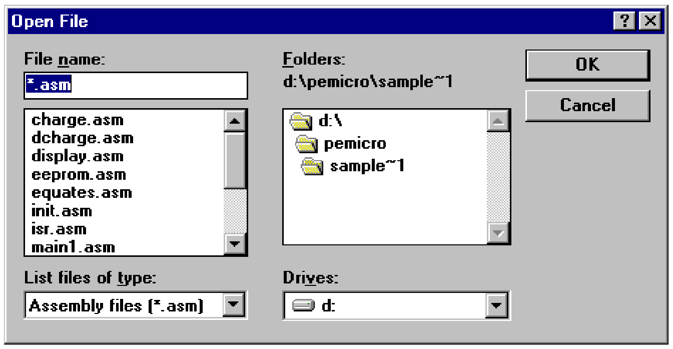
Choose **New File** from the **File** menu to open a new client window in the WinIDE Main window. The title of the new window in the title bar defaults to `[NONAME#]`, where # reflects the number of new source windows created during this session. If there is an active project, the project name appears in the title bar. If there is no project, `[No Project]` precedes the window name.

Use this new window to enter source code. When the contents of this window are saved, the WinIDE editor prompts for a new filename. This new filename replaces the `[NONAME#]` in the title bar.

Alternatives: Type CTRL + N or click the NEW toolbar button. This is the keyboard equivalent to choosing the **File-New File** menu option.

4.10.2 Open File

Choose **Open File** from the **File** menu to open the **Open File** dialog window (**Figure 4-7**) and choose an existing filename, file type, directory, and network (if applicable) to open.


Open File

File name: *.asm

charge.asm
dcharge.asm
display.asm
eeprom.asm
equates.asm
init.asm
isr.asm
main1.asm

List files of type: Assembly files (*.asm)

Folders: d:\pemicro\sample~1

d:\
pemicro
sample~1

Drives: d:

Figure 4-7. Open File Dialog Window

Each file opens in its own client window within the main WinIDE window.

Alternatives: Type CTRL + O or click the OPEN button on the toolbar. This is the keyboard equivalent to choosing the **File-Open File** menu option.

4.10.3 Save File

Choose **Save File** from the **File** menu to save the file in the active source window.

- If the file is saved for the first time (that is, it has not yet been named), the **Save As** dialog appears. Enter a new filename for the file and accept the current file type, directory or folder, and drive, or choose new options. Press the OK button to save the file to the selected drive/directory.
- If the file has been saved previously (and has a name), the file is saved with the filename in the directory and drive previously specified, and the source window remains open.

Alternatives: Type CTRL + S or click the SAVE button on the toolbar. This is the keyboard equivalent to choosing the **File-Save File** menu option.

4.10.4 Save File As

Choose **Save File As** from the **File** menu to save the contents of the active source window and assign a new filename. The **Save As** dialog opens. Enter a new filename in the **File Name** field and click the OK button to save the file and return to the source window.

To save the file with the name of an existing file, select the filename in the **File Name** list, and click the OK button. A confirmation dialog will ask for confirmation to overwrite the existing file.

4.10.5 Close File

Choose **Close File** from the **File** menu to close the file in the active source window.

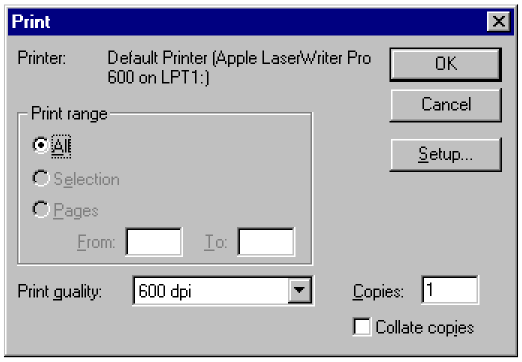
If the **Give user option to save each file** option in the **General Environment** tab in the **Environment Settings** dialog is chosen, the **Information** dialog will display, reminding the user to save changes to the .ASM file.

Alternatives: Type Ctrl + D or click the CLOSE toolbar button. This is the keyboard equivalent to choosing the **File - Close File** menu option.

4.10.6 Print File

Choose **Print** from the **File** menu to open the **Print** dialog (**Figure 4-8**) and choose options for printing the active source window.

The **Print** dialog for the operating system and printer capabilities opens with options for **Print range**, **Print quality**, and open the **Print Setup** dialog to change printer settings.



Printer: Default Printer (Apple LaserWriter Pro 600 on LPT1:)

Print range: [✓ All] Selection Pages

Print quality: 600 dpi

Copies: 1

Collate copies

Figure 4-8. Print Dialog Window

NOTE: *The **Print** option is active when at least one source window is open. The WinIDE editor disables the option if no window is open.*

4.10.7 Print Setup

Choose the **Print Setup** option from the **File** menu to open the **Print Setup** dialog for the operating system and printer. Use this dialog to choose the printer, page orientation, paper size, and other printer options.

4.10.8 Exit

Choose the **Exit** option from the **File** menu to close the editor. If a project or source window is open, the editor closes the files and exits.

Alternatives: Type ALT + F4. This is the keyboard equivalent to choosing the **File-Exit** menu option.

4.11 WinIDE Edit Options

This section describes the WinIDE **Edit** menu options for creating or editing source file contents.

To perform an edit operation, click once on the **Edit** menu title to open the **Edit** menu (**Figure 4-9**). Click on an option to perform the operation.

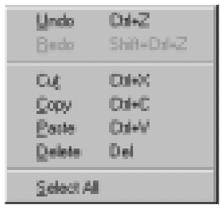


Figure 4-9. WinIDE Edit Menu

4.11.1 Undo

Choose **Undo** to undo or reverse the last action or change made in the active source window.

Changes made to the contents of the window (and that are undoable or reversible) are saved in an undo stack, where they accumulate up to a maximum of 20 instances. Changes can be reversed in descending order of the sequence in which they were made. If no more changes remain in the stack, the **Undo** option is disabled.

Reversible actions are local to each source window. Commands that are not reversible do not contribute to the undo stack. The user cannot, for example, undo the command to open a new window using the UNDO command.

Alternatives: Type CTRL + Z. This is the keyboard equivalent to selecting the **Edit-Undo** menu option.

4.11.2 Redo

Choose **Redo** to restore the most recently undone action in the active window.

The **Redo** option restores actions undone or reversed by the **Undo** option, in ascending order, that is, last action first. Reversible changes to the window's contents accumulate in the window's undo stack. Once a change has been reversed using the **Undo** option, the change can be reversed, using the **Redo** option. When no more changes remain (that is, the top of the redo stack is reached) the **Redo** option is disabled.

Some commands are not reversible. They do not contribute to the undo stack and therefore cannot be redone. For instance, since reversible actions are local to each source window, opening a new window is an action that cannot be undone using the UNDO command or redone using the REDO command.

NOTE: *The **Redo** option is active only if the **Undo** option has been used to modify the contents of the active source window.*

Alternative: Type SHIFT + CTRL + Z. This is the keyboard equivalent to selecting the **Edit-Redo** menu option.

4.11.3 Cut

Choose **Cut** from the **Edit** menu to cut the currently selected text from the active source window and place it on the system clipboard.

NOTE: *The **Cut** option is active only when text in the active source window has been selected.*

Alternative: Type CTRL + X. This is the keyboard equivalent to selecting the **Edit-Cut** menu option.

4.11.4 Copy

Choose **Copy** from the **Edit** menu to copy the selected text from the active source window to the Windows clipboard.

NOTE: *The **Copy** option is available only if text in the active source window has been selected.*

Alternatives: Type CTRL +C or click the COPY toolbar button. This is the keyboard equivalent to selecting the EDIT-COPY menu option.

4.11.5 Paste

Choose **Paste** from the **Edit** menu to paste the contents of the Windows clipboard into the active source window at the insertion-point location.

Alternatives: Type CTRL + V or click the PASTE button on the toolbar. This is the keyboard equivalent to selecting the **Edit - Paste** menu option.

4.11.6 Delete

Choose **Delete** from the **Edit** menu to delete the selected text from the active source window without placing it on the Windows clipboard. Text deleted using the **Delete** option can be restored only by using the **Undo** option.

Alternatives: Press the DELETE key. This is the keyboard equivalent to selecting the **Edit-Delete** menu option.

4.11.7 Select All

Choose **Select All** from the **Edit** menu to select all text in the active source window.

4.12 WinIDE Environment Options

This section describes the WinIDE **Environment** menu options for managing project information and setting up environment and font settings for a project.

Environment settings represent the current environment and configuration information for the WinIDE editor. These settings are stored in the WINIDE.INI file, from which they are loaded each time the editor is started and saved each time the editor is exited.

When the editor is started, the application opens the WINIDE.INI file and reads the project information. If there is an open project, the project file's environment settings are read and used instead. This lets the user have different environment configurations for different projects.

Environment information stored in the WINIDE.INI file includes:

- Its name, if a project is open
- Current font information
- Current source directory and project directory paths
- The preferences and options set in the **Environment Settings** dialog tabs, including:
  - **General Environment** options
  - **General Editor** options
  - Executable options for assembler, debugger, compiler, and programmer

To choose an environment option, click once on the **Environment** menu title (**Figure 4-10**) to open the menu. Click on the option to execute.

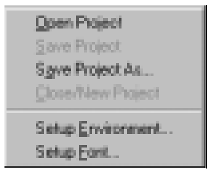


Figure 4-10. WinIDE Environment Menu

Project files have the extension .PPF. They store two kinds of information:

- Environment settings — User settings and WinIDE configuration parameters
- Desktop information — Open edit windows, size, location, and markers

4.12.1 Open Project

Choose **Open Project** from the **Environment** menu to choose the project file in the **Specify project file to open** dialog (**Figure 4-11**).

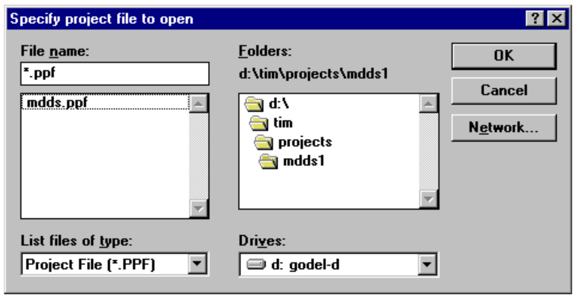


Figure 4-11. Specify Project File to Open Dialog Window

1. Enter the project name in the **File name:** text box or select the project name from the list box below it.
2. Press the OK button to open the new project file or press the CANCEL button to close the dialog without opening a file.

4.12.2 Save Project

Choose **Save Project** from the **Environment** menu to save the current project in the currently specified file and pathname.

4.12.3 Save Project As

Choose **Save Project As** from the **Environment** menu to display the **Specify project file to save** dialog (**Figure 4-12**).

1. Enter the project name in the **File name:** text box or select the project name from the list box below it.
2. Press the OK button to open the new project file or press the CANCEL button to close the dialog without opening a file.

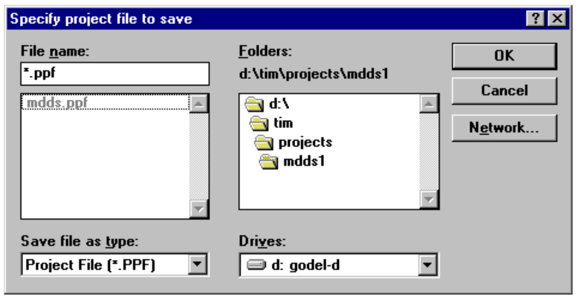


Figure 4-12. Specify Project File to Save Dialog Window

4.12.4 Close/New Project

Choose **Close/New Project** from the **Environment** menu to:

- Close an active current project file
- Open a new project

4.12.5 Setup Environment

Choose **Setup Environment** from the **Environment** menu to display the **Environment Settings** dialog box.

The **Environment Settings** dialog contains these five tabs:

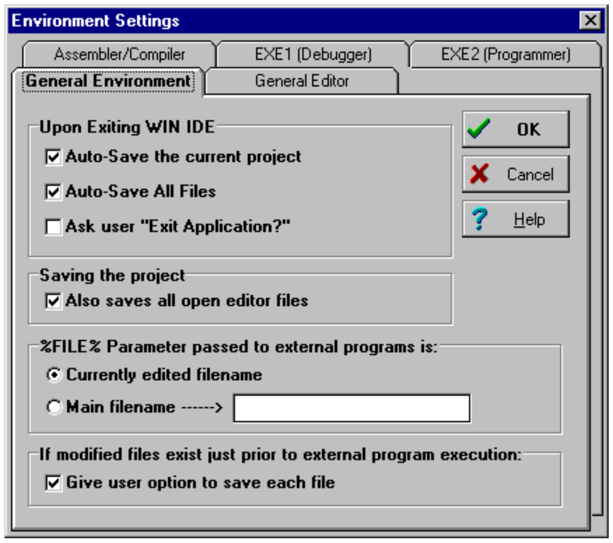
- **General Environment**
- **General Editor**
- **Assembler/Compiler**

- **EXE 1** — Default: ICS08GPW in-circuit simulator
- **EXE 2** — Default: PROG08SW programmer
- **EXE 3** — Default: ICD08SW debugger
- **EXE 4**

In the **Environment Settings** tabs, choose options by marking option buttons (sometimes called radio buttons), check boxes, and entering information in text boxes.

4.12.5.1 General Environment Tab

Click the **General Environment** tab in the **Environment Settings** dialog (**Figure 4-13**) to change options for saving the project files, exiting the WinIDE editor, and storing a filename to be passed to an external program as a parameter.


Environment Settings

Upon Exiting WIN IDE
☑ Auto-Save the current project
☑ Auto-Save All Files
☐ Ask user "Exit Application?"

Saving the project
☑ Also saves all open editor files

%FILE% Parameter passed to external programs is:
● Currently edited filename
○ Main filename ------>

If modified files exist just prior to external program execution:
☑ Give user option to save each file

Figure 4-13. Environment Settings Dialog General Environment Tab

NOTE: *Clicking the* OK *button on any tab saves all changes made in the* ***Environment Settings*** *dialog and closes the dialog.*

The **General Environment** tab offers these options:

- **Upon Exiting the WinIDE Editor**
  - **Auto-Save the Current Project** — Select this option to save the currently open project automatically, with the file extension .PPF, without prompting. The editor saves all currently open files with the current project. If this option is not selected, the editor prompts to save the open project when exiting. This setting only has an effect if a project is open when exiting.
  - **Auto-Save All Files** — Select this option to save all open editor files automatically, without prompting, when exiting. If this option is not selected, the editor will prompt to save open files when exiting.
  - **Ask user Exit Application** — Select this option to display an `Exit Application` confirmation message when exiting. If this option is not selected, the editor will close without asking for confirmation when the user chooses the **Exit** option from the **File** menu.
- **Saving the Project**
  - **Also save all open editor files** — Select this option to save all open editor files whenever saving the project file. If this option is not selected, project/environment information is written to the project files, but editor files are not saved when the **Save Project** option is chosen from the **Environment** menu.
- **%FILE% Parameter passed to executable programs is** — The `%FILE%` parameter specifies what is passed on the command line in place of the `%FILE%` string. The `%FILE%` string can be specified as a command-line parameter for executable programs launched from within the WinIDE editor.
  - **Currently edited filename** — Select this option to use the name of the current active file (the window with focus) as the `%FILE%` parameter substitution.
  - **Main Filename** — Select this option to use the filename in the **Main** filename edit box as the `%FILE%` parameter substitution.

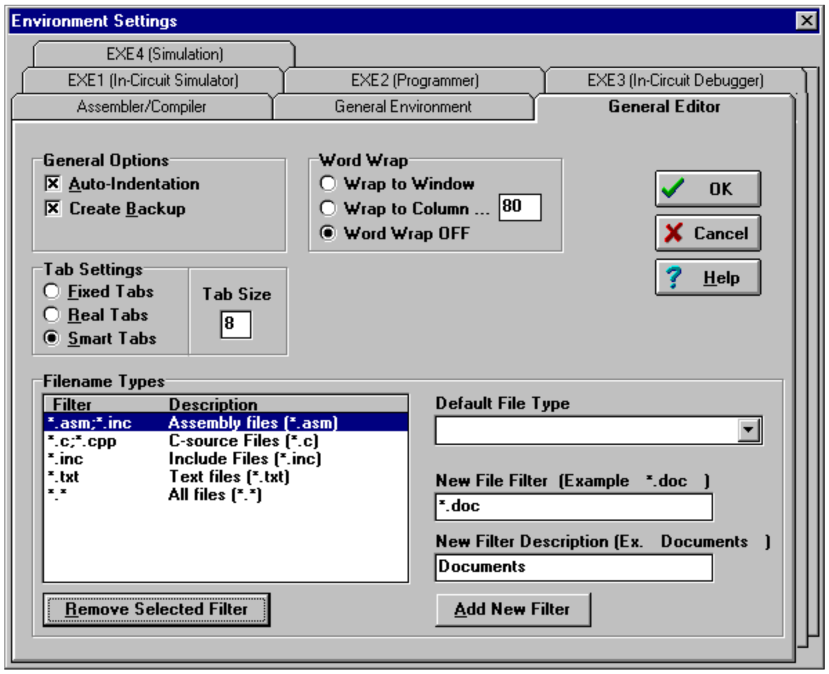
NOTE: *When using include files, the user must enter the full pathname of the file containing the included files in the **Main** filename edit box.*

- **If Modified files exist just prior to external program execution** — All executable programs which can be launched from the WinIDE editor offer the option to save all open editor files before the executable is launched.
  - **Give user option to save each file** — Select this option to be prompted to save each modified file before the external program is launched. If this option is not selected, the external program runs without asking for confirmation. The result may be that an external program runs while modified files exist in the editing environment, a circumstance that may be undesirable and lead to incorrect results.

4.12.5.2 General Editor Tab

Click the **General Editor** tab in the **Environment Settings** dialog (**Figure 4-14**) to bring the **General Editor** tab to the front. Use the **General Editor** tab to change editing options such as indentation, word wrap, tab settings, and filename types.

NOTE: *To change font options, choose the* ***Setup Fonts*** *option from the* ***Environment*** *menu.*



General Options
[✓] Auto-Indentation
[✓] Create Backup

Word Wrap
○ Wrap to Window
○ Wrap to Column ... 80
● Word Wrap OFF

Tab Settings
○ Fixed Tabs
○ Real Tabs
● Smart Tabs

Tab Size 8

Filename Types

Filter	Description
.asm;.inc	Assembly files (*.asm)
.c;.cpp	C-source Files (*.c)
*.inc	Include Files (*.inc)
*.txt	Text files (*.txt)
.	All files (*.*)

Default File Type

New File Filter (Example *.doc)
*.doc

New Filter Description (Ex. Documents)
Documents

Figure 4-14. Environment Settings Dialog: General Editor Tab

- **General Options**
  - **Auto-Indentation** — Select this option to place the cursor in the column of the first non-space character of the previous line when the ENTER key is pressed. If this option is not checked, the cursor goes to the first column. For example, if the current line begins with two tab spaces, pressing the ENTER key will begin the next line with two tab spaces, aligning the new line under the first text of the previous line.
  - **Create Backup** — Select this option to create a backup file whenever a file is saved. The WinIDE editor will copy the current disk version of the file (the last save) to a file of the same name with the .BAK extension, then save the current edited copy over the editing filename. The default (and recommended) setting for this option is on, giving the user the option to return or review the previous version of the file. If this option is not selected, the currently edited file will be saved, but no backup will be made.
- **Word Wrap**
  - **Wrap to Window** — Select this option to have the cursor wrap to the left when it reaches the far right side of the window. This lets the user see all the text in the file without scrolling the line. If this option is not selected, text wraps only when the ENTER key is pressed.
  - **Wrap to Column** — Select this option to wrap text to the left side when the cursor reaches a specified column. This lets the user see all the text in the file without scrolling the line. Set the column number at which text wrapping should occur in the edit box to the right of this option.
  - **Word Wrap OFF** — Select this option to turn text wrapping off. To view or edit text, which does not fit horizontally in the window, use the scroll controls. In general, this option should be on when writing or editing code.
- **Tab Settings**
  - **Fixed Tabs** — Select this option to use spaces to emulate tabs. Pressing the tab key inserts a number of spaces to bring the cursor to the position of the next tab stop. Changing the tab size affects only future tab spacings. Past tabs remain unchanged.

- **Real Tabs** — Select this option to use actual tab characters. Pressing the tab key insets a tab character. The tab character is displayed as a number of spaces determined by the tab size, but is really a tab character. Changing the tab size affects the display of all tabs in the file, present and future.
- **Smart Tabs** — Select this option to enable smart tabs:
  i. If the previous line contains text, pressing the TAB key advances the cursor to the same column as the beginning of the next character group on the previous line.
  ii. If the previous line does not contain text, smart tabs behave as fixed tabs.
- **Tab Size** — Enter the number of spaces in a tab. This setting affects how all tabs operate: fixed, real, or smart tabs. This number is the default display size of all tab characters and the size in spaces of a tab in both fixed and smart modes. If the tab size is N, the tab stops are at 1, N + 1, 2N + 1, 3N + 1, and so on.

• **Filename Types**

- As part of the **Environment Settings/General Editor** options, WinIDE allows a default file type to be set by using the file filter. This is useful when working primarily with one type of file (although a filter may also include a group of extensions). A list of the filter descriptions and their corresponding extensions is displayed to the left. To select a filter, use the **Default File Type** pull-down box on the right to choose the file type. Also define filters and add them to the list by first entering the appropriate information into the boxes for **New File Filter** and **New Filter Description** on the lower right, then selecting the ADD NEW FILTER button. Filters to be removed may be deleted by highlighting a filter and using the REMOVE SELECTED FILTER button.

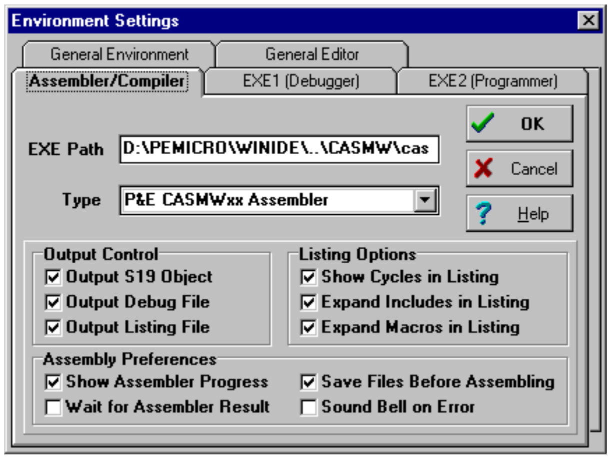
4.12.5.3 Assembler/Compiler Tab

In addition to running an external compiler, other external programs such as third-party programmers, debuggers, or simulators may need to be run. The WinIDE editor allows configuration of as many as three external programs: two general-purpose programs and one compiler. Use the settings on the

Assembler/Compiler tab of the WinIDE **Environment Settings** dialog to set up external programs.

Click the **Assembler/Compiler** tab heading in the **Environment Settings** dialog (**Figure 4-15**) to bring the tab to the front. Use the options on this tab to change the settings and parameters for the assembler or compiler path and type and specify output, listing, and assembly preferences.

- **EXE Path** — Enter the full path and executable name of the compiler in the text box. The extensions EXE/COM/BAT are legal. For a DOS executable or BATch file, create a PIF file to prevent the screen from changing video modes when the executable runs.



EXE Path: D:\PEMICRO\WINIDE\..\CASMW\cas

Type: P&E CASMWxx Assembler

Output Control
- [✓] Output S19 Object
- [✓] Output Debug File
- [✓] Output Listing File

Listing Options
- [✓] Show Cycles in Listing
- [✓] Expand Includes in Listing
- [✓] Expand Macros in Listing

Assembly Preferences
- [✓] Show Assembler Progress
- [ ] Wait for Assembler Result
- [✓] Save Files Before Assembling
- [ ] Sound Bell on Error

Figure 4-15. Environment Settings Dialog: Assembler/Compiler Tab

- **TYPE** — Click on the downward-pointing arrow to the right of the **Type** list box to display the compiler types. Click on the compiler type to select it. The options in the **Assembler/Compiler** tab change according to the compiler type chosen:
  - With the CASM08W compiler selected, a number of compiler options are available.
  - If a different compiler is selected, options allow the user to specify the parameters to pass to the compiler.

- **Output Control** — These options specify the output files that the assembler will create:
  - **Output S19 Object** — Select this option to have the assembler output an S19 object file. The S19 object file contains the compiled instructions from the program assembled. The output S19 file has the same name as the assembly file, but with the .S19 extension. **Appendix A. S-Record Information** gives more information about the S19 file format.
  - **Output Debug File** — Select this option to have the assembler produce a debug .MAP file. The debug .MAP file contains symbol information as well as line number information for source-level debugging from the program assembled. The output debug file has the same name as the assembly file, but with the .MAP extension.
  - **Output Listing File** — Select this option to have the assembler produce a listing file. The listing file shows the source code as well as the object codes that were produced from the assembler. Listing files are useful for debugging as they let the user see exactly where and how the code assembled. The output listing file has the same name as the assembly file, but with the .LST extension.
- **Listing Options** — These options specify how the assembler generates the listing file:
  - **Show Cycles in Listing** — Select this option to include cycle information for each compiled instruction in the listing (.LST) file. View the cycle information to see how long each instruction takes to execute. The cycle count appears to the right of the address, enclosed in brackets.
  - **Expand Includes in Listing** — Select this option to expand all include files into the current listing file. This lets the user view all source files in a main listing file. If this option is not checked, only the `$Include` statement for each included file will be seen, not the source file.
  - **Expand Macros in Listing** — Select this option to expand all macros into the listing file. Each time the macro is used, the listing will show the instructions comprising the macro. If this option is not selected, the user will see only the macro name, not its instructions.

- **Assembly Preferences**
  - **Show Assembler Progress** — Select this option to display a pop-up window showing the current assembly status, including:
    - Pass the assembler is currently on
    - File currently being assembled
    - Line currently being assembled

    If this option is not checked, the user must wait for the assembly result to be displayed on the status bar at the bottom of the Environment window.
  - **Wait for Assembler Result** — Select this option and the **Show Assembler Progress** option to cause a progress window displaying the assembly result to stay up when assembly is done. The assembly result window will remain until it is dismissed by clicking the OK button. In general, do not select this option, as the assembler results are shown in the status bar at the bottom of the WinIDE window.
  - **Save files before Assembling** — Select this option to save all open files to disk before running the assembler. This is important because the assembler/compiler reads the file to be compiled from the disk, not from the open windows in the WinIDE editor. If the file is not saved before assembling it, the assembler will assemble the last saved version. In general, leave this option checked.
  - **Sound Bell on Error** — Select this option to have the assembler beep if it encounters an error.
- **Other Assembler/Compiler** — If the **Other Assembler/Compiler** is chosen from the **Type** list, the WinIDE editor offers these additional options:
  - **Options** — Enter the options to pass to the compiler on the command line. Such options generally consist of switches that instruct the compiler and a filename. Enter the `%FILE%` string in the command line to insert either the current filename or the filename specified in the **Main Filename** option in the **EXE Path** text box of the **General Environment** tab options (**Figure 4-13**).

- **Confirm command line** — Select this option to display a window describing the desired executable and the parameters that should pass to the executable, just before the assembler/compiler is run. This gives the option to cancel the assemble/compile, continue as described, or modify parameters before continuing with the assembly. If this option is not selected, the assembler/compiler runs without prompting to confirm parameters.
- **Recover Error from Compiler** — Select this option to have the WinIDE editor attempt to recover error/success information from the assembler/compiler and open the file with the error line highlighted and displayed in the status bar when an error is encountered. For this feature to work, the **Error Filename** and **Error Format** options must also be set in this tab. If this option is not checked, the WinIDE editor will not look for a compiler result and will not display the results in the status bar.
- **Wait for compiler to finish** — Select this option to have the WinIDE editor disable itself until the compiler terminates. This option must be selected for the editor to attempt to recover error/success information from the assembler/compiler. Further, turning this option on prevents the user from running external programs from the editor that may require compilation or assembly results. If this option is not selected, the editor starts the assembler/compiler and continues letting Windows' multitasking capabilities take care of the program.
- **Save files before Assembling** — Select this option to save all open files to disk before running the assembler. This can be very important since the assembler/compiler reads the file to be compiled from the disk and not from the memory of the WinIDE editor. If the file being assembled isn't saved, the assembler or compiler will assemble the last saved version. For this reason, this option should remain checked.

• **Error Format** — Click the down arrow to the right of the **Error Format** list box to display the list of error formats (**Figure 4-16**). If the WinIDE editor is to attempt to read back an error from a compiler, it must understand the error syntax. This option lets the user select an error format from a list of supported formats. If the **Recover Error from compiler** option is checked, and the filename specified in the **Error**

Filename text box is found, the editor parses that file from end to beginning looking for the error. If the editor finds an error, it opens the file, highlights the error line, and displays the error in the status bar.

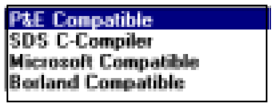


Figure 4-16. Error Format List

- **Error Filename** — Enter the filename to which the editor pipes the compiler/assembler error output. Some compilers provide a switch for piping error output to a file; others require that this be handled manually. As most compilers are DOS-based, a batch file can be created into which to pipe the output. For example:

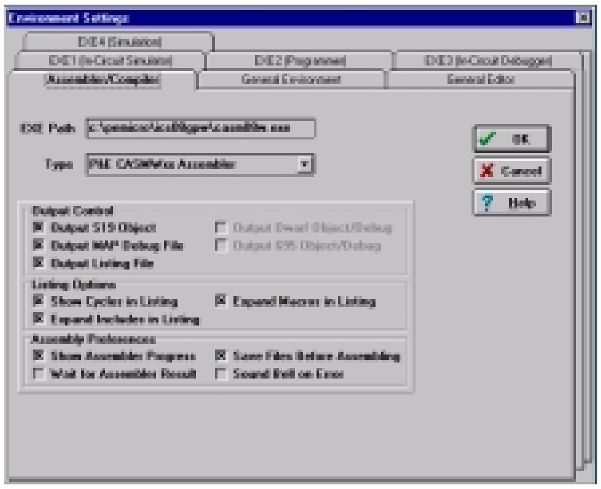
  ```
  COMPILER   OPTIONS   >   ERROR.TXT
  ```

  This batch file creates the file ERROR.TXT and sends the assembler/compiler output to that file. Most C compilers require a batch file to run the compiler through its various steps (compiling, linking), to which the user may add a pipe for error output.

  Once the environment reads this error file, the WinIDE editor displays the results and then deletes the error file. To keep a copy of the file, add such instructions to the batch file.

4.12.5.4 Executable Tabs: EXE 1-3

Choose either the **EXE 1 (in-circuit simulator)**, the **EXE 2 (programmer)**, or the **EXE 3 (debugger)** tab in the **Environment Settings** dialog to bring the tab to the front. Enter options for the general-purpose external programs, for example, ICS08GPW, that will be used with this project. **Figure 4-17** illustrates the tabs. The options are the same for all tabs.



EXE Path: c:\pemicro\ics08gpw\casm08w.exe

Type: P&E CASM08 Assembler

Output Control
- ☒ Output S19 Object
- ☒ Output MAP Debug File
- ☒ Output Listing File
- ☐ Output Dwarf Object/Debug
- ☐ Output ERS Object/Debug

Listing Options
- ☒ Show Cycles in Listing
- ☒ Expand Includes in Listing
- ☒ Expand Macros in Listing

Assembly Preferences
- ☒ Show Assembler Progress
- ☐ Wait for Assembler Result
- ☒ Save Files Before Assembling
- ☐ Sound Bell on Error

Figure 4-17. Environment Settings Dialog: EXE Tabs

- **Type** — Enter a description of the executable type in the **Type** text box. This string will appear in other parts of WinIDE editor. The default for Executable 1 is debugger. For the ICS08GPW, the user may choose to change the **Type** to ICS. This will change the label on this tab and elsewhere in the dialog.
  - **EXE Path** — Enter the full path and executable name of Executable 1 in the **EXE Path** text box. The executable name may have an .EXE, .COM, or .BAT extension. For a DOS-based executable or batch file, the user may choose to create a PIF file to prevent the screen from changing video modes when the file is run.
  - **Options** — Enter the options to be passed to the executable on the command line in the **Options** text box. In general, options will consist of switches that instruct the executable from the command line. Add a filename using the `%FILE%` string. The `%FILE%` string inserts either the currently active filename or the filename specified by the `%FILE%` parameter, set in the `%FILE%` parameters to pass to the external programs field in the **General Environment** tab.
  - **Confirm Command line before running** — Select this option to display a window describing the executable to be run and the parameters which will be passed, just before the assembler/compiler is run. This gives the option to cancel the assemble/compile, continue

as described, or modify parameters before continuing. If this option is not selected, the assembler/compiler will be run without prompting to confirm parameters.

– **Save all files before running** — Select this option to save all open files to disk before running the executable. This is important since external programs that must read the edit file read only the last version saved to disk. In general, always select this option.

4.12.6 Setup Fonts

Select the **Setup Fonts** option in the **Environment** menu to open the **Setup Fonts** dialog (**Figure 4-18**) to change font options in the editor.

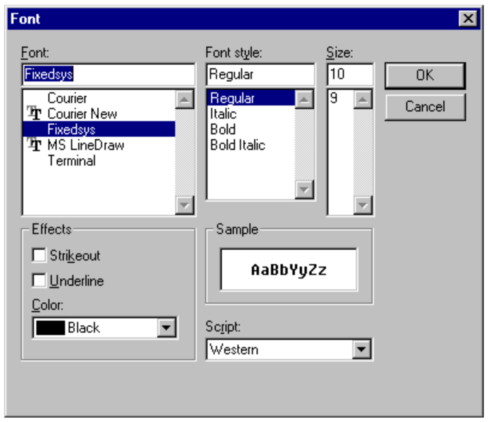


Figure 4-18. Setup Fonts Dialog Window

- **Font** — The **Font** text box displays the name of the current font. To change the current font, select another font name from the **Font** list. Use the scroll arrows if necessary to view all the font choices.
- **Font Style** — The **Font Style** text box displays the name of the current font style. To change the current font style, select another font style name from the **Front Style** list.
- **Size** — The **Size** text box displays the current font size. To change the size, enter a new number in the text box or choose a font size from the list.

- **Effects** — Toggle special font effects:
  - **Strikeout** — Choose this option to produce a horizontal strike-through line in the selected text
  - **Underline** — Choose this option to produce a horizontal underscore line below the selected text
- **Color** — Choose the text color from the drop-down list box. Click on the downward pointing arrow to display the **Color** list. Use the scrolling arrows to view all of the choices, if necessary.
- **Sample** — As **Font** options are chosen, an example of the text that will result is shown in the **Sample** area.
- **Script** — If multilingual support is installed, use this option to choose a non-Western script.

4.13 WinIDE Search Options

This section describes the WinIDE **Search** menu options for specifying search criteria and entering a line number to go to in a source file.

To perform a search operation, click once on the **Search** menu to open the menu (**Figure 4-19**). Click on the option to execute.

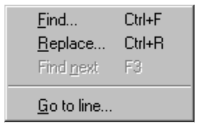


Figure 4-19. Search Menu

4.13.1 Find

Choose the **Find** option from the **Search** menu to open the **Find** dialog (**Figure 4-20**). In the **Find what:** box, enter the string to search for. The search will be performed in the active WinIDE editor source window.

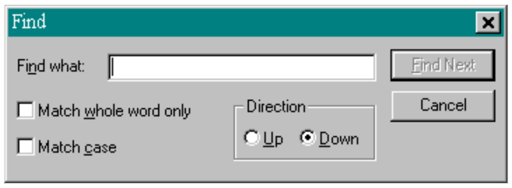


Figure 4-20. Find Dialog Window

Enter the search string and choose from these options to refine the search:

- **Match whole word only** — Choose this option to limit the search to whole words and not character strings that are part of a longer word or string.
- **Match case** — Choose this option to perform a case sensitive search, that is, to find words with a specific uppercase and/or lowercase arrangement.
- **Direction —** Up/Down: Click on an option to direct the search:
  - Choose the **Down** option to direct the search from the current cursor position in the text to the end or bottom of the file.
  - Choose the **Up** option to direct the search from the current position in the text to the beginning or top of the file.

Press the FIND NEXT button to start the search.

NOTE: *The **Find** window is modeless and can remain open, allowing the user to interact with either the **Find** dialog or the source window.*

Alternatives: Press CTRL + F. This is the keyboard equivalent to selecting the **Search-Find** menu option.

4.13.2 Replace

Select the **Replace** option to open the **Replace** dialog (**Figure 4-21**) to search for and substitute text in the active source window.

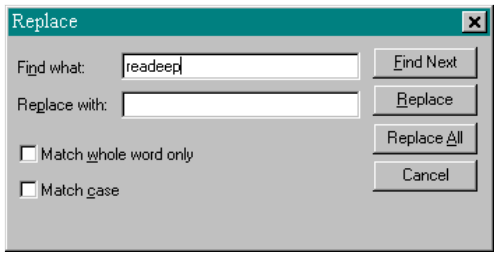


Figure 4-21. Replace Dialog Window

In the **Find what** text box, enter the text string to find; in the **Replace with text** box, enter the text string to replace it with. Refine the search using the **Match whole word only** or **Match case** options.

- **Match whole word only** — Choose this option to limit the search to whole words and not character strings that are part of a longer word or string.
- **Match case** — Choose this option to perform a case sensitive search, that is, to find words with a specific uppercase and/or lowercase arrangement.

Press the CANCEL button to close the **Replace** dialog.

Alternatives: Press CTRL + R. This is the keyboard equivalent to selecting the **Search-Replace** menu option.

4.13.3 Find Next

Select the **Find Next** option from the **Search** menu to find the next occurrence of the previous search string without displaying the **Find** dialog.

Alternatives: Press F3. This is the keyboard equivalent to selecting the **Search-Find Next** menu option.

4.13.4 Go to Line

Select the **Go to Line** option from the **Search** menu to open the **Go to Line Number** dialog (**Figure 4-22**). Note line numbers in the status bar and use the dialog to navigate between points in the text.

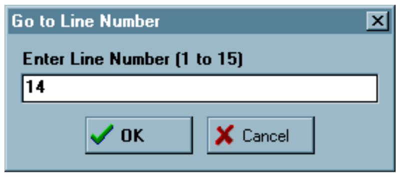


Figure 4-22. Go to Line Number Dialog Window

The dialog instruction includes the range of line numbers available in the active window. Enter the line number to go to, and press the OK button.

4.14 WinIDE Window Options

This section describes the WinIDE **Window** menu options for managing the arrangement of open client windows in the main WinIDE window.

To perform a Window operation, click once on the **Window** menu to open the menu (**Figure 4-23**). Click on the option to execute.

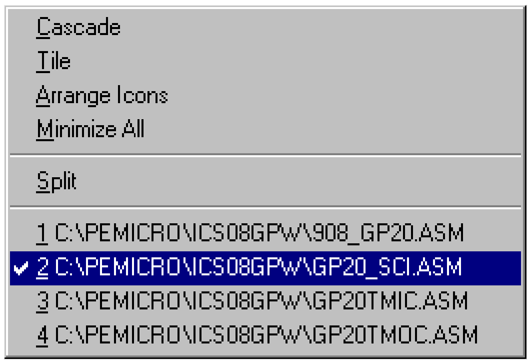
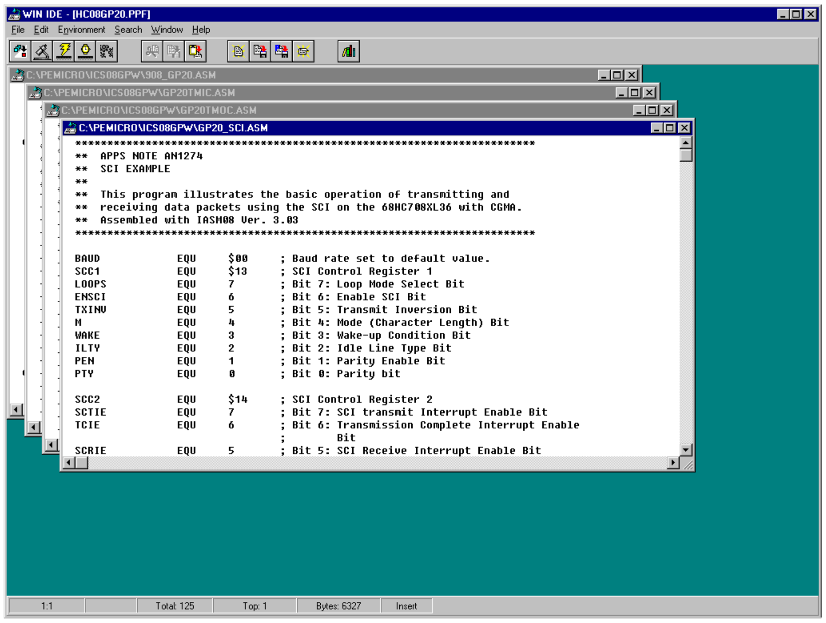

Cascade
Tile
Arrange Icons
Minimize All
Split
1 C:\PEMICRO\ICS08GPW\908_GP20.ASM
✓ 2 C:\PEMICRO\ICS08GPW\GP20_SCI.ASM
3 C:\PEMICRO\ICS08GPW\GP20TMIC.ASM
4 C:\PEMICRO\ICS08GPW\GP20TMOC.ASM

Figure 4-23. WinIDE Window Menu

4.14.1 Cascade

Select the **Cascade** option from the **Window** menu to arrange the open source windows in overlapping or "cascaded" style (**Figure 4-24**), like fanned cards. In this arrangement, open source windows are all set to the same size and shape, one overlapping the other from the upper left hand to the lower right hand corner of the WinIDE main window, with their title bars visible.


WIN IDE - [HC08GP20.PPF]
File Edit Environment Search Window Help
C:\PEMICRO\ICS08GPW\908_GP20.ASM
C:\PEMICRO\ICS08GPW\GP20TMIC.ASM
C:\PEMICRO\ICS08GPW\GP20TMOC.ASM
C:\PEMICRO\ICS08GPW\GP20_SCI.ASM
1:1
Total: 125
Top: 1
Bytes: 6327
Insert

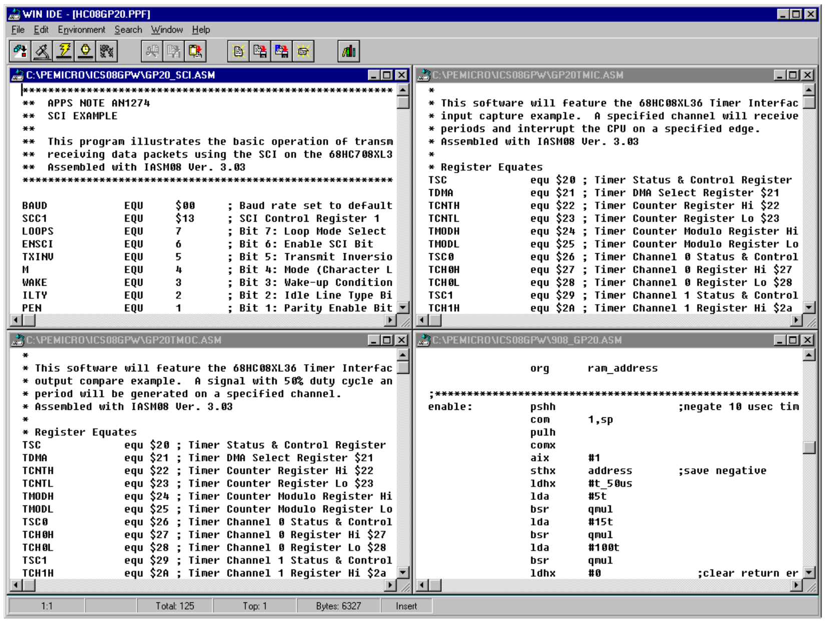
```
************************************************************************
**  APPS NOTE AN1274
**  SCI EXAMPLE
**
**  This program illustrates the basic operation of transmitting and
**  receiving data packets using the SCI on the 68HC708XL36 with CGMA.
**  Assembled with IASM08 Ver. 3.03
************************************************************************

BAUD            EQU     $00     ; Baud rate set to default value.
SCC1            EQU     $13     ; SCI Control Register 1
LOOPS           EQU     7       ; Bit 7: Loop Mode Select Bit
ENSCI           EQU     6       ; Bit 6: Enable SCI Bit
TXINV           EQU     5       ; Bit 5: Transmit Inversion Bit
M               EQU     4       ; Bit 4: Mode (Character Length) Bit
WAKE            EQU     3       ; Bit 3: Wake-up Condition Bit
ILTY            EQU     2       ; Bit 2: Idle Line Type Bit
PEN             EQU     1       ; Bit 1: Parity Enable Bit
PTY             EQU     0       ; Bit 0: Parity bit

SCC2            EQU     $14     ; SCI Control Register 2
SCTIE           EQU     7       ; Bit 7: SCI transmit Interrupt Enable Bit
TCIE            EQU     6       ; Bit 6: Transmission Complete Interrupt Enable
                                ;        Bit
SCRIE           EQU     5       ; Bit 5: SCI Receive Interrupt Enable Bit
```

Figure 4-24. WinIDE with Subordinate Windows Cascaded

To choose a window from the cascaded display, click on its title bar. This moves the selected window to the top of the stack and makes it the active window.

4.14.2 Tile

Select the **Tile** option from the **Window** menu to arrange the open source windows in tiled fashion (**Figure 4-25**). The user will see the entire window border for each, although not necessarily the window's entire contents.


WIN IDE - [HC08GP20.PPF]
File Edit Environment Search Window Help
C:\PEMICRO\ICS08GPW\GP20_SCI.ASM
C:\PEMICRO\ICS08GPW\GP20TMIC.ASM
C:\PEMICRO\ICS08GPW\GP20TMOC.ASM
C:\PEMICRO\ICS08GPW\908_GP20.ASM
1:1
Total: 125
Top: 1
Bytes: 6327
Insert

```
***********************************************************
**  APPS NOTE AN1274
**  SCI EXAMPLE
**
**  This program illustrates the basic operation of transm
**  receiving data packets using the SCI on the 68HC708XL3
**  Assembled with IASM08 Ver. 3.03
***********************************************************

BAUD            EQU     $00     ; Baud rate set to default
SCC1            EQU     $13     ; SCI Control Register 1
LOOPS           EQU     7       ; Bit 7: Loop Mode Select
ENSCI           EQU     6       ; Bit 6: Enable SCI Bit
TXINV           EQU     5       ; Bit 5: Transmit Inversio
M               EQU     4       ; Bit 4: Mode (Character L
WAKE            EQU     3       ; Bit 3: Wake-up Condition
ILTY            EQU     2       ; Bit 2: Idle Line Type Bi
PEN             EQU     1       ; Bit 1: Parity Enable Bit
```

```
*
* This software will feature the 68HC08XL36 Timer Interfac
* input capture example.  A specified channel will receive
* periods and interrupt the CPU on a specified edge.
* Assembled with IASM08 Ver. 3.03
*
* Register Equates
TSC             equ $20 ; Timer Status & Control Register
TDMA            equ $21 ; Timer DMA Select Register $21
TCNTH           equ $22 ; Timer Counter Register Hi $22
TCNTL           equ $23 ; Timer Counter Register Lo $23
TMODH           equ $24 ; Timer Counter Modulo Register Hi
TMODL           equ $25 ; Timer Counter Modulo Register Lo
TSC0            equ $26 ; Timer Channel 0 Status & Control
TCH0H           equ $27 ; Timer Channel 0 Register Hi $27
TCH0L           equ $28 ; Timer Channel 0 Register Lo $28
TSC1            equ $29 ; Timer Channel 1 Status & Control
TCH1H           equ $2A ; Timer Channel 1 Register Hi $2a
```

```
*
* This software will feature the 68HC08XL36 Timer Interfac
* output compare example.  A signal with 50% duty cycle an
* period will be generated on a specified channel.
* Assembled with IASM08 Ver. 3.03
*
* Register Equates
TSC             equ $20 ; Timer Status & Control Register
TDMA            equ $21 ; Timer DMA Select Register $21
TCNTH           equ $22 ; Timer Counter Register Hi $22
TCNTL           equ $23 ; Timer Counter Register Lo $23
TMODH           equ $24 ; Timer Counter Modulo Register Hi
TMODL           equ $25 ; Timer Counter Modulo Register Lo
TSC0            equ $26 ; Timer Channel 0 Status & Control
TCH0H           equ $27 ; Timer Channel 0 Register Hi $27
TCH0L           equ $28 ; Timer Channel 0 Register Lo $28
TSC1            equ $29 ; Timer Channel 1 Status & Control
TCH1H           equ $2A ; Timer Channel 1 Register Hi $2a
```

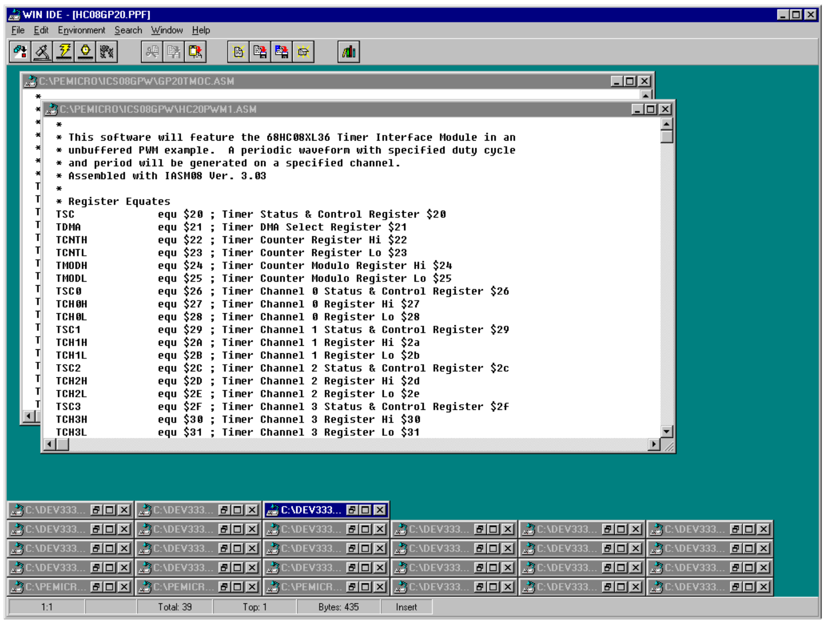
```
                org     ram_address

;**********************************************************
enable:         pshh                    ;negate 10 usec tim
                com     1,sp
                pulh
                comx
                aix     #1
                sthx    address         ;save negative
                ldhx    #t_50us
                lda     #5t
                bsr     qmul
                lda     #15t
                bsr     qmul
                lda     #100t
                bsr     qmul
                ldhx    #0                 ;clear return er
```

Figure 4-25. WinIDE with Subordinate Windows Tiled

If the contents of a source window cannot be displayed in their entirety, use the scroll bars. The tiled arrangement is practical to use when cutting and pasting from one window to another.

4.14.3 Arrange Icons

Select the **Arrange Icons** option from the **Window** menu to rearrange the icons of minimized windows into columns and rows at the bottom of the WinIDE main window (**Figure 4-26**).


WIN IDE - [HC08GP20.PPF]
File Edit Environment Search Window Help
C:\PEMICRO\ICS08GPW\GP20TMOC.ASM
C:\PEMICRO\ICS08GPW\HC20PWM1.ASM
C:\DEV333...
C:\PEMICR...
1:1
Total: 39
Top: 1
Bytes: 435
Insert

```
*
* This software will feature the 68HC08XL36 Timer Interface Module in an
* unbuffered PWM example.  A periodic waveform with specified duty cycle
* and period will be generated on a specified channel.
* Assembled with IASM08 Ver. 3.03
*
* Register Equates
TSC              equ $20 ; Timer Status & Control Register $20
TDMA             equ $21 ; Timer DMA Select Register $21
TCNTH            equ $22 ; Timer Counter Register Hi $22
TCNTL            equ $23 ; Timer Counter Register Lo $23
TMODH            equ $24 ; Timer Counter Modulo Register Hi $24
TMODL            equ $25 ; Timer Counter Modulo Register Lo $25
TSC0             equ $26 ; Timer Channel 0 Status & Control Register $26
TCH0H            equ $27 ; Timer Channel 0 Register Hi $27
TCH0L            equ $28 ; Timer Channel 0 Register Lo $28
TSC1             equ $29 ; Timer Channel 1 Status & Control Register $29
TCH1H            equ $2A ; Timer Channel 1 Register Hi $2a
TCH1L            equ $2B ; Timer Channel 1 Register Lo $2b
TSC2             equ $2C ; Timer Channel 2 Status & Control Register $2c
TCH2H            equ $2D ; Timer Channel 2 Register Hi $2d
TCH2L            equ $2E ; Timer Channel 2 Register Lo $2e
TSC3             equ $2F ; Timer Channel 3 Status & Control Register $2f
TCH3H            equ $30 ; Timer Channel 3 Register Hi $30
TCH3L            equ $31 ; Timer Channel 3 Register Lo $31
```

Figure 4-26. WinIDE: One Source Window Displayed, Remaining Windows Minimized

4.14.4 Minimize All

Select the **Minimize All** option from the **Window** menu to minimize all open source windows and display them as icons at the bottom of the WinIDE main window (**Figure 4-27**).

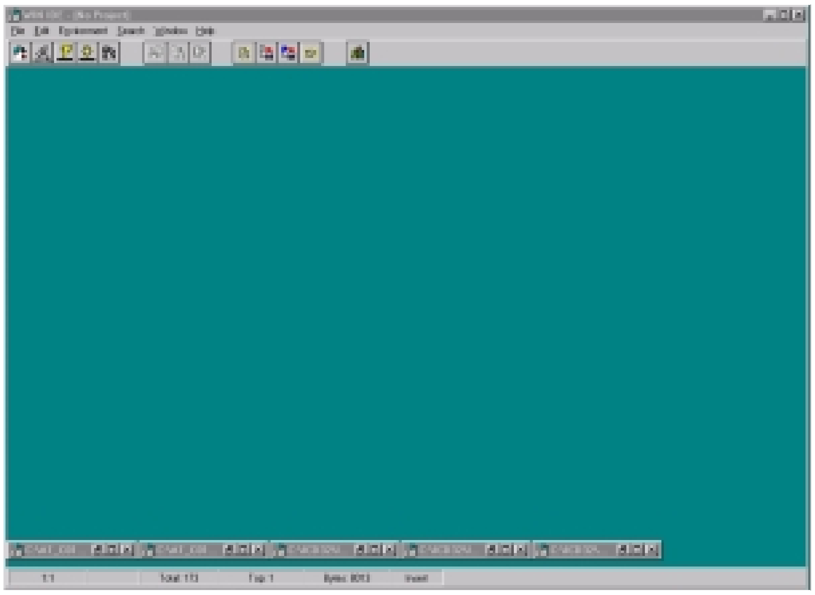


Figure 4-27. WinIDE with Subordinate Windows Minimized

4.14.5 Split

Select the **Split** option from the **Window** menu to divide the active source window into two or more separate panes, each capable of displaying a different view of the same file. To toggle the split window view, click on the **Split** option. A check mark appears beside the option when the split view is in effect.

Adjust the relative size of the panes by dragging the split bar, a double horizontal line separating the panes. Position the pointer over the split bar until it changes to the split pointer (**Figure 4-28**).

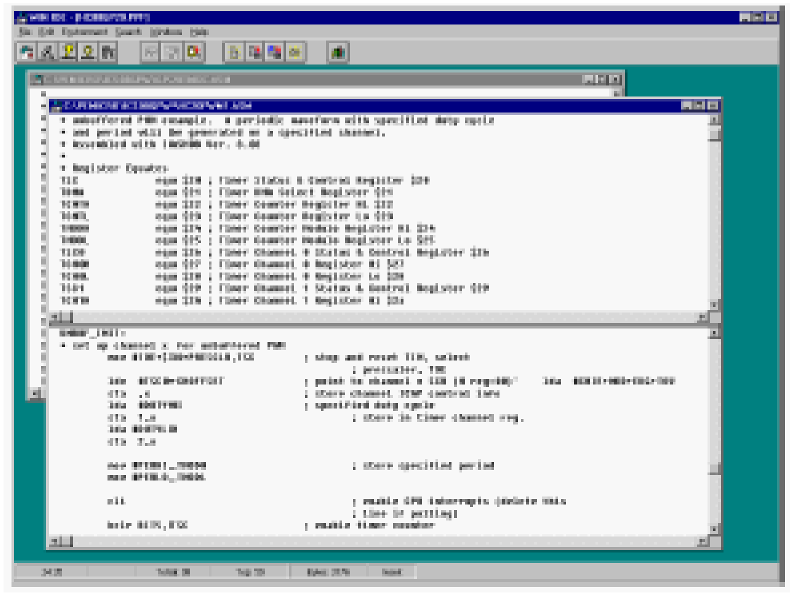


Figure 4-28. WinIDE Cascaded Windows with Active Window Split

Section 5. CASM08W Assembler Interface

5.1 Contents

5.2 Introduction . . . 92
5.3 CASM08W Assembler User Interface . . . 93
5.4 Assembler Parameters . . . 94
5.5 Assembler Outputs . . . 95
5.5.1 Object Files . . . 95
5.5.2 Map Files . . . 95
5.5.3 Listing Files . . . 96
5.5.4 Error Files . . . 97
5.5.5 Files from Other Assemblers . . . 97
5.6 Assembler Options . . . 98
5.6.1 Operands and Constants . . . 98
5.6.2 Comments . . . 99
5.7 Assembler Directives . . . 99
5.7.1 BASE . . . 99
5.7.2 Cycle Adder . . . 100
5.7.3 Conditional Assembly . . . 102
5.7.4 INCLUDE . . . 102
5.7.5 MACRO . . . 103
5.8 Listing Directives . . . 104
5.8.1 Listing Files . . . 105
5.8.2 Labels . . . 107
5.9 Pseudo Operations . . . 107
5.9.1 Equate (EQU) . . . 108
5.9.2 Form Constant Byte (FCB) . . . 108
5.9.3 Form Double Byte (FDB) . . . 109
5.9.4 Originate (ORG) . . . 109
5.9.5 Reserve Memory Byte (RMB) . . . 109
5.10 Assembler Error Messages . . . 109
5.11 Using Files from Other Assemblers . . . 111

5.2 Introduction

This chapter describes the operation of the CASM08W assembler, including methods for interfacing with the assembler from the WinIDE, setting assembler options and directives, generating and using output files and formats, and understanding assembler-generated error messages.

To be used in the target microcontroller CPU, the source code for the program must be converted from its mnemonic codes to the machine code that the target CPU can execute. The CASM assembler program accomplishes this by reading the source code mnemonics and assembling an object code file that can be programmed into the memory of the target microcontroller. Depending on the parameters specified for the assembler, other supporting files can be produced that are helpful in the debugging process.

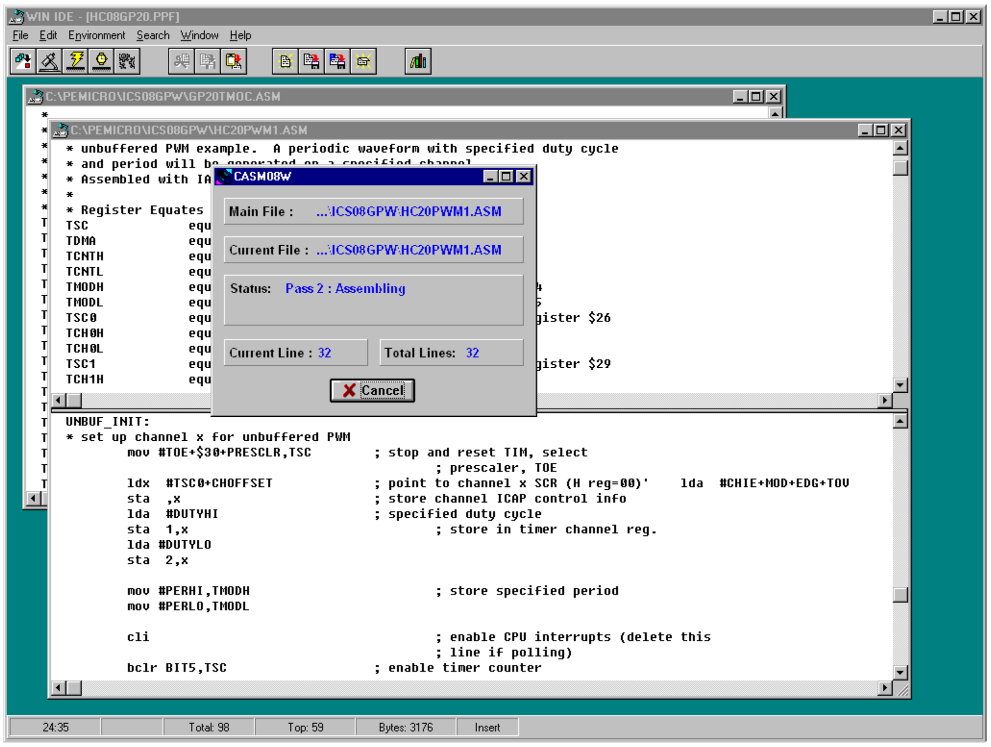
When the user clicks on the ASSEMBLE/COMPILE FILE button or uses the F4 hotkey in the WinIDE, the CASM cross assembler is activated to process the active file in the WinIDE main window according to the parameters entered. In addition to two kinds of object code files, the assembler produces .MAP and/or .LST files as well.

Listing files show the original source code, or mnemonics, including comments, as well as the object code translation. Use this listing during the debugging phase of the development project. It also provides a basis for documenting the program.

5.3 CASM08W Assembler User Interface

The assembler interface consists of a window that appears briefly in the WinIDE main window during assembly. This window (**Figure 5-1**) contains information about the file being assembled:

- **Main File** — Path and filename of the main file being assembled
- **Current File** — Path and filename of the current file being assembled
- **Status** — Assembler status as the assembly proceeds
- **Current Line** — Current line position of the assembler
- **Total Lines** — Total number of lines in the file being assembled


WIN IDE - [HC08GP20.PPF]
File Edit Environment Search Window Help
C:\PEMICRO\ICS08GPW\GP20TMOC.ASM
C:\PEMICRO\ICS08GPW\HC20PWM1.ASM
CASM08W
Cancel
24:35
Total: 98
Top: 59
Bytes: 3176
Insert

```
* unbuffered PWM example.  A periodic waveform with specified duty cycle
* and period will be generated on a specified channel
* Assembled with IA
*
* Register Equates
TSC              equ
TDMA             equ
TCNTH            equ
TCNTL            equ
TMODH            equ
TMODL            equ
TSC0             equ                                  gister $26
TCH0H            equ
TCH0L            equ
TSC1             equ                                  gister $29
TCH1H            equ
```

Main File : ...\ICS08GPW\HC20PWM1.ASM

Current File : ...\ICS08GPW\HC20PWM1.ASM

Status: Pass 2 : Assembling

Current Line : 32   Total Lines: 32

```
UNBUF_INIT:
* set up channel x for unbuffered PWM
        mov #TOE+$30+PRESCLR,TSC        ; stop and reset TIM, select
                                                ; prescaler, TOE
        ldx  #TSC0+CHOFFSET             ; point to channel x SCR (H reg=00)'    lda  #CHIE+MOD+EDG+TOV
        sta  ,x                         ; store channel ICAP control info
        lda  #DUTYHI                    ; specified duty cycle
        sta  1,x                                ; store in timer channel reg.
        lda #DUTYLO
        sta  2,x

        mov #PERHI,TMODH                        ; store specified period
        mov #PERLO,TMODL

        cli                                     ; enable CPU interrupts (delete this
                                                ; line if polling)
        bclr BIT5,TSC                   ; enable timer counter
```

Figure 5-1. WinIDE with CASM08W Assembler Window Displayed

5.4 Assembler Parameters

The CASM08W assembler may be configured by passing parameters in either of two ways:

- From the WinIDE editor, as described in **4.7 Command-Line Parameters** and **4.12.5.3 Assembler/Compiler Tab**.
- From the command line, by using Windows 95 (or later version) **Program Item Property** dialog. Refer to the Windows documentation for information on this procedure.

The following parameters may be entered in any order. To specify multiple parameters, separate them with spaces. All parameters default to off.

- `Filename` — Required parameter specifying the path name and filename of the CASM08W assembler executable
- `S` — Optional parameter to generate Motorola .S19 S-record object file
- `L` — Optional parameter to generate an .LST listing file
- `D` — Optional parameter to generate P&E Microcomputer Systems, Inc. .MAP debugging file
- `H` — Optional parameter to generate Intel® hex object file
- `C` — Optional parameter to show cycle counts in listing file
- `M` — Optional parameter to expand MACROS in listing file
- `I` — Optional parameter to expand INCLUDE files in listing file
- `Q` — Optional parameter to suppress screen writes except errors

Example:

```
     C:\CASM\CASM08W.EXE MYFILE S L D
```

5.5 Assembler Outputs

This section describes the assembler outputs.

5.5.1 Object Files

When an object file in the command-line in the **Program Item Properties** in Windows is specified, using the S or H parameters, the object file is created during assembly. The object file has the same name as the file being assembled, with the extension .HEX or .S19, depending on the specification given:

- Motorola uses the S-record 8-bit object code file format for object files. For more information, see **Appendix A. S-Record Information**.
- .HEX is the Intel 8-bit object code format.

In either case, the object code file produced by the CASM08W assembler is a text file containing numbers that represent the binary opcodes and data of the assembled program. This object code file can be sent to the MCU using a programmer or bootstrap program, at which time it is converted to the binary format required by the target CPU.

The object filename depends on the choice made in the command line of the Windows **Program Item Properties**. By default, the object filename is that of the file being assembled, with the proper object file format extensions. An existing file with the same name will be overwritten.

5.5.2 Map Files

If a map file using the D parameter is specified, the P&E debug .MAP file is created during the assembly. P&E Microcomputer Systems products (such as the MMDS and the MMEVS) use these map files during the source-level debugging process.

Map files contain the directory path information under which they are created, and cannot, therefore, be moved to a new directory. If the map file must be used from a different directory, place the file in the new directory and reassemble, using the map file option D in the Windows command line.

5.5.3 Listing Files

Listing files display each line of source code and the resulting (assembled or compiled) object code. Listing files show exactly how and where each code was assembled.

If a listing file in the environment settings is specified, it is created during assembly. The listing file will have the same name as the file being assembled, with the .LST extension, and will overwrite any previous file with the same name.

Listing files contain these fields in the following format:

```
AAAA [CC] VVVVVVVV LLLL Source Code . . . .
```

Where:

AAAA	The first four hexadecimal digits are the address of the command in the target processor memory.
[CC]	The number of machine cycles used by the opcode. This value, which always appears in brackets, is a decimal value. If an instruction has several possible cycle counts (as would be the case when the assembler encounters a branch instruction) and the assembler cannot determine the actual number of cycle counts, the CC field will show the best case (lowest number).
VVVVVVVV	Hexadecimal digits (the number of digits depends on the actual opcode) represent values put into that memory address.
LLLL	Line count
Source code	The actual source code

The symbol table listing every label and its value is at the end of the listing file.

If a listing file using the `L` parameter in the Windows command line is specified, a file with the same name as the file being assembled and the extension .LST can be produced by the assembler. This file serves as a program listing showing the binary numbers that the CPU needs, alongside the assembly language statements from the source code.

For more information about using the assembler listing directives, see the summary of assembler directives in **Table 5-2**.

5.5.4 Error Files

Error files contain assembly error information. The CASM08W highlights any errors that it encounters during the assembly and displays the error message in the CASM08W window. Depending on the environment settings, the assembler may also open the file in which the error was encountered and create an error file with the assembly filename and the .ERR extension.

5.5.5 Files from Other Assemblers

It is possible to use files produced by another assembler with the CASM08W assembler, providing they are properly prepared before using. To prepare a source file from a third-party assembler for use with the CASM08W, follow these steps:

1. Precede all comments by a semicolon.
2. Using the WinIDE (or other editor) global search and replace command, change any assembler-specific directives, listing directives, pseudo operations, etc., as required to create a file which is compatible with the CASM08W. Remember that assembler directives must begin with the characters `$`, `/`, `.`, or `#`, and must begin in column 1.
3. If necessary, use the BASE directive to change the default base for the operands. (CASM08W defaults to hexadecimal base.)

5.6 Assembler Options

The CASM08W assembler supports all Motorola opcode mnemonics.

NOTE: *Opcode mnemonics cannot start in column one. If a label begins the line, there must be at least one space between the label and the opcode.*

5.6.1 Operands and Constants

Operands are addresses, labels, or constants, as defined by the opcode. Assembly-time arithmetic is allowed within operands. Such arithmetic may use these operations:

*	multiplication
/	division
+	addition
-	subtraction
<	left shift
>	right shift
%	remainder after division
&	bitwise and
|	bitwise or
^	bitwise xor

Operator precedence follows algebraic rules. Use parentheses to alter precedence. If an expression contains more than one operator, parenthesis, or embedded space, the entire expression must be put inside braces (**{ }**).

```
jmp start       ;start is a previously defined label
jmp start+3     ;jump to location start + 3
jmp (start > 2) ;jump to location start divided by 4
```

Constants are specific numbers in assembly-language commands. The default base for constants is hexadecimal, but it can be changed using the **Change Base Address** dialogs for the Memory and Code windows. To temporarily override the default base, use either the appropriate prefix or suffix (**Table 5-1**), but not both.

The assembler also accepts ASCII constants. Specify an ASCII constant by enclosing it in single or double quotes. A character ASCII constant has an equivalent value: ‘A’ is the same as 41H. An example of a string constant is:

```
    db “this is a string”
```

Table 5-1. Change Base Prefixes/Suffixes

Base	Prefix	Suffix
2	%	Q
8	@	O
10	!	T
16		H

5.6.2 Comments

Use semicolons to delineate comments. A comment may start in any column and run until the end of its line. Additionally, if any line has an asterisk (*) or semicolon (;) in column 1, the entire line is a comment.

5.7 Assembler Directives

Assembler directives are keywords that control the progress and the modes of the CASM08W assembler. To invoke an assembler directive, enter a /, #, or $ as the first character of a line. Enter the directive immediately after this initial character, along with the appropriate parameters values.

Directives supported by the assembler vary according to manufacturer. **Table 5-2** summarizes the CASM08W assembler directives. A caret (^) indicates that a parameter value must follow the directive. Note also that a space must separate a directive and its parameter value.

5.7.1 BASE

The BASE assembler directive changes the default base of the current file. The parameter specified must be in the current base or have a base qualifier (prefix or suffix). The next base remains in effect until the end of the file or until another BASE directive is entered.

The original default base is hexadecimal, but the default can be changed to binary, octal, or decimal default bases instead. It is good practice to specify a base explicitly to be always sure that base is currently in effect.

5.7.2 Cycle Adder

The CASM08W assembler contains an internal counter for instruction cycles called the cycle adder. Two assembler directives, CYCLE_ADDER_ON and CYCLE_ADDER_OFF, control this counter.

When the assembler encounters the CYCLE_ADDER_ON directive, it clears the cycle adder. The cycle adder starts a running total of instruction cycles as subsequent instructions are assembled. For instructions that have variable numbers of instruction cycles, the cycle adder uses the smallest number.

When the assembler encounters the CYCLE_ADDER_OFF directive, it writes the current cycle-adder value to the .LST file and disables the cycle adder.

Table 5-2. Assembler Directives

Directive	Action
BASE ^	Change the default input base to binary, octal, decimal, or hexadecimal.
CYCLE_ADDER_OFF	Stop accumulating instruction cycles and print the total.
CYCLE_ADDER_ON	Start accumulating instruction cycles.
INCLUDE ^	Include specified file in source code.
MACRO ^	Create a macro.
MACROEND	End a macro definition.
RAMEND ^	Set logical end of RAM space.
RAMSTART ^	Set default for ramloc pseudo operation.
Conditional Directive	**Action**
SET	Sets the value of its parameter to true. Maximum number of SETs is 25.
SETNOT	Sets the value of its parameter to false. Maximum number of SETNOTs is 25.
IF or IFNOT	Determines the block of code to be used for conditional assembly; the code between the IF and ENDIF will be assembled if the given parameter value is true; the code between IFNOT and ENDIF will be assembled if the parameter value is false.
ELSEIF	Provides alternative to ENDIF when precedes ENDIF; for example, if the parameter value is true, the code between IF and ELSEIF will be assembled, but the code between ELSEIF and ENDIF will not be assembled. If the parameter value is false, code between IF and ELSEIF will not be assembled, but code between ELSEIF and ENDIF will be assembled. ELSEIF gives the same alternative arrangement to a directive sequence that begins with IFNOT.
ENDIF	See IF, IFNOT, ELSEIF.

5.7.3 Conditional Assembly

The CASM08W assembler allows the user to specify blocks of code to be assembled only upon certain conditions. To set up such conditional assembly procedures, use the conditional assembler directives summarized in **Table 5-2**.

Example of conditional assembly directives:

```
$SET debug          ;sets debug = true
$SETNOT test        ;sets test = false
nop                 ;always assembles
nop                 ;always assembles
$IF debug           ;if debug = true
jmp start           ;assembles
$ELSEIF             ;if debug = false
jmp end             ;does not assemble
$ENDIF              ;
nop                 ;always assembles
nop                 ;always assembles
$IF test            ;if test = true
jmp test            ;does not assemble
$ENDIF              ;
```

5.7.4 INCLUDE

If the CASM08W assembler encounters the INCLUDE directive, it takes source code from the specified file and continues until it encounters another INCLUDE directive or until it reaches the end of the main file. When the assembler reaches the end of the main file, it continues taking source code from the file that contained the include directive.

The file specification of the INCLUDE directive must be in either single or double quotes. If the file is not in the current directory, the specification should also include the full path name as well as the filename.

The user may nest INCLUDE to a maximum depth of 10, that is, each included file may contain up to 10 additional included files.

Examples:

```
$INCLUDE “INIT.ASM”
$INCLUDE “C:\project\init.asm*"
```

5.7.5 MACRO

A macro is a named block of text to be assembled. Similar in some ways to an included file, the macro allows labels and parameter values.

The MACRO directive begins the macro definition. The name of the macro is the parameter value for the MACRO directive. All subsequent code, until the assembler encounters the MACROEND directive, is considered the macro definition.

Assembler directives may not be used within a macro, nor does the definition require parameter names. Instead, the macro definition includes the sequential indicators %n for the nth parameter values of the macro call. The assembler will ignore parameter values on the MACRO directive line, so such values may be helpful for internal documentation.

Example:

This macro example illustrates a macro that divides the accumulator value by 4:

```
$MACRO divide_by_4    ;starts macro definition
asr a                 ;divides accumulator by 2
asr a                 ;divides quotient by 2
$MACROEND             ;ends macro definition
```

This macro example illustrates a macro that creates a time delay:

```
$MACRO delay          count
                      ldaa #$01
loop:                 deca
                      bne loop
$MACROEND
```

In this macro, the CASM08W assembler ignores the parameter `count` on the MACRO directive line. The parameter *count* merely indicates the role of the parameter value passed to the macro. That value is substituted for the sequential indicator %1. The first time this macro is called, the CASM08W assembler changes the label `loop`, on lines 3 and 4, to `loop:0001`. If the calling line

```
delay 100t
```

invokes this macro, the loop would occur 100 times. The suffix `t` represents the decimal base.

The CASM08W assembler ignores extra parameter values sent to a macro. If the macro does not receive enough parameter values, the assembler issues an error message.

Labels change automatically each time they are used. Labels used within macros may not be longer than 10 characters, because the assembler appends a 4-digit hexadecimal number to the label to ensure label uniqueness.

Although code may not jump into a macro, it may jump out of a macro. Macros cannot be forward-referenced.

5.8 Listing Directives

Listing directives are source-code keywords that control output to the .LST listing file. These directives pertain only to viewing the source-code output; the directives, which may be interspersed anywhere in source code, do not affect the actual code assembled. **Table 5-3** summarizes the listing directives.

Table 5-3. Listing Directives

Directive	Action
EJECT OR PAGE	Begins a new page
HEADER ^	Specifies a header on listing pages; the header can be defined only once; the default header is blank; the header string is entered in quotes.
LIST	Turns on the .LST file output.
NOLIST	Turns off the .lst file output. This directive is the counterpart of the list directive; at the end of a file, this directive keeps the symbol table from being listed.
PAGELENGTH ^	Sets the length of the page; the default parameter value is 166 lines (! = decimal)
PAGEWIDTH ^	Sets the width of the output, word wrapping additional text; the default parameter value is 160 columns (! = decimal)f
SUBHEADER '^'	Makes the string specified in quotes (double or single) a subheader on the listing pages; the subheader takes effect on the next page

Note: The caret (^) character following a directive indicates a mandatory parameter value that must be supplied.

5.8.1 Listing Files

If a listing file is requested using the `L` parameter in the command line of the Windows **Program Item Properties**, or the **Output Listing File** option is checked in the **Assembler/Compiler** tab in the **Environment Settings** dialog, the listing file (.LST) is created during the assembly.

This listing file has the same name as the file being assembled, but with the extension .LST. Any existing file with the same name will be overwritten.

The listing file has this format (file fields shown in the example are described in **Table 5-4**):

```
AAAA  [CC]  VVVVVVVV  LLLL  Source Code. . . . .
```

Example:

```
0202 [05] 1608  37  bset 3,tcsr  clear timer overflow flag
```

The listing file fields are described in **Table 5-4**.

Table 5-4. Listing File Fields

Field Contents	Field Description
AAAA	The first field contains four hexadecimal digits indicating the address of the command in the target processor (MCU) memory. The assembler generates this field.
[CC]	The second field indicates the number of machine cycles used by the opcode. The assembler generates this field. Note that this value appears only if the cycle counter (Cycle Cntr) was turned on before assembly. Also note that the CC value, which always appears in brackets, is a decimal value. If a command has several possible cycle counts and the assembler cannot determine the actual number, the CC field shows the best case (lowest number). An example of a command that may have several possible counts is a branch command.
VVVVVVVV	The third field contains a label consisting of four hexadecimal digits indicating the values placed into that memory address and, possibly, the next several memory addresses. Refer to this label in other commands. The size of this field depends on the actual opcode. The assembler derives this field from the source code.
LLLL	The fourth field may contain up to four digits indicating the line count. The assembler derives this field from the source code.

Table 5-4. Listing File Fields (Continued)

Field Contents	Field Description
`Source code`	The last field contains actual source code from the source code file.
`Listing table`	The listing table provides a summary of every label and its value, displayed in table format at the end of each listing file.

Example listing table:

```
MAIN1.ASM     Assembled with CASM08W  2/27/97  12:06:39
              PM  PAGE 2

      0000             26  porta  equ  $0000
      0000             27  portb  equ  $0001
      0000             28  portc  equ  $0002
      0000             29  portd  equ  $0003
      0000             30  ddra   equ  $0004
      0000             31  ddrb   equ  $0005
      0000             32  ddrc   equ  $0006
      0000             3   ddrd   equ  $0007
      . . . .
      Symbol table
      DONSCN     08DD
      DONSCN1    08EE
      OPTSC      0866
      OPTSE2     0877
      OPTSC3     0888
      . . . .
```

5.8.2 Labels

As the program code is written, the user will not necessarily know the addresses where commands will be located. The assembler solves this problem using a system of labels, providing a convenient way to identify specific points in the program without knowing the exact addresses. The assembler later converts these mnemonic labels into specific memory addresses and even calculates the offsets for branch commands for the CPU to use.

Labels within macros must not exceed 10 characters in length.

Examples:

```
Label:
ThisIsALabel:
Loop_1
This_label_is_much_too_long:
```

The assembler would truncate the last example to 16 characters.

5.9 Pseudo Operations

The CASM08W assembler also allows pseudo operations (in place of opcode mnemonics), which are summarized in **Table 5-5**.

Table 5-5. Pseudo Operations Allowed by the CASM08W

Pseudo Op Code	Action
equ	Associates a binary value with a label
fcb m or db m	Defines byte storage, where m = label, number, or string. Strings generate ASCII code for multiple bytes; number and label parameters receive single bytes. Separate multiple parameters with commas.
fdb n or dw n	Defines word storage, where n = label, number, or string. Two bytes are generated for each number or label. Separate multiple parameters with commas.
org n	Sets the origin to the value of the number or label n. No forward references of n are allowed.
rmb n or ds n	Defines storage, reserving n bytes, where n = number or label. No forward references of n are allowed.

5.9.1 Equate (EQU)

The equate directive associates a binary value with a label. The value may be either an 8-bit value or a 16-bit address value. This directive does not generate any object code.

During the assembly process, the assembler must keep a cross-reference list where it stores the binary equivalent of each label. When a label appears in the source program, the assembler looks in this cross-reference table to find the binary equivalent. Each EQU directive generates an entry in this cross-reference table.

An assembler reads the source program twice. On the first pass, the assembler just counts bytes of object code and internally builds the cross-reference table. On the second pass, the assembler generates the listing file and/or the S-record object file, as specified in the command line parameters for the assembler. This two-pass arrangement allows the programmer to reference labels that are defined later in the program.

EQU directives should appear near the beginning of a program, before their labels are used by other program statements. If the assembler encounters a label before it has been defined, the assembler has no choice but to assume the worse case and assign the label a 16-bit address value. This would cause the extended addressing mode to be used in places where the more efficient direct addressing mode could have been used. In other cases, the indexed 16-bit offset addressing mode may be used where a more efficient 8-bit or no offset indexed command could have been used.

5.9.2 Form Constant Byte (FCB)

The arguments for this assembler directive are labels or numbers (separated by commas) that the assembler can convert into a single byte of data. Each byte specified by the FCB directive generates a byte of machine code in the object code file. Use FCB directives to define constants in a program.

5.9.3 Form Double Byte (FDB)

The arguments for this assembler directive are labels or numbers (separated by commas) that the assembler can convert into 16-bit data values. Each argument specified in an FDB directive generates two bytes of machine code in the object code file.

5.9.4 Originate (ORG)

The originate directive sets the location counter for the assembler. The location counter keeps track of the address where the next byte of machine code will be stored in memory.

As the assembler translates program statements into machine code commands and data, it advances the location counter to point to the next available memory location.

Every program has at least one ORG directive to establish the program's starting place. Most complete programs will also have a second ORG directive near the end of the program to set the location counter to the address where the reset and interrupt vectors are located. The reset vector must always be specified. It is good practice to also specify interrupt vectors, even if the user does not expect to use interrupts.

5.9.5 Reserve Memory Byte (RMB)

Use this assembler directive to set aside space in RAM for program variables. The RMB directive does not generate any object code, but it normally generates an entry in the assembler’s internal cross-reference table.

5.10 Assembler Error Messages

Configure the CASM08W assembler to highlight any errors that it encounters during assembly and display an error message on the prompt line. **Table 5-6** summarizes these messages.

Table 5-6. Assembler Error Messages

Message	Probable Cause	Corrective Action
`Conditional assembly variable not found`	The variable in the IF or IFNOT statement has not been declared via SET or SETNOT directive.	Declare the variable using the SET or SETNOT directive.
`Duplicate label`	The label in the highlighted line already has been used.	Change the label to one not used already.
`Error writing .LST or .MAP file—check disk space`	Insufficient disk space or other reason prevents creation of an .LST or .MAP file.	Make sure there is sufficient disk space. Make sure that the CONFIG.SYS file lets multiple files be open at the same time (see DOS or Windows manual for commands).
`Error writing object file—check disk space`	Insufficient disk space or other reason prevents creation of an object file.	Make sure there is sufficient disk space. Make sure the CONFIG.SYS file allows multiple files be open at the same time (see DOS or Windows manual for commands).
`Include directives nested too deep`	Includes are nested 11 or more levels deep.	Nest includes no more than 10 levels deep.
`INCLUDE file not found`	Assembler could not find the file specified in the INCLUDE directive.	Make sure that quotes enclose the file name to be included; if necessary, specify the full path name as well.
`Invalid base value`	Value is inconsistent with current default base (binary, octal, decimal, or hexadecimal).	Use a qualifier prefix or suffix for the value or change the default base.
`Invalid opcode, too long`	The opcode on the highlighted line is wrong.	Correct the opcode.
`MACRO label too long`	A label in the macro has 11 or more characters.	Change the label to have no more than 10 characters.
`MACRO parameter error`	The macro did not receive sufficient parameter values.	Send sufficient parameter values to the macro.
`Out of memory`	The assembler ran out of system memory.	Create a file that consists only of an INCLUDE directive, which specifies the primary file. Assembling this file leaves the maximum memory available to the assembler.
`Parameter invalid, too large, missing, or out of range`	Operand field of the highlighted line has an invalid number representation. Or the parameter value evaluates to a number too large for memory space allocated to the command.	Correct the representation or change the parameter value.

Table 5-6. Assembler Error Messages (Continued)

Message	Probable Cause	Corrective Action
`Too many conditional assembly variables`	There are 26 or more conditional variables.	Limit conditional variables to 25 or fewer.
`Too many labels`	The assembler ran out of system memory.	Create a file that consists only of an INCLUDE directive, which specifies the primary file. Assembling this file leaves the maximum memory available to the assembler.
`Undefined label`	The label parameter in the highlighted line has not been declared.	Declare the label.
`Unrecognized operation`	The highlighted opcode is unknown or is inconsistent with the number and type of parameters.	Correct the opcode or make it consistent with parameters.
`'}' not found`	A mathematical expression is missing its closing brace.	Insert the closing brace.

5.11 Using Files from Other Assemblers

To prepare a source file made by another assembler with CASM08W, follow these steps:

1. Divide large files into smaller files. Typically, use one file for system variables and EQUates, another file for I/O (input/output) routines. The main file should be the one called. Remember that include filenames must be in quotes and must contain the file extensions.
2. Make sure all comments in the source file are preceded by a semicolon.
3. Use the global find-and-replace operation in the editor to change any assembler directives, listing directives, and/or pseudo operations, if they exist in the source code. Remember that assembler directives must begin with the character $, /,., or #, and must start in column 1.
4. If necessary, use the BASE directive to change the default base for operands. (CASM08W defaults to hexadecimal.)

Section 6. ICS08GPW In-Circuit Simulator User Interface

6.1 Contents

6.2 Introduction 115

6.3 ICS08GPW Description 115
6.3.1 ICS08GPW Simulation Speed 116
6.3.2 System Requirements for ICS08GPW 116
6.3.3 File Types and Formats 116

6.4 Starting ICS08GPW 118

6.5 ICS08GPW Windows 120

6.6 Code Windows 121
6.6.1 To Display the Code Windows Shortcut Menus 121
6.6.2 Code Window Shortcut Menu Functions 122
6.6.3 Code Window Keyboard Commands 123

6.7 Variables Window 124
6.7.1 Displaying the Variables Shortcut Menu 124
6.7.2 Variables Shortcut Menu Options 124
6.7.3 Variable Window Keyboard Commands 126

6.8 Memory Window 126

6.9 Status Window 128

6.10 CPU Window 130
6.10.1 Changing Register Values 130
6.10.2 CPU Window Keyboard Commands 132

6.11 Cycles Window 132

6.12 Stack Window 132
6.12.1 Interrupt Stack 133
6.12.2 Subroutine Stack 133

6.13 Trace Window. 134

6.14 Breakpoint Window 135
6.14.1 Adding a Breakpoint 136
6.14.2 Editing a Breakpoint 137
6.14.3 Deleting a Breakpoint 137
6.14.4 Removing All Breakpoints. 138

6.15 Register Block Window 138

6.16 Entering Debugging Commands. 140

6.17 ICS08GPW Toolbar 140

6.18 ICS08GPW Menus 142

6.19 File Options. 143
6.19.1 Load S19 File 144
6.19.2 Reload Last S19 144
6.19.3 Play Macro 145
6.19.4 Record Macro. 145
6.19.5 Stop Macro 146
6.19.6 Open Logfile 146
6.19.7 Close Logfile 148
6.19.8 Exit 148

6.20 ICS08GPW Execute Options 148
6.20.1 Reset Processor 149
6.20.2 Step. 149
6.20.3 Multiple Step 149
6.20.4 Go. 149
6.20.5 Stop 150
6.20.6 Repeat Command. 150

6.21 ICS08GPW Window Options. 150
6.21.1 Open Windows. 151
6.21.2 Change Colors 151
6.21.3 Reload Desktop 152
6.21.4 Save Desktop 152

6.2 Introduction

This section describes the in-circuit simulator user interface, toolbar buttons, windows, subwindows, messages, and menu options.

6.3 ICS08GPW Description

The ICS08GPW in-circuit simulator software is the debugging component of a complete development environment when used in conjunction with the WinIDE editing environment, the CASM08W command-line assembler, the PROG08SW FLASH memory programmer, and the ICD08SW in-circuit debugger.

With this package, designers can create source code, assemble the code, debug the code, and program 68HC908GP20 devices. The WinIDE environment operates as a standard ASCII file (such as an assembly file) editor for Windows and includes some speed buttons for calling upon customized assemblers, compilers, and debuggers. In conjunction with the CASM08W assembler and the ICD08SW in-circuit debugger, this environment can allow assembled files to be downloaded and tested while the original source code is modified and assembled.

The ICS08GPW gets input and output for the device from the external hardware pod attached to the host computer. I/O from a custom target system can be used by attaching the pod to the target board with the appropriate cables. 68HC908GP20 devices can be programmed through the M68HC08PGMR board using the PROG08SW software.

ICS08GPW is a non-real-time debugger. MCU code runs only as fast as it can be simulated by the host PC. For real-time execution, use the ICD08SW real-time, in-circuit debugger.

ICS08GPW accepts any standard Motorola S19 S-record files as input for simulation. These object files can be created by any HC08 assembler, such as CASM08W. However, to perform source-level debugging through these files (in the code window), P&E-compatible source-level map files must be loaded. These map files can be generated through CASM08W. If a third-party compiler (assembly or C) is being used, the compiler must be able to produce P&E-compatible source-level map files.

6.3.1 ICS08GPW Simulation Speed

The user should be aware of a difference in speed between simulation and in-circuit simulation.

Simulation is generally faster but does not involve real input and output. The software can be set for simulation at startup by using the SIM08 command. In addition, if power to the pod is off at startup, the user will have the option of choosing simulation from the buttons in the communications error window.

In-circuit simulation is slower but involves real input and output. The POD command allows the user to reconnect to the module for in-circuit simulation.

6.3.2 System Requirements for ICS08GPW

The ICS08GPW runs under Windows 95 or later versions.

The host computer should have a minimum of 2 Mbytes of RAM (system memory) available for assembly processes, as well as sufficient disk space to store the files that the ICS08GPW creates.

6.3.3 File Types and Formats

Use a number of file types in conjunction with the ICS08GPW simulator. The following topics describe the use and structure of each type:

- S19 (Object) Files — The ICS08GPW software accepts any standard Motorola S19 files as input for simulation. S19 object files can be created by any HC08 assembler (such as CASM08W) and contain the actual object code that is simulated by the ICS08GPW. Specify the S19 files to be used on the command line or load them using the LOAD command in the ICS08GPW Status window.
  - The object file has the same name as the file assembled, with the extension .HEX or .S19, and contains the actual assembled (or object) code to debug. If an object file in the environment settings is specified, it is created during assembly.

- The CASM08W (and some other assemblers) product object files in the .S19 format. The Motorola S19 object code format is described in detail in **Appendix A. S-Record Information**. Hex files are the Intel 8-bit object code format.

- Map files contain source-level debugging information. To debug symbolic or source code in the code window, one or more P&E map files must also be loaded. The *.MAP source-level map file can be generated by specifying the map files option on the command line when running the CASM08W assembler or loaded using the LOADMAP command in the ICS08GPW Status window. If a map file is specified in the environment settings, it is created during assembly.

NOTE: *Map files contain directory information, so they cannot be moved. To place map files in another directory, move map files to the new directory and reassemble the file in the new directory so the new map files will contain the correct directory information.*

If a third-party assembly language or C compiler is used, it must be able to produce compatible source-level map files.

- Script files are plain ASCII text files containing ICS08GPW simulator commands. Use any command in the ICS08GP command set in script files. Running the script file then has the effect of entering the commands in it in the ICS08GPW command line. The user can create script files in the WinIDE editor using files created by other text editors following these rules:
  - Enter each command on its own line.
  - Preface comments with a semi-colon.
  - Use commands from the ICS08GPW command set and WAIT.

- Logfiles are simple ASCII text files, sometimes called scratch pad files. The logfile records the sequence and content of commands executed, and the debugger responds to the commands. The user can view logfiles from within the WinIDE editor. The ICS08GPW simulator creates logfiles if the LOGFILE or LF command is active.

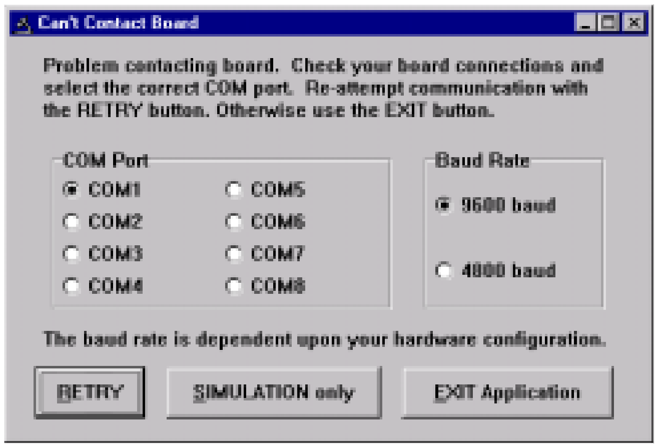
6.4 Starting ICS08GPW

Start the ICS08GPW simulator by itself in stand-alone mode (with no inputs or outputs from the hardware pod) or run it from within the WinIDE editor. Also the user can modify the ICS08GPW environment in the WinIDE editor.

- To run the simulator from the WinIDE editor, use either of these methods:
  - Click the DEBUGGER (EXE1) button on the WinIDE toolbar
  - Press the F6 hotkey
- To modify how the software starts from WinIDE editor:
  - From the WinIDE **Environment** menu, choose the **Setup Environment** option to open the **Environment Settings** dialog.
  - Select the **EXE1 Debugger** tab heading (see **Figure 4-17. Environment Settings Dialog: EXE Tabs**), if it is not already on top, to set options for the ICS08GPW simulator. For more information about the options in the tab, see **4.12.5.4 Executable Tabs: EXE 1-3**.
- To run the simulator directly from Windows, double click the ICS08GPW icon using this method:
  - In Windows 95 (or later versions), choose the ICS08GPW icon from the ICS08GPW group in the **Start** menu.

After startup, the software will establish communication with the board at the given parameters and the status bar will read `Attempting to contact COM 1`.

- If the ICS08GPW software can communicate with the pod through the serial port, the status bar message reads `Contact with pod established`.
- If the software is not able to connect with the pod, the **Can't Contact Board** dialog window (**Figure 6-1**) appears.


Can't Contact Board

Problem contacting board. Check your board connections and select the correct COM port. Re-attempt communication with the RETRY button. Otherwise use the EXIT button.

COM Port: [✓ COM1] COM2 COM3 COM4 COM5 COM6 COM7 COM8

Baud Rate: [✓ 9600 baud] 4800 baud

The baud rate is dependent upon your hardware configuration.

Figure 6-1. Can't Contact Board Dialog Window

If the communication parameters for the communications port and baud rate are incorrect in the **Can't Contact Board** dialog window, change them and then press the RETRY button. If the board is not connected or the user doesn't want to use I/O from the board, then click the SIMULATION ONLY button. Otherwise, press the EXIT APPLICATION button.

When starting the ICS08GPW software for the first time, the **Pick Device** dialog offers choices of different M68HC08 devices (chips). To open this dialog and change the device later, enter the CHIPMODE command in the ICS08GPW Status window command line.

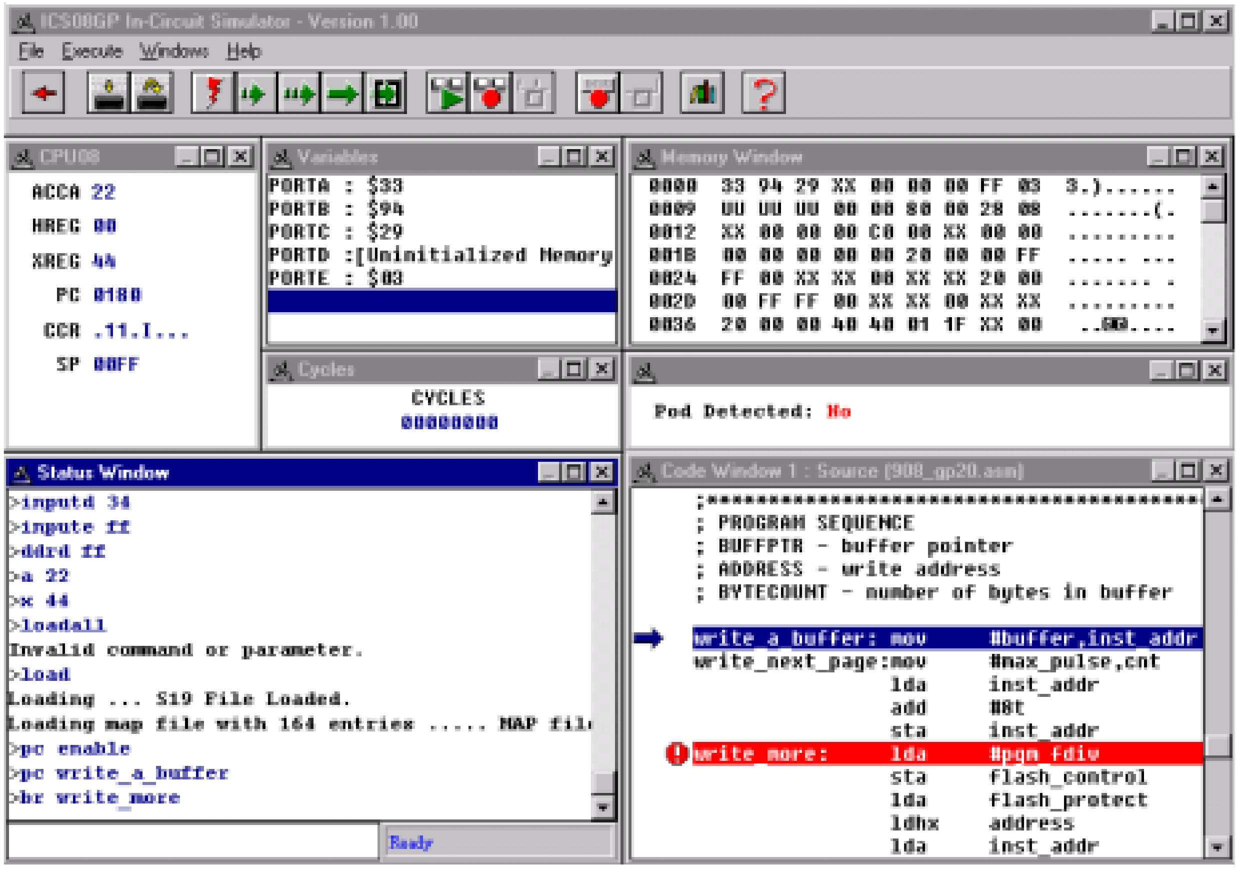
NOTE: *If a file named STARTUP.08 exists in the current directory, the WinIDE runs it as a macro file on startup. See the MACRO command for more information.*

6.5 ICS08GPW Windows

The ICS08GPW user interface consists of windows in which system and code information is shown and into which the ICS08GPW command set can be entered (**Figure 6-2**).

The ICS08GPW also displays these subwindows when appropriate:

- Stack window
- Trace window
- Breakpoint window
- Programmer windows
- Register Block window


ICS08GP In-Circuit Simulator - Version 1.00
File
Execute
Windows
Help
CPU08
Variables
Memory Window
Cycles
Status Window
Code Window 1 : Source (908_gp20.asm)
Ready

```
ACCA 22
HREG 00
XREG 44
  PC 0180
 CCR .11.I...
  SP 00FF
```

```
PORTA : $33
PORTB : $94
PORTC : $29
PORTD :[Uninitialized Memory
PORTE : $03
```

```
0000  33 94 29 XX 00 00 00 FF 03  3.)......
0009  UU UU UU 00 00 80 00 28 08  .......(.
0012  XX 00 00 00 C0 00 XX 00 00  .........
001B  00 00 00 00 00 20 00 00 FF  ..... ...
0024  FF 00 XX XX 00 XX XX 20 00  ....... .
002D  00 FF FF 00 XX XX 00 XX XX  .........
0036  20 00 00 40 40 01 1F XX 00   ..@@....
```

```
CYCLES
00000000
```

```
Pod Detected: No
```

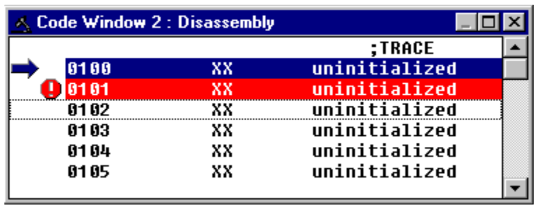
```
>inputd 34
>inpute ff
>ddrd ff
>a 22
>x 44
>loadall
Invalid command or parameter.
>load
Loading ... S19 File Loaded.
Loading map file with 164 entries ..... MAP fil
>pc enable
>pc write_a_buffer
>br write_more
```

```
;************************************
; PROGRAM SEQUENCE
; BUFFPTR - buffer pointer
; ADDRESS - write address
; BYTECOUNT - number of bytes in buffer

write_a_buffer: mov      #buffer,inst_addr
write_next_page:mov      #max_pulse,cnt
                lda      inst_addr
                add      #8t
                sta      inst_addr
write_more:     lda      #pgm_fdiv
                sta      flash_control
                lda      flash_protect
                ldhx     address
                lda      inst_addr
```

Figure 6-2. ICS08GPW Windows Default Positions

6.6 Code Windows

The Code windows (Code 1 and Code 2) can be set to display source code in either source or disassembly modes. Code windows also give visual positions of the current program counter (PC) and all breakpoints within the source code. Display both code windows simultaneously. Each code window is independent. The user can configure each window to display different parts of the source code or different assembly modes. See **Figure 6-3**.



```
                              ;TRACE
0100          XX          uninitialized
0101          XX          uninitialized
0102          XX          uninitialized
0103          XX          uninitialized
0104          XX          uninitialized
0105          XX          uninitialized
```

Figure 6-3. Code Window 2 in Disassembly Mode with Breakpoint Toggled

6.6.1 To Display the Code Windows Shortcut Menus

To display the **Code 1** or C**ode 2 Windows Shortcut** menu (**Figure 6-4**), position the cursor in either the Code 1 or Code 2 window and click the right mouse button.

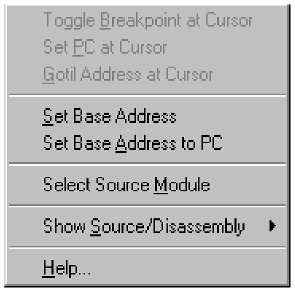


Figure 6-4. Code Window Shortcut Menu

6.6.2 Code Window Shortcut Menu Functions

The **Code Window Shortcut** menu (**Figure 6-4**) offers these options:

Toggle Breakpoint at Cursor

Choose this option to set or remove the breakpoint at the current cursor location. This option is enabled only if the user has clicked on a line of code to select it.

Set PC at Cursor

Choose this option to set the program counter (PC) to the current cursor location. This option is enabled only if the user has clicked on a line of code to select it.

Gotil Address at Cursor

Choose this option to execute the source code until the PC gets to the line at the current cursor location. When PC gets to that point, execution stops. This option is enabled only if the user has clicked on a line of code to select it.

Set Base Address

Choose this option to open the **Window Base Address** dialog (**Figure 6-5**) and set the new address for the first code line in the Code window.

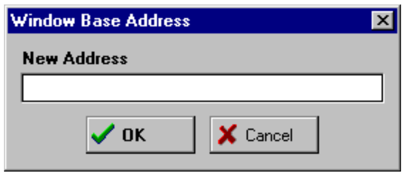


Figure 6-5. Window Base Address Dialog Window

Set Base Address to PC

Choose this option to set the PC to the address of the first line in the Code window.

Select Source Module

Choose this option to select a source module, if a MAP file has been loaded into memory.

Show Disassembly

Choose this option to display the Code window contents in disassembly mode.

Show Source/Disassembly

Choose this option to display the Code window contents in both disassembly and source modes.

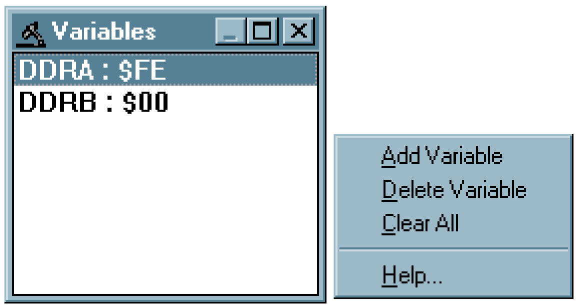
6.6.3 Code Window Keyboard Commands

Use these keys to navigate in the Code windows:

- Press the up arrow (↑) key to scroll the Code window contents up one line.
- Press the down arrow (↓) key to scroll the Code window contents down one line.
- Press the HOME key to scroll to the Code window's base address.
- Press the END key to scroll to the Code window's last address.
- Press the PAGE UP key to scroll the Code window up one page.
- Press the PAGE DOWN key to scroll the Code window down one page.
- Press the F1 key to show the help contents topic.
- Press the ESCAPE (ESC) key to move the cursor to the command line of the Status window.

6.7 Variables Window

The Variables window (**Figure 6-6**) displays current variables during execution. Use the **Variables Shortcut** menu to add or remove variables from the current list.



DDRA : $FE
DDRB : $00

Figure 6-6. Variables Window with Shortcut Menu

6.7.1 Displaying the Variables Shortcut Menu

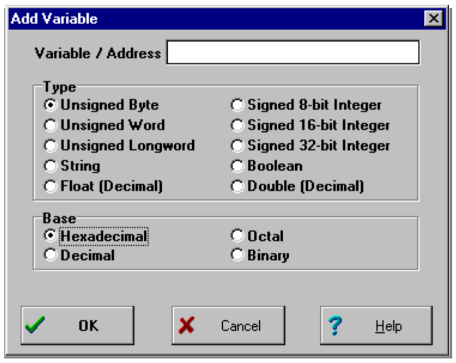
To display the **Variables Shortcut** menu, position the cursor in the Variables window and click the right mouse button.

6.7.2 Variables Shortcut Menu Options

The **Variables Shortcut** menu offers these options for managing variables:

- **Add Variable** — Choose this option to open the **Add Variable** dialog (**Figure 6-7**) to add a variable or address to the current variable list. Select the variable type (size) and base.

  Enter values for commands in the simulator as either labels (which the user has defined in the map file or with the SYMBOL command) or as numbers. Specify the base in which variables are shown using the options in the **Add Variable** dialog (**Figure 6-7**). The default number format for the ICS08GPW is hexadecimal.



Type: [✓ Unsigned Byte], Unsigned Word, Unsigned Longword, String, Float (Decimal), Signed 8-bit Integer, Signed 16-bit Integer, Signed 32-bit Integer, Boolean, Double (Decimal)

Base: [✓ Hexadecimal], Decimal, Octal, Binary

Figure 6-7. Add Variable Dialog Window

To override the default base for any number, also enter either a prefix or suffix (but not both) shown in **Table 6-1** in the command lines.

Table 6-1. Base Prefixes and Suffixes

Base	Prefixes	Suffixes
16	$	H
10	!	T
8	@	O
2	%	Q

Example:

```
$FF = !255 = @377 = %11111111 = 11111111Q =
377O = 255T = 0FFH
```

Use the **Type** options in the **Add Variable** dialog to choose a variable type: 8-bit bytes, 16-bit words, 32-bit longs, or ASCII strings.

- **Delete Variable** — Choose this option to remove the selected (highlighted) variable from memory and from the current variable list.
- **Clear All** — Choose this option to clear all variables in the current variable list.

6.7.3 Variable Window Keyboard Commands

Use these keys to navigate in the Variable window:

- Press the INSERT key to add a variable.
- Press the DELETE key to delete a variable.
- Press the up arrow (↑) key to scroll the Variable window up one variable.
- Press the down arrow (↓) to scroll the Variable window down one variable.
- Press the HOME key to scroll the Variable window to the first variable.
- Press the END key to scroll the Variable window to the last variable.
- Press the PAGE UP key to scroll the Variable window up one page.
- Press the PAGE DOWN key to scroll the Variable window down one page.
- Press the F1 key to shows the help contents topics.
- Press the ESCAPE (ESC) key to move the cursor to the command line of the Status window.

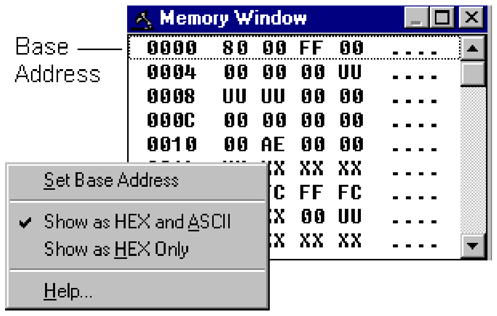
6.8 Memory Window

Use the Memory window (**Figure 6-8**) to view and modify the memory in ICS08GPW. View bytes by using the scrollbar on the right side of the window.

To modify a set of bytes:

1. Double click on the bytes to open the **Modify Memory** dialog for that address.
2. Enter the MM command in the command line of the Status window.

NOTE: *The value xx means that the memory location is uninitialized and indeterminate. The value W means that it is unimplemented, invalid memory.*



```
0000   80 00 FF 00   ....
0004   00 00 00 UU   ....
0008   UU UU 00 00   ....
000C   00 00 00 00   ....
0010   00 AE 00 00   ....
          X XX XX    ....
          C FF FC    ....
          X 00 UU    ....
          X XX XX    ....
```

Figure 6-8. Memory Window with Shortcut Menu

Use the options from the **Memory Window Shortcut** menu to perform these memory functions:

Set Base Address

Choose this option to set the first memory address to display in the Memory window.

Show as Hex and ASCII

Choose this option to display memory values in both hex and ASCII formats.

Show as Hex Only

Choose this option to display memory values in hex format only, allowing more bytes per row.

Use these keys to navigate in the Memory window:

- Press the up arrow (↑) to scroll the Memory window up one line.
- Press the down arrow (↓) to scroll the Memory window down one line.
- Press the HOME key to scroll the Memory window to memory address $0000.
- Press the END key to scroll the Memory window to the last address in the memory map.
- Press the PAGE UP key to scroll the Memory window up one page.
- Press the PAGE DOWN key to scroll the Memory window down one page.
- Press the F1 key to show the help contents topic.
- Press the ESCAPE (ESC) key to move the cursor to the command line of the Status window.

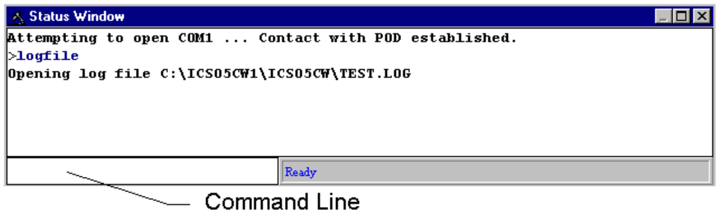
6.9 Status Window

The Status window (**Figure 6-9**) accepts ICS08GPW commands entered on the command line, executes them, and returns an error message or status update message, as in the message area of the window.

The Status window message area displays all ICS08GPW commands (including implemented ICS08GPW menu options and toolbar buttons) and command results.

Use the scroll controls on the right side of the Status window to view previous commands or use these keys to navigate within the message area:

- Press the up arrow (↑) key to scroll the window up one line.
- Press the down arrow (↓) key to scroll the window down one line.
- Press the HOME key to scroll the window to the first status line.
- Press the END key to scroll the window to the last status line.
- Press the PAGE UP key to scroll the window up one page.
- Press the PAGE DOWN key to scroll the window down one page.
- Press the F1 key to display the help contents topic.

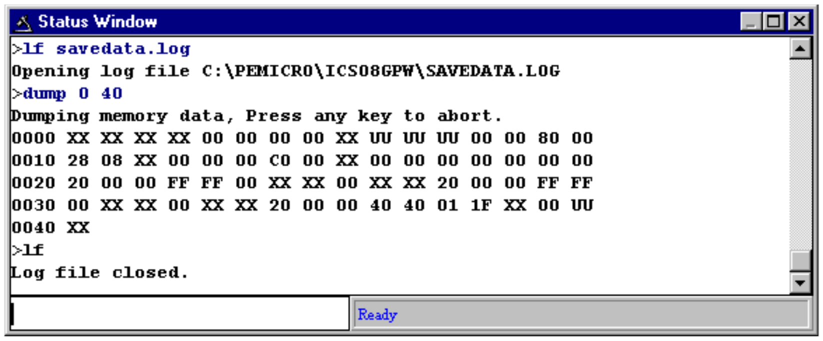


```
Attempting to open COM1 ... Contact with POD established.
>logfile
Opening log file C:\ICS05CW1\ICS05CW\TEST.LOG
```

Figure 6-9. Status Window

To save the information displayed in the Status window, enable logging:

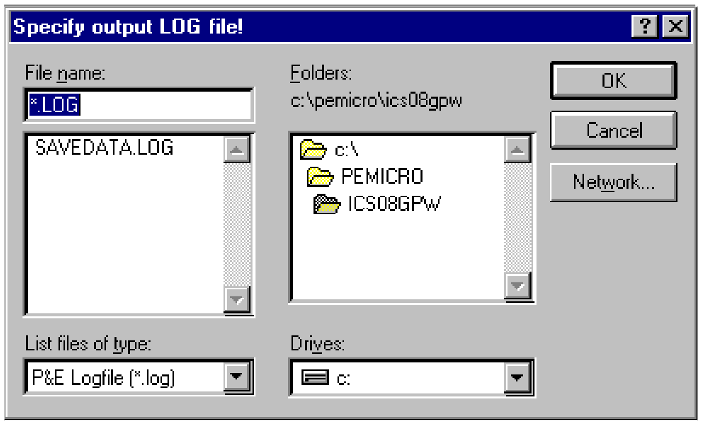
- Choose the **Start Logfile** option from the ICS08GPW file menu or enter the LF command in the Status window command line (**Figure 6-10**).



```
>lf savedata.log
Opening log file C:\PEMICRO\ICS08GPW\SAVEDATA.LOG
>dump 0 40
Dumping memory data, Press any key to abort.
0000 XX XX XX XX 00 00 00 00 XX UU UU UU 00 00 80 00
0010 28 08 XX 00 00 00 C0 00 XX 00 00 00 00 00 00 00
0020 20 00 00 FF FF 00 XX XX 00 XX XX 20 00 00 FF FF
0030 00 XX XX 00 XX XX 20 00 00 40 40 01 1F XX 00 UU
0040 XX
>lf
Log file closed.
```

Figure 6-10. Results of Entering the LF Command in the Status Window

- The **Specify output LOG file** dialog (**Figure 6-11**) opens.



Specify output LOG file!

File name: *.LOG

SAVEDATA.LOG

Folders: c:\pemicro\ics08gpw

c:\
PEMICRO
ICS08GPW

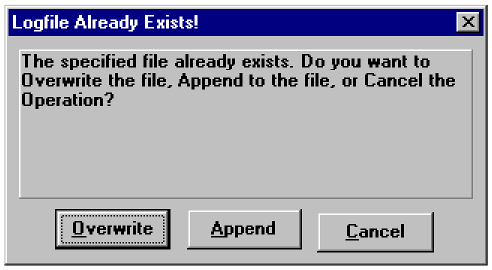
List files of type: P&E Logfile (*.log)

Drives: c:

Figure 6-11. Specify Output LOG File! Dialog Window

- In the dialog, choose a path and filename for the logfile. Press OK to create the file (or CANCEL to close the dialog without opening the logfile).

- If a logfile that already exists is chosen, the **Logfile Already Exists** window (**Figure 6-12**) appears, asking if the user wants to overwrite the existing file or append the status messages to the end of the existing file. Choose OVERWRITE or APPEND to begin logging in the file or CANCEL to close the dialog without opening the logfile.


Logfile Already Exists!

The specified file already exists. Do you want to Overwrite the file, Append to the file, or Cancel the Operation?

Figure 6-12. Logfile Already Exists Window

- Status window messages are added to the logfile while logging is enabled.

To end logging, choose the **End Logfile** option from the ICS08GPW file menu or enter the LF command in the ICS08GPW Status window command line.

6.10 CPU Window

The CPU window displays the current register values.

6.10.1 Changing Register Values

Use the CPU window (**Figure 6-13**) or its shortcut menu options to view and modify the current state of registers within the CPU.

- To change CPU register values using the shortcut menu options, position the cursor in the CPU window and click the right mouse button. Choose the option from the shortcut menu shown on the right of **Figure 6-13**. Enter the new value in the dialog and press OK to close the dialog and save the new value.

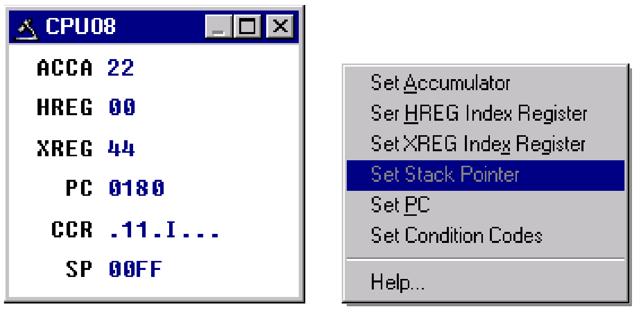


Figure 6-13. CPU08 Window with Shortcut Menu

- To change CPU register value in the CPU window:
  - To change the CPU accumulator (ACCA), HREG index register, XREG index register, and program counter (PC) values from the CPU window, double click on the value and enter the new value in the dialog box. Press OK to close the dialog and save the new value.
  - To change the CPU CCR values, double click the CCR value in the CPU window to open the **Change CCR** dialog (**Figure 6-14**). Change the H, I, N, V, Z, or C CCR bits by pressing the button below each to toggle condition code register bits between 1 (on) and 2 (off). Press OK to close the dialog and save the values.

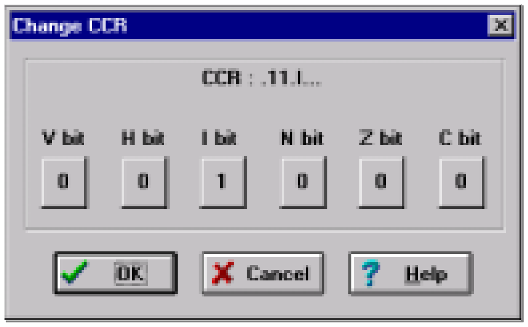


Figure 6-14. Change CCR Dialog Window

  - To change the CPU stack pointer (SP) value from the CPU window, position the cursor in the CPU window and click the right mouse button to open the CPU shortcut menu. Choose the **Set Stack Pointer** option. In the **Change SP Value** dialog, enter the new value. Press OK to close the dialog and save the value.

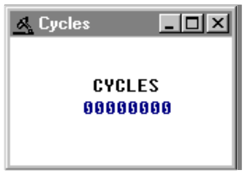
6.10.2 CPU Window Keyboard Commands

Use these keyboard commands to navigate in the CPU window:

- Press the F1 key to shows the help contents topics.
- Press the ESCAPE (ESC) key to move the cursor to the command line of the Status window.

6.11 Cycles Window

Use the Cycles window (**Figure 6-15**) to view the number of processor cycles that passed during execution of code in the simulator. This is valuable when counting the number of cycles that a section of code requires. To calculate the timing of code for a device, take the number of cycles shown in the window and multiply by the amount of time that a cycle represents in the target system. For a 4-MHz HC08, the time per cycle is 250 ns.



CYCLES
00000000

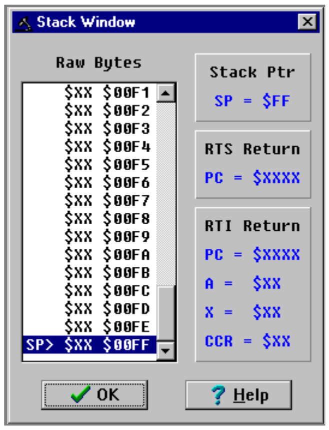
Figure 6-15. Cycles Window

6.12 Stack Window

Use the Stack window (**Figure 6-16**) to view:

- Values that have been pushed on the stack
- The stack pointer value
- CPU results if an RTI (return-from-interrupt) or RTS (return-from-subroutine) instruction is executed at that time.

To display the stack window, enter the STACK command in the ICS08GPW Status window command line.



```
Raw Bytes                 Stack Ptr
     $XX  $00F1           SP = $FF
     $XX  $00F2
     $XX  $00F3           RTS Return
     $XX  $00F4           PC = $XXXX
     $XX  $00F5
     $XX  $00F6           RTI Return
     $XX  $00F7           PC = $XXXX
     $XX  $00F8           A =  $XX
     $XX  $00F9           X =  $XX
     $XX  $00FA           CCR = $XX
     $XX  $00FB
     $XX  $00FC
     $XX  $00FD
     $XX  $00FE
SP>  $XX  $00FF
```

Figure 6-16. Stack Window

NOTE: *The value xx means that the memory location is uninitialized and indeterminate. The value W means that it is unimplemented, invalid memory.*

6.12.1 Interrupt Stack

During an interrupt, the Stack window displays:

- The interrupt stack
- Data values in the stack
- 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.

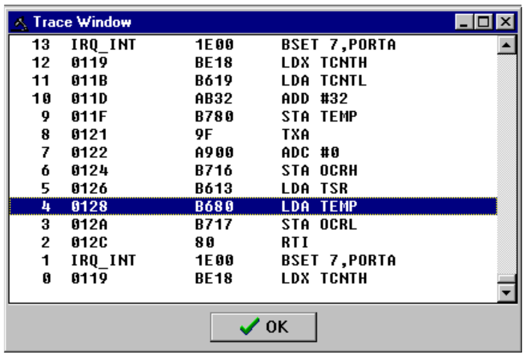
6.12.2 Subroutine Stack

During execution of a subroutine, the stack window displays the subroutine stack that indicates the restored state of the CPU upon return from a subroutine.

NOTE: *M68HC08 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. It is important to know whether program execution is in an interrupt or in a subroutine to know which stack data interpretation is valid.*

6.13 Trace Window

Use the Trace window (**Figure 6-17**) to view instructions captured while tracing is enabled.



```
13   IRQ_INT    1E00   BSET 7,PORTA
12   0119       BE18   LDX TCNTH
11   011B       B619   LDA TCNTL
10   011D       AB32   ADD #32
 9   011F       B780   STA TEMP
 8   0121       9F     TXA
 7   0122       A900   ADC #0
 6   0124       B716   STA OCRH
 5   0126       B613   LDA TSR
 4   0128       B680   LDA TEMP
 3   012A       B717   STA OCRL
 2   012C       80     RTI
 1   IRQ_INT    1E00   BSET 7,PORTA
 0   0119       BE18   LDX TCNTH
```

Figure 6-17. Trace Window

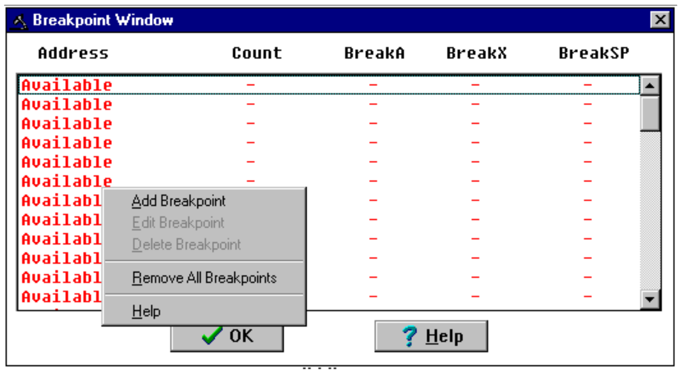
To display the Trace window, enter the SHOWTRACE command in the command line of the ICS08GPW Status window.

To enable or disable tracing, enter the TRACE command. If tracing is off, the command will toggle tracing on; if tracing is on, the command toggles tracing off.

The trace buffer is a 1024 instruction circular buffer that contains all addresses that have been executed. When the trace window displays instructions, it disassembles instructions at the addresses stored in the trace buffer. For this reason, the tracing function cannot be used for self-modifying code. If a buffer slot does not have an address stored in it, the trace window displays the phrase `No Trace Available`. The number in the beginning of a trace line is the slot number in the trace buffer. The slot number is an offset for the instruction in that slot compared to the current instruction executing (slot number = 0).

6.14 Breakpoint Window

Use the Breakpoint window (**Figure 6-18**) to view all breakpoints currently set in the current debugging session, and to add, modify, or delete breakpoints. A maximum of 64 breakpoints can be set.



Address	Count	BreakA	BreakX	BreakSP
Available	–	–	–	–
Available	–	–	–	–
Available	–	–	–	–
Available	–	–	–	–
Available	–	–	–	–
Available	–	–	–	–
Availabl		–	–	–
Availabl		–	–	–
Availabl		–	–	–
Availabl		–	–	–
Availabl		–	–	–
Availabl		–	–	–

Figure 6-18. Breakpoint Window with Shortcut Menu

To display the Breakpoint window, enter the SHOWBREAKS command in the ICS08GPW Status window command line.

If a breakpoint slot is empty, the word `available` appears under the **Address** column.

6.14.1 Adding a Breakpoint

To add a breakpoint, with the cursor in the Breakpoint window, click the right mouse button to open the **Breakpoint Shortcut** menu. Select the **Add Breakpoint** option from the shortcut menu. In the **Edit Breakpoint** dialog (**Figure 6-19**), enter the address for the new breakpoint in the **Address** text box. Press the OK button to close the dialog and save the new breakpoint.

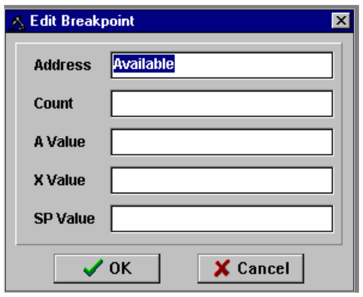


Figure 6-19. Edit Breakpoint Dialog Window

Qualify the breakpoint using these qualifiers:

- **Count** — Enter the number of times the address will be reached before breaking, for instance, break after *n* times (the default is *n* = 1).
- **Accumulator value** —Enter the number the accumulator value must reach before breaking, for instance, break if address and A = *n*.
- **X index register value** — Enter the number the index register value must reach before breaking, for instance, break if address and X = *n*.
- **Stack pointer value** — Enter the number the stack pointer value must reach before breaking, for instance, break if address and SP = *n*.

6.14.2 Editing a Breakpoint

To edit a breakpoint or view address information, double click on any empty breakpoint slot in the Breakpoint window listbox. The **Edit Breakpoint** dialog window (**Figure 6-19**) displays address information for the empty breakpoint slot. Enter the appropriate address and other conditional qualifiers and press the OK button to exit.

In the Breakpoint window, select the breakpoint to edit. Then use one of the following methods to open the **Breakpoint Shortcut** menu and edit the breakpoint:

- Click the right mouse button to open the **Breakpoint Shortcut** menu and select the **Edit Breakpoint** menu option.
- Press the INSERT key.
- Double click on the breakpoint in the listbox. In the **Edit Breakpoint** dialog window, enter the new breakpoint address and conditional qualifiers. Press the OK button to close the dialog and store the new settings (or press the CANCEL button to close the dialog without saving new settings).

6.14.3 Deleting a Breakpoint

In the Breakpoint window, choose the breakpoint to delete, and use one of these methods to delete the breakpoint:

- Click the right mouse button to open the **Breakpoint Shortcut** menu and select the **Delete Breakpoint** menu option.
- Press the DELETE key to remove the selected breakpoint from the breakpoint list.

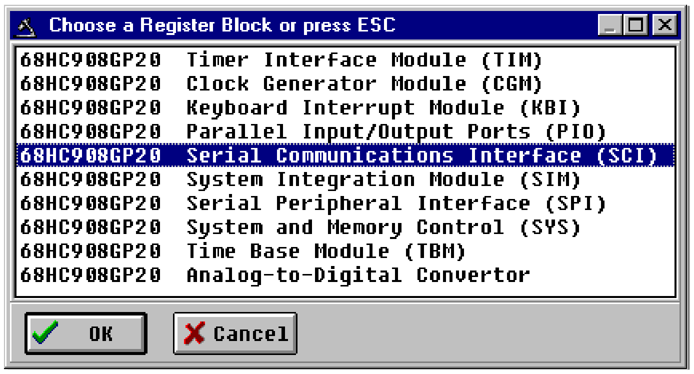
Press the OK button to close the Breakpoint window and store the changes or press CANCEL to close the window without saving the changes.

6.14.4 Removing All Breakpoints

In the Breakpoint window, click the right mouse button to open the **Breakpoint Shortcut** menu. Choose the **Remove All Breakpoints** menu option to clear all breakpoints. Press the OK button to store changes and close the Breakpoint window or press the CANCEL button to close the Breakpoint window without saving changes.

6.15 Register Block Window

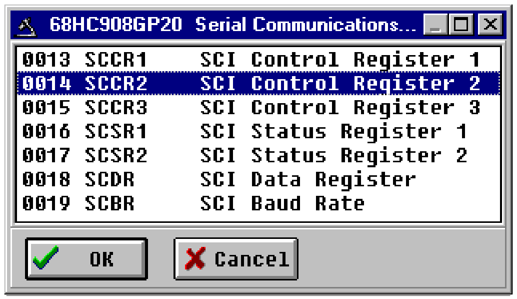
The Register Block window (**Figure 6-20**) can be opened by pressing the REGISTER FILES button on the ICS08GPW toolbar or by entering the R command in the Status window command line.



```
68HC908GP20  Timer Interface Module (TIM)
68HC908GP20  Clock Generator Module (CGM)
68HC908GP20  Keyboard Interrupt Module (KBI)
68HC908GP20  Parallel Input/Output Ports (PIO)
68HC908GP20  Serial Communications Interface (SCI)
68HC908GP20  System Integration Module (SIM)
68HC908GP20  Serial Peripheral Interface (SPI)
68HC908GP20  System and Memory Control (SYS)
68HC908GP20  Time Base Module (TBM)
68HC908GP20  Analog-to-Digital Convertor
```

Figure 6-20. Register Block Window

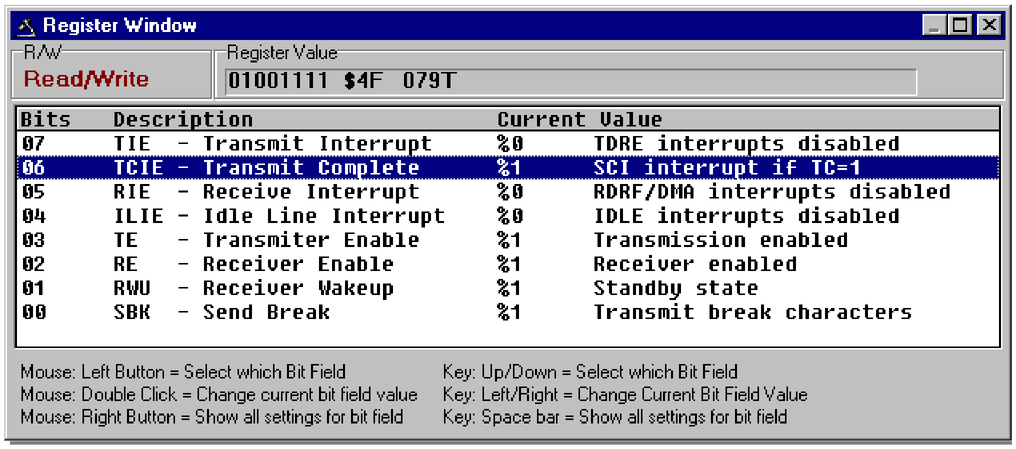
If register files have been installed on the host computer, selecting a block brings up the Register Files window (**Figure 6-21**), which shows a list of the files, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.



```
0013 SCCR1     SCI Control Register 1
0014 SCCR2     SCI Control Register 2
0015 SCCR3     SCI Control Register 3
0016 SCSR1     SCI Status Register 1
0017 SCSR2     SCI Status Register 2
0018 SCDR      SCI Data Register
0019 SCBR      SCI Baud Rate
```

Figure 6-21. Register Files Window

Selecting a file brings up the Register window (**Figure 6-22**), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.



```
01001111 $4F 079T

Bits   Description                  Current Value
07     TIE  - Transmit Interrupt    %0      TDRE interrupts disabled
06     TCIE - Transmit Complete     %1      SCI interrupt if TC=1
05     RIE  - Receive Interrupt     %0      RDRF/DMA interrupts disabled
04     ILIE - Idle Line Interrupt   %0      IDLE interrupts disabled
03     TE   - Transmiter Enable     %1      Transmission enabled
02     RE   - Receiver Enable       %1      Receiver enabled
01     RWU  - Receiver Wakeup       %1      Standby state
00     SBK  - Send Break            %1      Transmit break characters
```

Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Figure 6-22. Register Window

6.16 Entering Debugging Commands

To enter commands in the ICS08GPW Status window command line:

1. Type the command and its options and/or arguments in the text area (the command line).
2. When the command is complete, press the ENTER key to execute the command.
3. If the command has not been entered correctly, the Status window will display a message such as `Invalid command or parameter`. If the command has been entered correctly, other prompts, messages, or data appropriate to the command entered are displayed in the Status window text area.
4. After the command has been executed, a new blank line appears in the command line.
5. The ICS08GPW maintains a command buffer containing the commands and system responses to the commands entered on the command line. Use the mouse or keyboard commands to sequence forward or backward through the command buffer.

For more instructions on using the ICS08GPW command set, see **Section 7. Debugging with ICS08GPW**.

6.17 ICS08GPW Toolbar

The ICS08GPW toolbar (**Figure 6-23**) provides a number of convenient shortcut buttons that duplicate the function of the most frequently used menu options. A tool tip or label pops up when the mouse button lingers over a toolbar button, identifying the button's function.

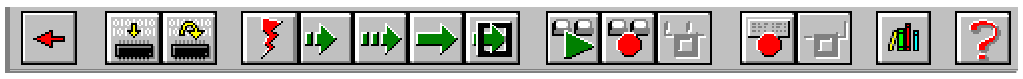


Figure 6-23. ICS08GPW Toolbar

Table 6-2 identifies and describes the ICS08GPW toolbar buttons.

Table 6-2. ICS08GPW Toolbar Buttons

Icon	Button Label	Button Function
	Back to Editor	Return to the WinIDE editor.
	Load S19 File	Open the **Specify S19 File** *to Load* dialog to choose an S19 file.
	Reload Current S19	Reload the last (most currently loaded) S19 file.
	Reset	Simulate a reset of the MCU and set the program counter (PC) to the contents of the reset vector (does not start execution of user code).
	Step	Execute the STEP command.
	Multiple Step	Execute the STEPFOR command.
	Go	Execute the GO command.
	Stop	Stop execution of assembly commands.
	Play Macro	Open the **Specify Macro File to Execute** dialog to choose a macro to execute.
	Record Macro	Open the **Specify Macro File to Record** dialog to enter a filename for the macro.
	Stop Macro Function	Stop recording the macro.
	Open Logfile	Execute the LOGFILE command; opens the **Specify Output Logfile** dialog.
	Close Logfile	Execute the LOGFILE command; closes the current logfile.
	Register Files	Open the Register Block window.
	Help	Display ICS08GPW Help.

6.18 ICS08GPW Menus

Table 6-3 summarizes WinIDE menus and options.

Table 6-3. ICS08GPW Menus and Options Summary

Menu	Option	Description
File	Load S19 File	Open the **Specify S19 File to Open** dialog to choose S19 file.
	Reload Last S19	Reload the last S19 file used, or (if none loaded) display the **Specify S19 File to Open** dialog.
	Play Macro	Open the **Specify Macro File to Execute** dialog.
	Record Macro	Open the **Save As** dialog.
	Stop Macro	Close the macro or script file.
	Open Logfile	Executes the LOGFILE command.
	Close Logfile	Executes the LOGFILE command.
	Exit	Close the ICS08GPW simulator.
Execute	Reset Processor	Reset the emulation MCU and program counter to the contents of the reset vector.
	Step	Execute the STEP command.
	Multiple Step	Execute the STEPFOR command.
	Go	Execute the GO command.
	Stop	Stop code execution.
	Repeat Command	Repeat the last command entered in the Status window command line.
Windows	Code 1	Toggles the Code 1 window open/closed.
	Code 2	Toggles the Code 2 window open/closed.
	Memory	Toggles the Memory window open/closed.
	Variables	Toggles the Variables window open/closed.
	Cycles	Toggles the Cycles window open/closed.
	Status	Toggles the Status window open/closed.
	CPU	Toggles the CPU window open/closed.
	Change Colors	Opens the **Changes Windows Colors** dialog.
	Chip Window	Show connection to hardware pod, yes or no.
	Reload Desktop	Executes the LOADDESK command to load the desktop settings from a file.
	Save Desktop	Executes the SAVEDESK command to save the current desktop settings to a file.

6.19 File Options

Use the **ICS08GPW File** menu options to load, reload, open, or close files, play or record macros, or exit the ICS08GPW application.

To perform a **File** operation, click once on the **File** menu (**Figure 6-24**) title to open the menu. Click on the option to execute.

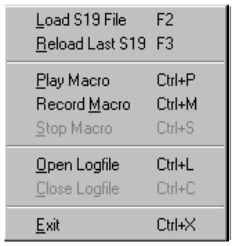
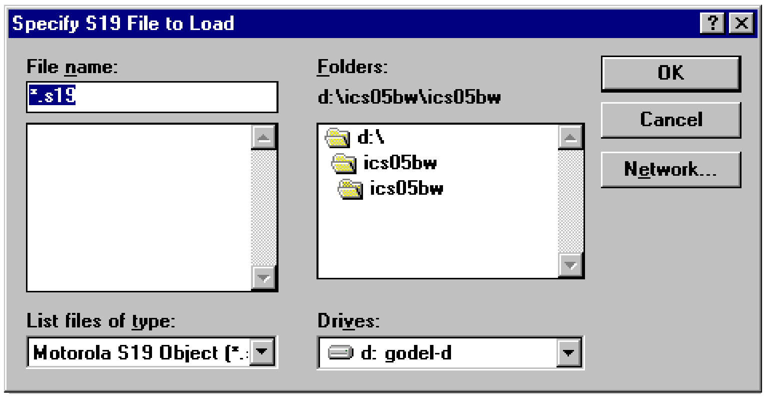


Figure 6-24. ICS08GPW File Menu

The following topics describe and explain the **ICS08GPW File** operations and dialogs.

6.19.1 Load S19 File

Select the **Load S19 File** option from the **File** menu to open the **Specify S19 File to Load** dialog (**Figure 6-25**). If the S19 file is not in the default directory, choose a filename, drive/directory, and network path of an object file or source file to load in the Debugger main window. Also use this option to load SLD map files.



File name: *.s19

Folders: d:\ics05bw\ics05bw

d:\
ics05bw
ics05bw

List files of type: Motorola S19 Object (*.:

Drives: d: godel-d

Figure 6-25. Specify S19 File to Load Dialog Window

To load an S19 or .MAP file, choose the **Load S19 File** option from the **File** menu to open the **Specify S19 File to Load** dialog. Choose the path and filename and press OK to open the selected file in the ICS08GPW (or press CANCEL to close the dialog without making a selection).

Alternatives: Press the F2 function key, click the LOAD S19 FILE toolbar button, or enter the LOAD command and filename and other arguments in the Status window command line.

6.19.2 Reload Last S19

Select the **Reload Last S19** option from the **File** menu to open the **Specify S19 File to Load** dialog (**Figure 6-25**) and select the most recently opened S19 or .MAP file to open in the Debugger main window. Follow the procedure for loading an S19 file (**6.19.1 Load S19 File**).

Alternatives: Press the F3 function key or click the RELOAD CURRENT S19 toolbar button. These are the keyboard equivalents to choosing the **File - Reload Last S19** menu option.

6.19.3 Play Macro

Select the **Play Macro** option from the **File** menu to open the **Specify MACRO File to Execute** dialog (**Figure 6-26**) to specify a macro filename and drive/directory path to play.

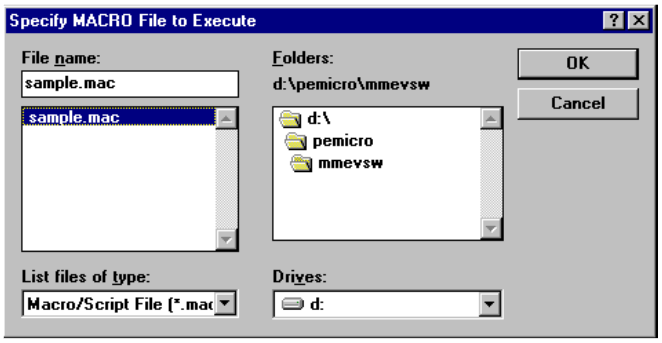


Figure 6-26. Specify MACRO File to Execute Dialog Window

Alternatives: Press the CTRL + P key combination or click the PLAY MACRO toolbar button These are the keyboard equivalents to choosing the **File - Play Macro** menu option.

6.19.4 Record Macro

Select the **Record Macro** option from the **File** menu to open the **Specify MACRO File to Record** dialog (**Figure 6-27**) and specify a macro filename and drive/directory path to record.

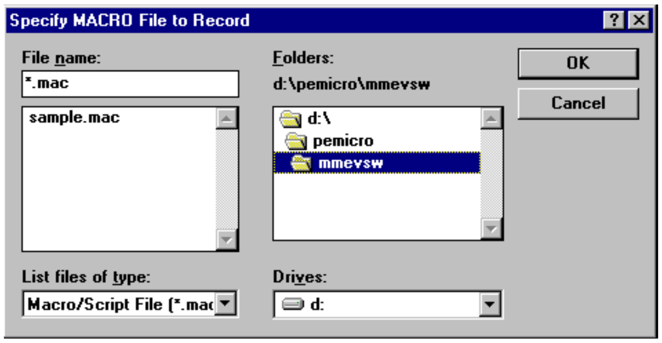


Figure 6-27. Specify MACRO File to Record Dialog Window

Debugger window will be recorded in the macro file and can be repeated by playing back the macro using the **File - Play Macro** menu option.

Alternatives: Press the CTRL + M key combination or click the RECORD MACRO toolbar button These are the keyboard equivalents to choosing the **File - Record Macro** menu option.

6.19.5 Stop Macro

Select the **Stop Macro** option from the **File** menu (or press the CTRL + S key combination) to stop the active macro's execution.

Alternatives: Press the CTRL + S key combination or click the STOP MACRO toolbar button. These are the keyboard equivalents to choosing the **File - Stop Macro** menu option.

6.19.6 Open Logfile

Select the **Open Logfile** option from the **File** menu to open the **Specify Output LOG File** dialog (**Figure 6-28**). Use this dialog to specify a logfile name and directory/drive path in which to save output log information for the current debugging session.

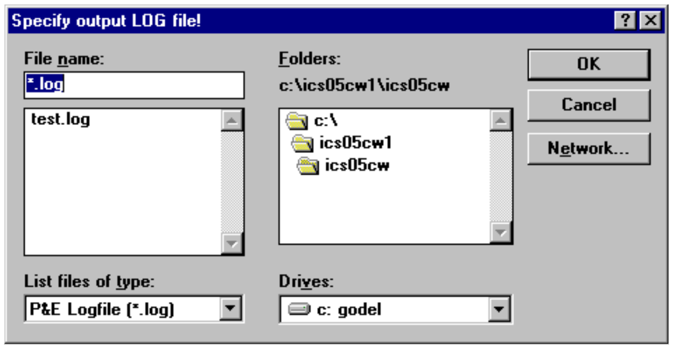
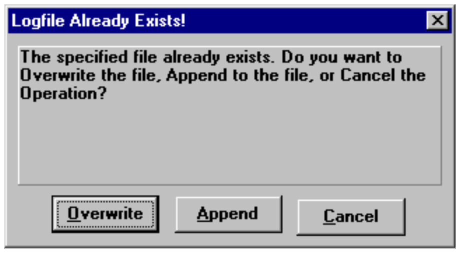


Figure 6-28. Specify Output LOG File Dialog Window

If the specified logfile exists, a message box (**Figure 6-29**) prompts the user to:

- Overwrite the existing logfile with current logging information.
- Append the current logging information at the end of the existing logfile.
- Cancel the open LOGFILE command without saving logging information.

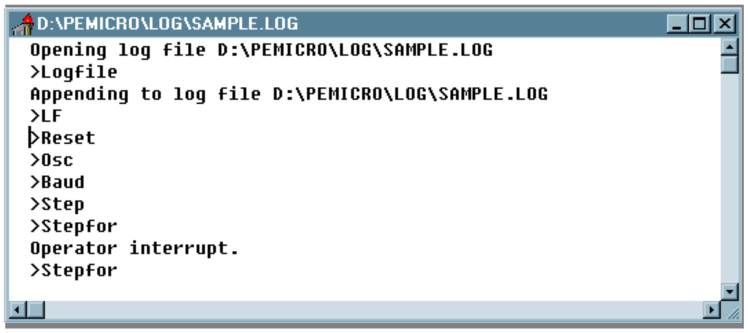


The specified file already exists. Do you want to Overwrite the file, Append to the file, or Cancel the Operation?

Figure 6-29. Logfile Already Exists Dialog Window

The open logfile does not appear in the Debugger window. To enable logging in a currently active logfile, the LF (LOGFILE) command must be executed as well; otherwise, no logging occurs in the open logfile.

The LF command begins logging of commands and responses to the specified external file. While logging is enabled, any line appended to the command log window also is written to the logfile (**Figure 6-30**). Logging to the external file continues until another LF command stops logging and closes the logfile.



```
Opening log file D:\PEMICRO\LOG\SAMPLE.LOG
>Logfile
Appending to log file D:\PEMICRO\LOG\SAMPLE.LOG
>LF
>Reset
>Osc
>Baud
>Step
>Stepfor
Operator interrupt.
>Stepfor
```

Figure 6-30. Sample Output Logfile

View the logfile in the WinIDE editor or in any program that displays text files.

Alternatives: Press the CTRL + L key combination or click the OPEN LOGFILE toolbar button These are the keyboard equivalents to choosing the **File - Open Logfile** menu option.

6.19.7 Close Logfile

Choose **Close Logfile** from the **File** menu to stop logging and close the active logfile.

Alternatives: Type CTRL + C or click the CLOSE LOGFILE button on the toolbar, or enter the LF command in the Status window command line. These are the keyboard equivalents to choosing the **File - Close Logfile** menu option.

6.19.8 Exit

Choose **Exit** from the **File** menu to close the debugger application.

Alternative: Type CTRL + X to exit the debugger application and close the subordinate and main windows. This is the keyboard equivalent to choosing the **File - Exit** menu option.

6.20 ICS08GPW Execute Options

Use the **ICS08GPW Execute** menu options to reset the emulation microcontroller and perform debugger routines.

To perform an execute operation, select **Execute** in the menu bar to open the **Execute** menu (**Figure 6-31**). Click on an option to perform the operation.

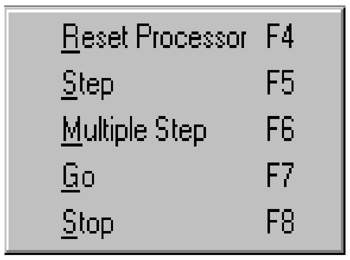


Figure 6-31. ICS08GPW Execute Menu

6.20.1 Reset Processor

Choose **Reset Processor** from the **Execute** menu to send the RESET command to the emulation MCU and reset the PC to the contents of the reset vector.

Alternative: Press the F4 function key. This is the keyboard equivalent of the **Execute - Reset Processor** menu option.

6.20.2 Step

Choose **Step** from the **Execute** menu to send the single step (TRACE) command to the MCU. The Step command executes a single instruction, beginning at the current PC address value.

NOTE: *The Step command does not execute instructions in real-time, so timer values cannot be tested using this command.*

Alternative: Press the F5 function key. This is the keyboard equivalent to choosing the **Execute - Step** menu option.

6.20.3 Multiple Step

Choose **Multiple Step** from the **Execute** menu to send the STEPFOR command to the MCU. The STEPFOR command begins continuous instruction execution, beginning at the current PC address value and continuing until any key is pressed.

NOTE: *The Multiple Step command does not execute instructions in real-time, so timer values cannot be tested using this command.*

Alternative: Press the F6 function key. This is the keyboard equivalent to choosing the **Execute - Multiple Step** menu option.

6.20.4 Go

Choose **Go** from the **Execute** menu to start execution of code in the ICS08GPW at the current address. Code execution continues until a stop command is entered, a breakpoint is reached, or an error occurs.

Alternative: Press the F7 function key. This is the keyboard equivalent to choosing the **Execute - Go** menu option.

6.20.5 Stop

Choose **Stop** from the **Execute** menu to stop program execution and update the ICS08GPW simulator windows with current data.

Alternative: Press the F8 function key. This is the keyboard equivalent to choosing the **Execute - Stop** menu option.

6.20.6 Repeat Command

Choose **Repeat** command from the **Execute** menu to repeat the execution of the last command entered in the **Status** window command line.

Alternative: Press the F9 function key. This is the keyboard equivalent to choosing the **Execute - Repeat Command** menu option.

6.21 ICS08GPW Window Options

Use the **Window** menu options to change the window displays in the ICS08GPW simulator.

To make changes to the windows, select **Window** in the menu bar to open the **Window** menu (**Figure 6-32**). Click on an option to perform the operation.

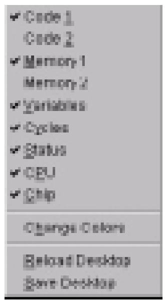


Figure 6-32. ICS08GPW Window Menu

6.21.1 Open Windows

The **Window** menu options itemize the source file windows that can be opened in the ICS08GPW. A check beside the window name toggles that window display to *on*. Uncheck the window name to close the window; check the window name to open it.

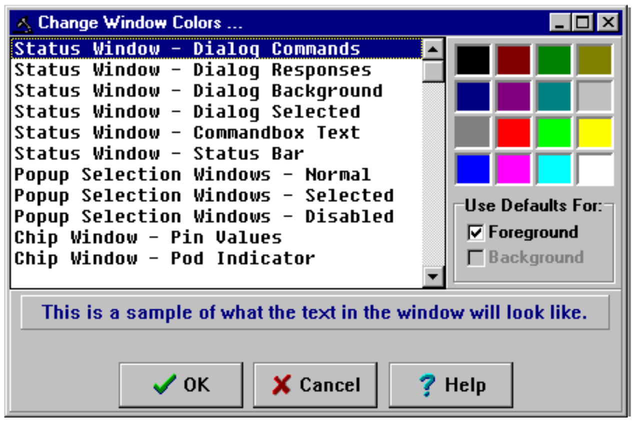
For example, **Figure 6-32** indicates that all ICS08GPW windows are open except Code 1. To open the Code 1 window, click on the **Code 1** option. To close it, click on the **Code 1** option to remove the check and close the window.

6.21.2 Change Colors

Choose **Change Colors** from the **Windows** menu to open the **Change Window Colors** dialog window (**Figure 6-33**).

The **Change Window Colors** dialog window displays the color settings for the ICS08GPW debugger windows or window components. To see the current settings, select the window or window element from the list on the left. To change the foreground or background color setting for this window or element, uncheck the **Use Defaults for Foreground/Background** checkbox. Use the left mouse button to select a foreground color, or use the right mouse button to select a background color. Press the OK button to save the color changes or press the CANCEL button to close the dialog without saving changes.

Some window items allow only the foreground or background to be changed.


Change Window Colors ...

```
Status Window - Dialog Commands
Status Window - Dialog Responses
Status Window - Dialog Background
Status Window - Dialog Selected
Status Window - Commandbox Text
Status Window - Status Bar
Popup Selection Windows - Normal
Popup Selection Windows - Selected
Popup Selection Windows - Disabled
Chip Window - Pin Values
Chip Window - Pod Indicator
```

Use Defaults For:
[✓] Foreground
[ ] Background

This is a sample of what the text in the window will look like.

Figure 6-33. Change Window Colors Dialog Window

6.21.3 Reload Desktop

Choose **Reload Desktop** from the **Windows** menu to reload the stored configuration for the current project.

This option is useful for restoring desktop windows to their stored sizes and locations after making changes. To make changes permanent, choose the **Save Desktop** option. The new window sizes and locations will be written over the old settings and stored with other project files.

6.21.4 Save Desktop

Choose **Save Desktop** from the **Windows** menu to save the current configuration of the desktop and the position and size of the windows in the ICS08GPW simulator.

Section 7. Debugging with ICS08GPW

7.1 Contents

7.2 Introduction. 153

7.3 ICS08GPW Debugging Command Syntax. 154

7.4 Command Set Summary 155

7.2 Introduction

This chapter consists of:

- A logical overview of the ICS08GPW debugging command set
- An explanation of rules for using the command set, including command syntax and arguments
- A summary of commands by type and function

The ICS08GPW simulator command set consists of commands for simulating, debugging, analyzing, and programming microcontroller programs.

Use the commands to:

- Initialize emulation memory
- Display and store data
- Debug user code
- Control the flow of code execution

7.3 ICS08GPW Debugging Command Syntax

A command is a line of ASCII text entered from the computer keyboard. For ICS08GPW debugging commands, enter the command and its arguments in the ICS08GPW Status window command line. Press ENTER to terminate each line and activate the command. The typical command syntax is:

```
command [<argument>]...
```

Where:

`command`	A command name in upper- or lower-case letters
`<argument>`	An argument indicator; when arguments are italicized, they represent a placeholder for the actual value entered; when not italicized, they indicate the actual value to enter. **Table 7-1** explains the possible argument values.

In command syntax descriptions:

[ ]	Brackets enclose optional items.
|	A vertical line means *or.*
...	An ellipsis means the preceding item can be repeated.
( )	Parentheses enclose items only for syntactical purposes.

Except where otherwise noted, numerical values in debugging command examples are hexadecimal.

7.4 Command Set Summary

Table 7-1 lists the argument types used for commands. **Table 7-2** lists the commands alphabetically and summarizes their functions. For detailed descriptions of the individual commands, refer to **Section 10. Debugging Command Set**.

Table 7-1. Argument Types

Type	Syntax Indicators	Explanation
Numeric	<n>, <rate>, <data>, <signal>, <frame>, <frequency>, <clips>, <count>, <value>	Hexadecimal values, unless otherwise noted. For decimal values, use the prefix ! or the suffix T. For binary values, use the prefix % or the suffix Q. Example: 64 = !100 = 100T = %1100100 = 1100100Q.
Address	<address>	Four or fewer hexadecimal digits, with leading zeros when appropriate. If an address is decimal or binary, use a prefix or suffix, per the explanation of numeric arguments.
Range	<range>	A range of addresses or numbers. Specify the low value, then the high value, separated by a space. Use leading zeros if appropriate.
Symbol	<symbol>, <label>	Symbols of ASCII characters, usually symbols from source code
Filename	<filename>	The name of a file, in DOS format, eight or fewer ASCII characters. Include an optional extension (three or fewer characters) after a period. If the file is not in the current directory, precede the name with one or more directory names.
Operator	<op>	+ (add); – (subtract); * (multiply); or / (divide)

Table 7-2. ICS08GPW Command Overview (Sheet 1 of 7)

Command	Description
A	Set the accumulator to specified value and display new value in CPU window (identical to the ACC command).
ACC	Set the accumulator to specified value and display new value in CPU window (identical to the A command).
ADCLR	Clear A/D input buffer.
ADDI	Enter input to A/D.
ASM	Assemble M68HC08 instruction mnemonics and place resulting machine code in memory at the specified address.
BELL	Sound PC bell the specified number of times.
BF	Fill a block of memory with a specified byte, word, or long value.
BR	Display or set instruction breakpoint to specified values or at cursor location.
BREAKA	Set accumulator breakpoint to halt code execution when the accumulator value equals the specified value.
BREAKHX	Set an HX register breakpoint to halt code execution when the value of the HX register equals the specified value.
BREAKSP	Set stack pointer breakpoint to halt code execution when the SP equals the specified value.
C	Set or clear the C bit of the CCR.
CAPTURE	Specify location to be monitored for changes in value.
CAPTUREFILE CF	Open a capture file to record changed values.
CCR	Set the CCR in the CPU to the specified hexadecimal value.
CGMXCLK	Set CGMXCLK to bus clock relationship.
CHIPMODE	Set chip for simulation.
CLEARMAP	Remove the current MAP file from memory (identical to the NOMAP command).
CLEARSYMBOL	Remove all user-defined symbols from memory.
COLORS	Set simulator colors
CYCLES CY	Change the value of the cycles counter.
DASM	Disassemble machine instructions, display addresses and contents as disassembled instructions in the Code window.
DDRA	Assign the specified byte value to the port A data direction register (DDRA).

Table 7-2. ICS08GPW Command Overview (Sheet 2 of 7)

Command	Description
DDRB	Assign the specified byte value to the port B data direction register (DDRB).
DDRC	Assign the specified byte value to the port C data direction register (DDRC).
DDRD	Assign the specified byte value to the port D data direction register (DDRD).
DDRE	Assign the specified byte value to the port E data direction register (DDRE).
DUMP	Send contents of a block of memory to the Status window in bytes, words or longs.
EVAL	Evaluate a numerical term or expression and give the result in hexadecimal, decimal, octal, and binary format.
EXIT	Terminate the software and close all windows (identical to the QUIT command).
G	Start execution of code at the current PC address or at an optional specified address (identical to the GO and RUN commands).
GO	Start execution of code at the current PC address or at an optional specified address (identical to the G and RUN commands).
GOEXIT	Similar to GO command except that the target is left running without any breakpoints, and the debugger software is terminated.
GOMACRO	Execute the program in the simulator beginning at the address in the PC and continue until a keypress, Stop Macro command from the toolbar, breakpoint, or error occurs.
GONEXT	Execute from the current PC address until the next instruction is reached. Used to execute past a subroutine call or past intervening interrupts.
GOTIL	Execute code beginning at the PC address and continue until the PC contains the specified ending address or until a keypress, Stop Macro command from the toolbar, breakpoint, or error occurs.
GOTOCYCLE	Execute code beginning at the current PC and continue until the cycle counter is equal to or greater than the value specified.
H HREG	Set or clear the H (half-carry) bit in the CCR.
HELP	Open the ICS08GPW Help File.

Table 7-2. ICS08GPW Command Overview (Sheet 3 of 7)

Command	Description
HX	Set both bytes of the concatenated index register H:X to the specified value.
I	Set or clear the I bit of the CCR.
INFO	Display information about the line highlighted in the source window.
INPUTA	Set the simulated inputs to port A.
INPUTB	Set the simulated inputs to port B.
INPUTC	Set the simulated inputs to port C.
INPUTD	Set the simulated inputs to port D.
INPUTE	Set the simulated inputs to port E.
INPUTS	Show the simulated input values to ports A and B.
INT IRQ	View or assign the state value of the MCU IRQ pin.
LF	Open a new or specified external file to receive log entries of commands and responses in the Status window (identical to the LOGFILE command).
LISTOFF	Turn off screen listing of stepping information.
LISTON	Turn on screen listing of stepping information.
LOAD	Load S19 object file and associated MAP file into the ICS08GPW.
LOADALL	Execute both the LOAD and LOADMAP commands.
LOADDESK	Load the desktop settings for window positions, size, and visibility.
LOADMAP	Load a MAP file containing source level debug information into the ICS08GPW.
LOADV	Execute the LOAD command, then automatically execute the VERIFY command.
LOGFILE	Open a new or specify an existing external file to receive log entries of commands and responses from the Status window (identical to the LF command).
MACRO	Execute a macro file containing debug command sequences.
MACROEND	Close the macro file in which the debug command sequences are being saved.
MACROSTART	Open a macro file and save all subsequent debug commands to this file until closed by the MACROEND command during an active ICS08GPW session.

Table 7-2. ICS08GPW Command Overview (Sheet 4 of 7)

Command	Description
MACS	Bring up a window with a list of macros.
MAP	View information from the current MAP file stored in memory (identical to the SHOWMAP command).
MD MDI	Display the contents of memory locations in the Memory window beginning at the specified address.
MD2	Display the contents of memory locations in Memory window 2, beginning at the specified address.
MM MEM	Modify contents of memory beginning at the specified address, and/or select bytes, words, longs.
N	Set or clear the N bit of the CCR.
NOBR	Remove one or all of active breakpoints.
NOMAP	Remove the current MAP file from memory, forcing the ICS08GPW to show disassembly in the code windows instead of user source code (identical to the CLEARMAP command).
NOSYMBOL	Remove all user-defined symbols from memory; symbols defined in a loaded MAP file are not affected by the NOSYMBOL command.
PC	Assign the specified value to the MCU program counter.
POD	Attempt to connect with the ICS08GPW circuit board through the specified COM port; when successful, the POD command returns the current status of ports, reset, and IRQ pins on the ICS08GPW board and the board version number.
PORTA PRTA	Assign the specified value to the port A output register latches.
PORTB PRTB	Assign the specified value to the port B output register latches.
PORTC PRTC	Assign the specified value to the port C output register latches.
PORTD PRTD	Assign the specified value to the port D output register latches.
PORTE PRTE	Assign the specified value to the port E output register latches.
QUIET	Toggles refresh of memory-based windows.
QUIT	Terminate the ICS08GPW application and close all windows (identical to the EXIT command).
R	Open window for Register files and start interactive setup of system registers such as I/O, timer, COP.

Table 7-2. ICS08GPW Command Overview (Sheet 5 of 7)

Command	Description
REG	Display contents of CPU registers in the Status window (identical to the STATUS command).
REM	Enter comments in a macro file.
RESET	Simulate a reset of the MCU and set the PC to the contents of the reset vector. Does not start execution of user code.
RESETGO	Simulate a reset of the MCU, set PC to contents of the reset vector, and start execution from the PC address.
RUN	Start execution of code at the current PC current or specified address (identical to the G or GO command).
SAVEDESK	Save the desktop settings for the ICS08GPW program when it is first opened or for use with the LOADDESK command.
SCCLR	Clear SCI I/O buffers.
SCDI	Enter input to SCI.
SCDO	View outputs from SCI.
SCRIPT	Execute a macro file containing debug command sequences (identical to the MACRO command).
SHOW	Display the contents of memory locations in the Memory window beginning at the specified address (identical to the MD command).
SHOWBREAKS	Open window displaying breakpoints used in the current debug session, and allow modifying breakpoints.
SHOWCODE	Display code in the Code windows beginning at the specified address, but without changing the value of the PC.
SHOWMAP	View current MAP file (identical to the MAP command).
SHOWPC	Display code starting from address in the PC in the Code window.
SHOWTRACE	Display the Trace window with the last 1024 instructions executed since the TRACE command issued.
SIM08	Switch from in-circuit simulation (hardware pod connected) to stand-alone simulation (no pod connected).
SNAPSHOT	Save window data to the open logfile.
SP	Assign specified value to the stack pointer used by the CPU and display in the CPU window.
SPCLR	Clear SPI input/output.
SPDI	Enter SPI inputs.
SPDO	View SPI outputs.

Table 7-2. ICS08GPW Command Overview (Sheet 6 of 7)

Command	Description
SPFREQ	Set input clock for SPI slave.
SS	Step through a specified number of source code instructions, starting at the current PC address value, then halt.
ST	Step through a specified number of assembly instructions, starting at the current PC address value, then halt (identical to the STEP and T commands).
STACK	Open the HC08 Stack window showing the stack pointer value, data stored on the stack, and the results of RTS or RTI instruction.
STATUS	Display the contents of the CPU registers in the Status window (identical to the REG command).
STEP	Step through a specified number of assembly instructions, starting at the current program counter address value, then halt (identical to the ST and T commands).
STEPFOR	Execute instructions continuously, one at a time, starting at the current PC address and continuing until an error condition, breakpoint, or keypress occurs.
SOURCEPATH	Determine the path for source code that is not in the current working directory.
STEPTIL	Step through instructions starting at current PC address and continue until PC value reaches the specified address, or until keypress, breakpoint, or error occurs.
SYMBOL	View current symbols or create new symbols.
T	Step through a specified number of assembly instructions, starting at the current PC address, then halt (identical to the ST and STEP commands).
TRACE	Toggle tracing.
UPLOAD_SREC	Upload the content of the specified memory block (range) in S19 file format, display the contents in the Status window, and enter information into the current logfile.
V	Set or clear the V bit in the condition code register (CCR).
VAR	Display specified address and contents in the Variables window for viewing during code execution.
VERIFY	Compare the contents of program memory with an S-record file.
VERSION VER	Display program version and date.
WAIT	Delay simulator command execution by a specified number of cycles.

Table 7-2. ICS08GPW Command Overview (Sheet 7 of 7)

Command	Description
WHEREIS	Display value of the specified symbol.
X XREG	Set the X register to the specified value and display in the CPU window.
Z	Toggle the Z bit in the CCR.

Section 8. PROG08SW FLASH Programmer

8.1 Contents

8.2 Introduction. 164

8.3 Startup and Parameters 165

8.4 Programming Commands 166
8.4.1 BM — Blank-Check Module. 166
8.4.2 CM — Choose Module .08P 166
8.4.3 EM — Erase Module 167
8.4.4 PB — Program Bytes. 167
8.4.5 PM — Program Module. 167
8.4.6 SM — Show Module 167
8.4.7 SS — Specify S Record 168
8.4.8 UM — Upload Module 168
8.4.9 UR — Upload Range 168
8.4.10 VM — Verify Module 168
8.4.11 VR — Verify Range 168
8.4.12 QU — QUit 169
8.4.13 RE — REset Chip 169
8.4.14 HE — HElp 169

8.5 Programming Example 169

8.2 Introduction

PROG08SW is a programmer for FLASH memory internal to a CPU08 processor. The programmer communicates with the processor in monitor mode (MON08) using an M68HC08 serial programmer (SPGMR), which connects the serial port of a PC or compatible computer to a DB9 connector on the SPGMR.

The programmer software consists of three parts:

1. PROG08SW.EXE — Executable that runs on the host PC
2. 908_GP20.08P — Programming algorithm file for the MC68HC908GP20 MCU
3. 908_GP32.08P — Programming algorithm file for the MC68HC908GP32 MCU

PROG08SW provides for a set of general interface functions and one user-specified function. These are used to control the erasing, verifying, programming, and viewing of modules to be programmed. These generic functions are implemented specifically for the MC68HC908GP20 by the 908_GP20.08P programming algorithm and by the 908_GP20.08P programming algorithm for the MC68HC908GP32.

The SPGMR can be configured to program a processor resident in a target system. For setup instructions, see **2.5 Connecting to a Target System**.

The programming routines for a particular module are loaded into the CPU08 on-chip RAM for execution during erasure, programming, verification, and showing of the module. The routines and associated comments for a particular module are in the form of Motorola S records stored in a file with a .08P extension.

Any of the enabled features of the PROG08SW programmer can be selected using the mouse or the up and down arrow keys or by typing the selection letters to the left of the selection display. Pressing ENTER or double clicking the mouse will execute the highlighted entry if it is enabled. The user will be prompted for any additional information required to execute the selected function. Before a module can be programmed from an S-record file, select such a file; otherwise, the user will be asked to select one.

8.3 Startup and Parameters

The PROG08SW FLASH programmer may be started and parameters may be passed to it in either of two ways:

- From the WinIDE editor, as described in **4.12.5.4 Executable Tabs: EXE 1-3**
- From the command line, by using the Windows 95 (or later versions) **Program Item Property** dialog. Refer to the Windows documentation for information on this procedure.

The following parameters may be entered in any order. To specify multiple parameters, separate them with spaces.

`[com(n)]`	The optional parameter `com(n)`, where `(n)` is a value from 1 to 8, specifies which communications port to use.
`[v]`	If the optional parameter `v` is specified as either V or v, then the range of S records is not verified during the programming or verification process. This can help speed up these functions.

Examples:

`PROG08SW com2 v`	com2 port is selected and S-records range is not verified.
`PROG08SW com7`	com7 port is selected and S-records range is verified.

8.4 Programming Commands

Programming commands are executed by selecting them from the pick list. This is done by either using the up and down arrow keys or by typing the first letter(s) on the line to select a command. Pressing ENTER causes the selected command to execute. Commands can also be executed from the menus or from the button bar. Any additional information needed for the command will be prompted for in a window which opens for that purpose. Errors caused by a command and any responses are presented in the status window.

NOTE: *Although the following commands are shown in the pick list, they are inactive for the M68HC980GP20 and will not execute if selected:*

- *BR*
- *EB*
- *EW*
- *PW*

In addition, there is one function that is allowed to be unique to the module being programmed. The selection menu name and the length of up to one hexadecimal parameter may be specified in a supporting .08P file.

8.4.1 BM — Blank-Check Module

This command checks the entire module to see if it has been erased. If not, the address of the first non-blank location is given along with its contents.

8.4.2 CM — Choose Module .08P

The user is presented with a list of available .08P files. Each .08P file contains information on how to program a particular module. Usually, the name of the file indicates what kind of module it relates to. For example, the file 908_GP20.08P specifies how to program a 68HC908GP20 device. Setup information and further descriptions of the module are provided in ASCII text within the module file. The user can look at this information with any standard text editor. This information is also presented in the status window when a .08P file is selected. A particular .08P file is selected by using the arrow keys to highlight the filename and then pressing the ENTER key. The currently selected

.08P file is shown in the .08P file selected window. After a .08P file is selected, the user is prompted for the base address of the module. This address is used as the beginning address for the module during programming and verification.

8.4.3 EM — Erase Module

This command erases the entire module. If the entire module is not erased, an error message is given.

8.4.4 PB — Program Bytes

The user is prompted for a starting address, which must be in the module. The user is then shown an address and a byte. Pressing the ENTER key shows the next location. The user can also enter in hex a byte to be programmed into the current location. Failure to program a location, entering an invalid hex value, or exceeding the address range of the module will exit the program bytes window. If a location fails to program, an error message is given. The symbols +, –, and = may be appended to the value being written. Respectively, they increase the address (default) (+), decrease the address (–), and hold the address constant (=).

8.4.5 PM — Program Module

For this command to work, the user must have selected an S-record file previously. The S records are then checked to see if they all reside in the module to be programmed. If not, the user is asked for permission to continue. If the answer is yes, only those S-record addresses that lie in the module are programmed. If a location could not be programmed, an error message is given.

8.4.6 SM — Show Module

The user is prompted for a starting address. If this address is not in the module, an error is given. A window is opened that shows the contents of memory as hex bytes and ASCII characters if printable. Non-printing characters are shown as periods (.). This window stays on the screen until the user presses the ESCAPE key.

8.4.7 SS — Specify S Record

This command asks the user for the name (and/or path) to a file of Motorola S records to be used in programming or verifying a module. If the file is not found, an error message is given. The currently selected file is shown in the S19 file selected window. The programmer accepts S1, S2, and S3 records. All other file records are treated as comments. If the user does not specify a filename extension, a default of .S19 is used.

8.4.8 UM — Upload Module

The user is then asked for a filename in which to upload S records. The default filename extension is set to .S19 if none is specified by the user. Motorola S records for the entire module are then written to the specified file.

8.4.9 UR — Upload Range

The user is prompted for a starting address, which must be in the module. Next, the user is asked for an ending address, which must also be in the module. The user is then asked for a filename in which to upload S records. The default filename extension is set to .S19 if none is specified by the user. Motorola S records are then written to the specified file.

8.4.10 VM — Verify Module

For this command to work, the user must have previously selected an S-record file. The S records are then checked to see if they all reside in the module to be programmed. If not, the user is asked for permission to continue. If the answer is yes, only those S-record addresses that lie in the module are verified. If a location could not be verified, an error message is given, indicating the address, the contents of that address, and the contents specified in the S-record file.

8.4.11 VR — Verify Range

For this command to work, the user must have selected an S-record file previously. The user is prompted for a starting address, which must be in the module. Next, the user is asked for an ending address, which must also be in the module. S-record addresses that lie in the module are verified. If a location could not be verified, an error message is given, indicating the address, the contents of that address, and the contents specified in the S-record file.

8.4.12 QU — QUit

This command terminates the programmer and returns to Windows.

8.4.13 RE — REset Chip

This causes a hardware reset to the CPU08 chip. This command can be used to recover from errors that cause the programmer not to be able to communicate with the processor through the MON08 monitor interface.

8.4.14 HE — HElp

This command opens a window of help topics on the screen. The user can then select a particular topic and page through its text description.

8.5 Programming Example

These programming steps illustrate a typical sequence for using the PROG08SW commands:

1. Start the PROG08SW software, as described in **8.3 Startup and Parameters**.

   When PROG08SW starts, it performs an automatic REset (see **8.4.13 RE — REset Chip**) and brings up the Choose Module selection window.
2. Select the 908_GP20.08P file from the Choose Module window.
3. Execute the Blank-Check Module command (**8.4.1 BM — Blank-Check Module**).
4. If the results from the BM command indicate that the module is not blank, execute the Erase Module command (see **8.4.3 EM — Erase Module**). Then repeat the BM command, which should now indicate that the module is blank.

5. With the Specify S-record command (see **8.4.7 SS — Specify S Record**), select the S-record file to load into the module.
6. Execute the Program Module command (see **8.4.5 PM — Program Module**).
7. To verify that the S-record file has been loaded correctly, execute the Verify Module command (see **8.4.10 VM — Verify Module**), which compares the S-record file with the contents of the module.

Section 9. ICD08SW In-Circuit Debugger

9.1 Contents

9.2 Introduction 172

9.3 MON08 Debugging Limitations and Tips 172

9.4 Startup and Parameters 175

9.5 User Interface 176
9.5.1 Status Window 176
9.5.1.1 Pop-Up Menu 176
9.5.1.2 Keystrokes 177
9.5.2 Code Window 177
9.5.2.1 Pop-Up Menu 178
9.5.2.2 Keystrokes 179
9.5.3 Variables Window 180
9.5.3.1 Pop-Up Menu 180
9.5.3.2 Keystrokes 181
9.5.4 Memory Window 181
9.5.4.1 Pop-Up Menu 182
9.5.4.2 Keystrokes 182
9.5.5 Colors Window 183
9.5.6 CPU08 Window 184
9.5.6.1 Pop-Up Menu 184
9.5.6.2 Keystrokes 185

9.6 Debugging Commands 185
9.6.1 Syntax and Nomenclature 185
9.6.2 Command Recall 186
9.6.3 Command Set Summary 187

9.2 Introduction

This chapter describes the use of the ICD08SW in-circuit debugger (ICD) for the M68HC08 serial programmer (SPGMR). The debugger employs a command set that allows real-time debugging within the limitations of the M68HC08's MON08 on-chip debugging monitor.

9.3 MON08 Debugging Limitations and Tips

These limitations are inherent in MON08 debugging and should be observed carefully:

1. Do not change the data direction or data value of PORTA bit 0. When written, these bits should be set to 0.
2. Do not enable keyboard interrupts for PORTA bit 0.
3. Do not step an instruction that branches to itself.
4. Do not step a software interrupt instruction (SWI).
5. The hardware breakpoint registers are reserved for use by the ICD08SW debugger. Attempting to use these registers for other purposes may not work.
6. Be careful about showing peripheral status and data registers in the memory or variables window. A refresh of these windows will read these registers and may cause the clearing of flags.
7. The debug monitor built into the HC08 processor uses up to 13 bytes of the stack. Do not write to these addresses from (SP–13) to SP. To load a program into RAM, the stack should be moved to the end of RAM; otherwise, the processor will use 13 bytes in the middle of the RAM block.

   Use the command SP 23F to move the stack. If the command is put into the STARTUP.ICD file, it will execute every time the debugger is entered.
8. If interrupts are turned on during stepping, the ICD08SW debugger will not step into the interrupt. Instead, it will execute the whole interrupt and stop on the instruction returned to after the interrupt.

The following tips provide useful insight:

1. Single stepping is allowed in both RAM and ROM.
2. The first breakpoint set is always a hardware breakpoint, and any additional breakpoints set are software breakpoints. To make sure that a hardware breakpoint is being set, use the NOBR command before setting it.

   Hardware breakpoints will stop execution in ROM and RAM. Software breakpoints will stop execution only in RAM.
3. Experiment with the register interpreter. Use the R command for this.
4. Code may be loaded only into RAM. When doing this, observe limitation number 7. To load code into FLASH memory, use the PROG08SW programmer included in this kit.
5. Executing an SWI instruction while running is functionally equivalent to hitting a breakpoint, except that execution stops at the instruction following the SWI.
6. A hardware breakpoint may be used to trap a data read/write to anywhere in the memory map. The ICD08SW debugger stops at the instruction after the one that accesses the data location.

   To trap a read/write to address 22, for example, first use the NOBR command to make sure that no breakpoints are set. Then, set the hardware breakpoint by using the BR 22 command. This is the same way that a hardware instruction breakpoint would be set. Clear the hardware breakpoint by using the NOBR command.
7. The LOADALL command in ICD08SW is the same as LOAD in the ICS08GPW simulator. The LOAD command loads only the object information, not the debug information.
8. To debug from ROM and see source code while stepping, use the LOADMAP command. This loads source-level information about the source file without loading the object file, which should have been programmed into FLASH with PROG08SW. Files with the extension .MAP are debug-format map files.
9. To write a byte to memory, use the MM <address> <n> value command. For example, to write $00 to address $4, enter MM 4 0.

10. The default base of the debugger is hexadecimal. See HELP for prefixes and suffixes to override the default.

11. To create a variable, use the VAR command. To clear all variables, use CLEARVAR.

12. CPU register values can be changed by entering a register name followed by a value. For example, to set the accumulator to $44, type either `A 44` <ENTER> or `ACCA 44` <ENTER>.

13. When the ICS08GP20 board is reset by the debugger, power to the microcontroller is turned off for a short duration. Although much of RAM may look the same, some values may have changed. To verify code that was loaded prior to the reset, use the VERIFY command.

14. All windows have right-button mouse menus, which allows access to much of the debugger's functionality. To open, place the mouse over the window and click the right mouse button.

15. If a GO command is entered without setting a breakpoint, the only way to regain control of the processor is to reset it.

16. The watchdog is not active while running ICD08SW. When a device is programmed and powered without the debugger – for example, on a target board – the watchdog is active by default.

17. If the security bytes (see **1.9 MC68HC908GP Security Feature**) are programmed with the PROG08SW programmer, the ICD08SW debugger automatically knows the security bytes. In this way, ICD08SW is able to reset the processor even if it has been programmed. All security information is stored in the file SECURITY.INI.

18. To save the ICD08SW desktop settings, use the SAVEDESK command. Retrieve them by using the LOADDESK command. The desktop settings are retrieved automatically upon startup.

9.4 Startup and Parameters

The ICD08SW debugger may be started, and parameters may be passed to it, in either of two ways:

- From the WinIDE editor, as described in **4.12.5.4 Executable Tabs: EXE 1-3**
- From the command line, by using the Windows 95 (or later version) **Program Item Property** dialog. Refer to the Windows documentation for information on this procedure.

The following optional parameters may be entered in any order. To specify multiple parameters, separate them with spaces.

`com <n>`	Chooses the serial port, where `<n>` is a value between 1 and 8
`running`	Starts the ICD with the CPU running. Sometimes it is desirable to leave the CPU running and exit the ICD debug software. To do this, use the GOEXIT command. To re-enter the ICD debug software, use the option `RUNNING` as a parameter on the startup command line. This option causes the debugger NOT to do a RESET at startup and to ignore any STARTUP.ICD macro file. To use this option, the CPU must have been previously left executing by the debugger.
`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.
`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 (the path part is also saved)

Example:

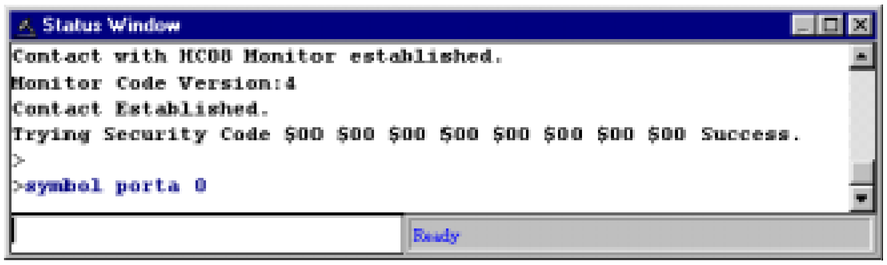
`ICD08SW com2 running` — Chooses com2 and starts the ICD with the CPU running

9.5 User Interface

The following sections describe the Windows user interface for the ICD08SW in-circuit debugger.

9.5.1 Status Window

The Status window, shown in **Figure 9-1**, 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.



```
Contact with HC08 Monitor established.
Monitor Code Version:4
Contact Established.
Trying Security Code $00 $00 $00 $00 $00 $00 $00 $00 Success.
>
>symbol porta 0
```

Figure 9-1. ICD08SW Status Window

Individual commands or a series of commands can be typed into the window or 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 that 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.

9.5.1.1 Pop-Up Menu

By pressing the right mouse button while the cursor is over the status window, the user is given a pop-up menu, which has the option:

Help — Displays this help topic

9.5.1.2 Keystrokes

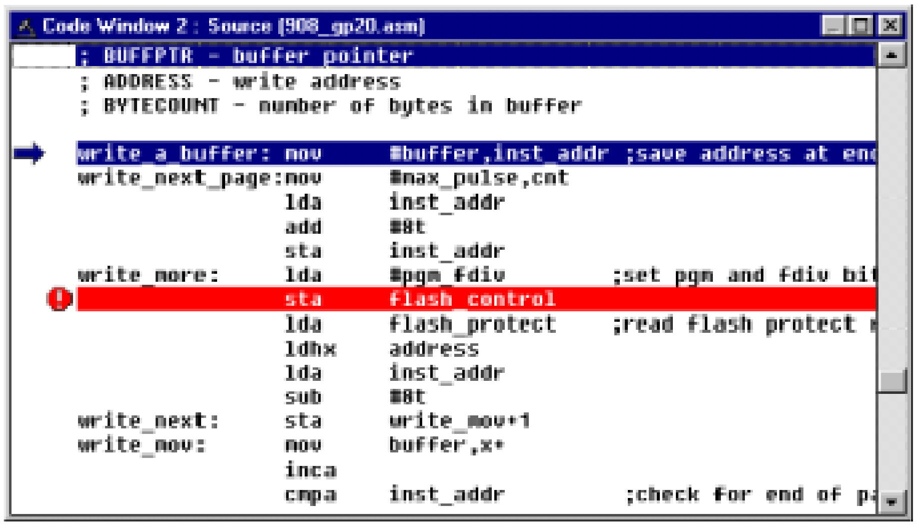
These keystrokes are valid while the status window is the active window:

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

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

9.5.2 Code Window

The Code window, shown if **Figure 9-2**, displays either disassembled machine code or the user's source code if it is available. The disassembly mode will always show disassembled code regardless if 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; otherwise, disassembly is shown. To show both modes at once, the user should have two code windows open, one set one to disassembly and the other to source/disassembly.



```
; BUFFPTR - buffer pointer
; ADDRESS - write address
; BYTECOUNT - number of bytes in buffer

write_a_buffer: mov      #buffer,inst_addr ;save address at en
write_next_page:mov      #max_pulse,cnt
                lda      inst_addr
                add      #8t
                sta      inst_addr
write_more:     lda      #pgm_fdiv         ;set pgm and fdiv bi
                sta      flash_control
                lda      flash_protect     ;read flash protect
                ldhx     address
                lda      inst_addr
                sub      #8t
write_next:     sta      write_mov+1
write_mov:      mov      buffer,x+
                inca
                cmpa     inst_addr          ;check for end of p
```

Figure 9-2. ICD08SW Code Window 2

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

9.5.2.1 Pop-Up Menu

By pressing the right mouse button while the cursor is over the code window, the user is given a pop-up menu with these options:

- **Toggle Breakpoint at Cursor** — This option is enabled if the user has already selected a line in the code window by clicking on it with the left mouse button. Choosing this option will set a breakpoint at the selected location or, if there is already a breakpoint at the selected location, will remove it.
- **Set PC at Cursor —** This option is enabled if the user has already selected a line in the code window by clicking on it with the left mouse button. Choosing this option will set the program counter (PC) to the selected location.
- **Gotil Address at Cursor** — This option is enabled if the user has already selected a line in the code window by clicking on it with the left mouse button. Choosing this option will set a temporary breakpoint at the selected line and start 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 pops 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 always shows disassembled code, regardless of whether a source file is loaded. The show source/disassembly mode shows source-code if source code is loaded and the current PC points to a valid line within the source code. Otherwise, disassembly is shown.
- **Help** — Displays help for this topic

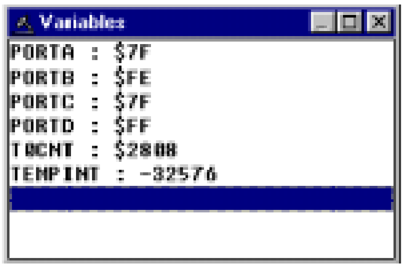
9.5.2.2 Keystrokes

These keystrokes are valid while the Code window is the active window:

`UP ARROW`	Scrolls window up one line
`DOWN ARROW`	Scrolls window down one line
`HOME`	Scrolls window to the Code window's base address.
`END`	Scrolls window to last address the window will show.
`PAGE UP`	Scrolls window up one page
`PAGE DOWN`	Scrolls window down one page
`F1`	Displays help for this topic
`ESC`	Makes the Status window the active window

9.5.3 Variables Window

The Variables window, shown in **Figure 9-3**, is used to view variables while the part is not running. The user may add or remove variables through the INSERT or DELETE keys, the pop-up menu, or the VAR command. Variables can be viewed as bytes (8 bits), words (16 bits), longs (32 bits), or strings (ASCII).



```
PORTA : $7F
PORTB : $FE
PORTC : $7F
PORTD : $FF
TACNT : $2808
TEMPINT : -32576
```

Figure 9-3. ICD08SW Variables Window

9.5.3.1 Pop-Up Menu

By pressing the right mouse button while the cursor is over the variables window, the user is given a pop-up menu which has these options:

- **Add Variable** — Adds a variable to the Variables window at the currently selected line. A pop-up window allows the user to specify the variable's address, type, and base.
- **Delete Variable** — Removes the selected variable from the Variables window. A variable is selected by placing the mouse cursor over the variable name and clicking the left mouse button.
- **Clear All** — Removes all variables from the Variables window
- **Help** — Displays help for this topic

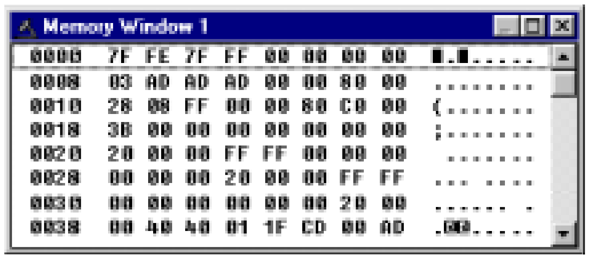
9.5.3.2 Keystrokes

These keystrokes are valid while the Variables window is the active window:

`INSERT`	Adds a variable
`DELETE`	Deletes a variable
`UP ARROW`	Scrolls window up one variable
`DOWN ARROW`	Scrolls window down one variable
`HOME`	Scrolls window to the first variable
`END`	Scrolls window to the last variable
`PAGE UP`	Scrolls window up one page
`PAGE DOWN`	Scrolls window down one page
`F1`	Displays help for this topic
`ESC`	Makes the Status window the active window

9.5.4 Memory Window

The Memory window, shown in **Figure 9-4**, 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. To modify a particular set of bytes, double click on them. Double clicking on bits brings up a byte modification window.



```
0000  7F FE 7F FF 00 00 00 00  █.█.....
0008  03 AD AD AD 00 00 80 00  ........
0010  28 00 FF 00 00 80 C0 00  (.......
0018  3B 00 00 00 00 00 00 00  ;.......
0020  20 00 00 FF FF 00 00 00   .......
0028  00 00 00 20 00 00 FF FF  ... ....
0030  00 00 00 00 00 00 20 00  ...... .
0038  00 40 40 01 1F CD 00 AD  .@@.....
```

Figure 9-4. ICD08SW Memory Window

9.5.4.1 Pop-Up Menu

By pressing the right mouse button while the cursor is over the Memory window, the user is given a pop-up menu which has these 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 (MD) Memory Display 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** — Displays help for this topic

9.5.4.2 Keystrokes

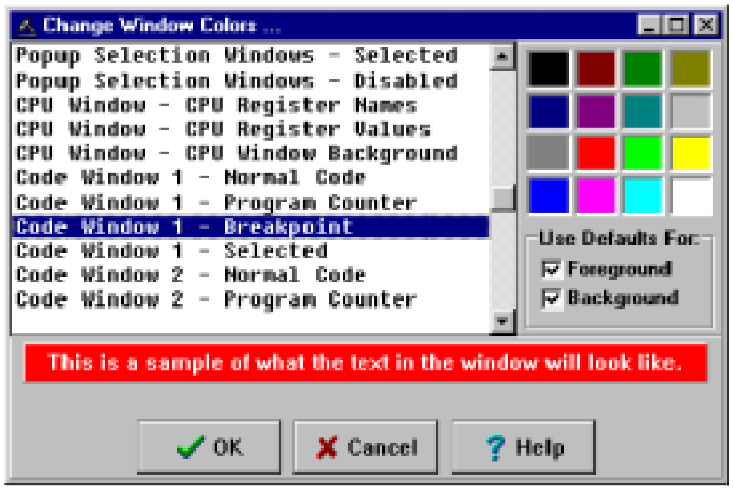
These keystrokes are valid while the Memory window is the active window:

`UP ARROW`	Scrolls window up one line
`DOWN ARROW`	Scrolls window down one line
`HOME`	Scrolls window to address $0000
`END`	Scrolls window to last address in the memory map.
`PAGE UP`	Scrolls window up one page
`PAGE DOWN`	Scrolls window down one page
`F1`	Displays this help topic
`ESC`	Makes the Status window the active window

9.5.5 Colors Window

The Colors window, shown in **Figure 9-5**, shows the colors that are set for all of the debugger windows. 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; 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 allow only the foreground or background to be changed. Press the OK button to accept the color changes. Press the CANCEL button to decline all changes.



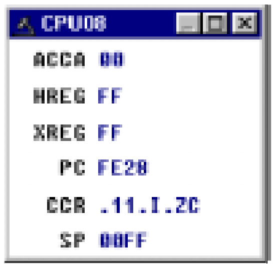
Popup Selection Windows - Selected
Popup Selection Windows - Disabled
CPU Window - CPU Register Names
CPU Window - CPU Register Values
CPU Window - CPU Window Background
Code Window 1 - Normal Code
Code Window 1 - Program Counter
Code Window 1 - Breakpoint
Code Window 1 - Selected
Code Window 2 - Normal Code
Code Window 2 - Program Counter

This is a sample of what the text in the window will look like.

Figure 9-5. ICD08SW Change Window Colors Window

9.5.6 CPU08 Window

The CPU08 window, shown in **Figure 9-6**, displays the current state of the 68HC08 CPU registers. The pop-up window allows modification of these values.



```
ACCA 00
HREG FF
XREG FF
  PC FE28
 CCR .11.I.ZC
  SP 00FF
```

Figure 9-6. ICD08SW CPU08 Window

9.5.6.1 Pop-Up Menu

By pressing the right mouse button while the cursor is over the CPU window, a pop-up menu appears with 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 HREG Index Register** — Sets the H index register to a user-defined value. Upon selecting this option, the user is prompted for a value.
- **Set XREG Index Register** — Sets the X index register to a user-defined value. Upon selecting this option, the user is prompted for a value.
- **Set Stack Pointer** — 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, as shown in **Figure 9-7**.

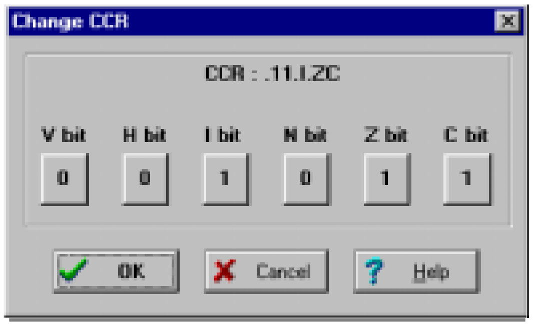


Figure 9-7. ICD08SW Set CCR Window

9.5.6.2 Keystrokes

These keystrokes are valid while the CPU window is the active window:

`F1`	Displays help for this topic
`ESC`	Make the Status window the active window

9.6 Debugging Commands

Debugging commands for use with the ICD08SW in-circuit debugger are described in the following sections. The individual commands are defined in detail in **Section 10. Debugging Command Set**.

9.6.1 Syntax and Nomenclature

A command is a line of ASCII text that was entered from the computer keyboard. For ICD08SW debugging commands, enter the command and its arguments in the Status window command line. Press ENTER to terminate each line and activate the command. The typical command syntax is:

```
command [<argument>]...
```

Where:

`command`	A command name, in upper- or lower-case letters
`<argument>`	An argument indicator; when arguments are italicized, they represent a placeholder for the actual value entered; when not italicized, they indicate the actual value to enter. **Table 7-1. Argument Types** explains the possible argument values.

The following nomenclature conventions apply to the ICD08SW in-circuit debugging commands:

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 or 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 semi-colon character is considered a comment. This helps in documenting macro (script) files.

9.6.2 Command Recall

The PGUP and PGDN keys can be used 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. The ESC key can delete a currently entered line, including one selected by scrolling through old commands.

NOTE: *Only the 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 for data to be written in memory. These user responses are not saved for scrolling; however, the original memory modify command is saved.*

9.6.3 Command Set Summary

Table 9-1 summarizes the debugging commands that may be used with ICD08SW. For detailed descriptions of each command, refer to **Section 10. Debugging Command Set**.

Table 9-1. ICD08SW Command Overview (Sheet 1 of 5)

Command	Description
A	Set the accumulator to specified value and display new value in CPU window (identical to the ACC command).
ACC	Set the accumulator to specified value and display new value in CPU window (identical to the A command).
ASCIIF3 ASCIIF6	Toggle memory windows between displaying data only and data with ASCII characters
ASM	Assemble M68HC08 instruction mnemonics and place resulting machine code in memory at the specified address.
BELL	Sound PC bell the specified number of times.
BF FILL	Fill a block of memory with a specified byte, word, or long value.
BR	Display or set instruction breakpoint to specified values or at cursor location.
C	Set or clear the C bit of the CCR.
CCR	Set the CCR in the CPU to the specified hexadecimal value.
CLEARMAP	Remove the current MAP file from memory.
CLEARSYMBOL	Remove all user-defined symbols from memory.
CODE	Show disassembled code in the Code window starting at address add. Specifying an address in the middle of an intended instruction may cause improper results.
COLORS	Set simulator colors.
DASM	Disassemble machine instructions, display addresses and contents as disassembled instructions in the Code window.
DUMP	Send contents of a block of memory to the Status window in bytes, words, or longs.
EVAL	Evaluate a numerical term or expression and give the result in hexadecimal, decimal, octal, and binary format.

Table 9-1. ICD08SW Command Overview (Sheet 2 of 5)

Command	Description
EXIT	Terminate the software and close all windows (identical to QUIT command).
G GO	Start execution of code at the current PC address or at an optional specified address. The G, GO, and RUN commands are identical.
GOEXIT	Similar to GO command except that the target is left running without any breakpoints, and the debugger software is terminated.
GONEXT	Execute from the current PC address until the next instruction is reached. Used to execute past a subroutine call or past intervening interrupts.
GOTIL	Execute code beginning at the PC address and continue until the PC contains the specified ending address or until a keypress, Stop Macro command (from the toolbar), breakpoint, or error occurs.
H HREG	Set or clear the H bit in the CCR.
HELP	Open the ICD08SW Help File.
HX	Set both bytes of the concatenated index register H:X to the specified value.
I	Set or clear the I bit of the CCR.
INFO	Display information about the line highlighted in the source window.
INT IRQ	Display the value of the IRQ pin.
LF LOGFILE	Open a new or specified external file to receive log entries of commands and responses in the Status window (identical to the LOGFILE command).
LOAD	Load S19 object file and associated MAP file into the ICD08SW.
LOADALL	Execute both the LOAD and LOADMAP commands.
LOADDESK	Load the desktop settings for window positions, size, and visibility.
LOADMAP	Load a MAP file containing source level debug information into the ICD08SW.
LOADV	Execute the LOAD command, then automatically execute the VERIFY command.

Table 9-1. ICD08SW Command Overview (Sheet 3 of 5)

Command	Description
LOAD_BIN	Loads a binary file of bytes starting at a specified address.
LOADV_BIN	Execute the LOAD_BIN command, then automatically execute the VERIFY command.
MACRO	Execute a macro file containing debug command sequences (identical to the SCRIPT command).
MACROEND	Close the macro file in which the debug command sequences are being saved.
MACROSTART	Open a macro file and save all subsequent debug commands to this file until closed by the MACROEND command during an active ICS08GPW session.
MACS	Bring up a window with a list of macros.
MAP	View current MAP file. The SHOWMAP command is identical.
MD MD1	Display the contents of memory locations in Memory window 1, beginning at the specified address.
MD2	Display the contents of memory locations in Memory window 2, beginning at the specified address.
MM MEM	Modify contents of memory beginning at the specified address, and/or select bytes, words, longs.
N	Set or clear the N bit of the CCR.
NOBR	Remove one or all of active breakpoints.
PC	Assign the specified value to the MCU program counter.
QUIET	Toggles refresh of memory-based windows.
QUIT	Terminate the ICD08SW application and close all windows (identical to the EXIT command).
R	Open window for Register files and start interactive setup of system registers such as I/O, timer, COP.
REG	Display contents of CPU registers in the Status window (identical to the STATUS command).
REM	Enter comments in a macro file.
RESET	Simulate a reset of the MCU and set the PC to the contents of the reset vector. Does not start execution of user code.
RUN	Start execution of code at the current PC current or specified address. The G, GO, and RUN commands are identical.

Table 9-1. ICD08SW Command Overview (Sheet 4 of 5)

Command	Description
SAVEDESK	Save the desktop settings for the ICS08GPW program when it is first opened or for use with the LOADDESK command.
SHOWCODE	Display code in the Code windows beginning at the specified address, but without changing the value of the PC.
SHOWMAP	View current MAP file. The MAP command is identical.
SHOWPC	Display code in the Code window, starting from the address in the PC.
SNAPSHOT	Save window data to the open log file.
SOURCEPATH	Determine the path for source code that is not in the current working directory.
SP	Assign specified value to the stack pointer used by the CPU and display in the CPU window.
SS	Step through a specified number of source code instructions, starting at the current PC address value, then halt.
ST	Step through a specified number of assembly instructions, starting at the current PC address, then halt (identical to the STEP and T commands).
STATUS	Display the contents of the CPU registers in the Status window (identical to the REG command).
STEP	Step through a specified number of assembly instructions, starting at the current program counter address, then halt (identical to the ST and T commands).
STEPFOR	Execute instructions continuously, one at a time, starting at the current PC address and continuing until reaching an error condition, breakpoint, or keypress.
STEPTIL	Step through instructions starting at current PC address and continue until PC value reaches the specified address, or until keypress, breakpoint, or error occurs.
SYMBOL	View current symbols or create new symbols.
T	Step through a specified number of assembly instructions, starting at the current PC address, then halt (identical to the ST and STEP commands).
UPLOAD_SREC	Upload the content of the specified memory block (range) in S19 file format, display the contents in the Status window, and enter information into the current logfile.
V	Set or clear the V bit in the condition code register (CCR).

Table 9-1. ICD08SW Command Overview (Sheet 5 of 5)

Command	Description
VAR	Display specified address and contents in the Variables window for viewing during code execution.
VERIFY	Compare the contents of program memory with an S-record file.
VER VERSION	Display program version and date.
WHEREIS	Display value of the specified symbol.
X XREG	Set the X register to the specified value and display in the CPU window.
Z	Toggle the Z bit in the CCR.

Section 10. Debugging Command Set

10.1 Contents

A or ACC	Set Accumulator Value	197
ADCLR	Clear A/D Input Buffer	198
ADDI	Input Data to A/D Converter	199
ASCIIF3 and ASCIIF6	Toggle ASCII Display	200
ASM	Assemble Instructions	201
BELL	Sound PC Bell	202
BF or FILL	Block Fill Memory	203
BR	Set Instruction Breakpoint	204
BREAKA	Set Accumulator Breakpoint	206
BREAKHX	Set HX Register Breakpoint	208
BREAKSP	Set Stack Pointer Breakpoint	210
C	Set/Clear Carry Bit	212
CAPTURE	Capture Changed Data	213
CAPTUREFILE or CF	Open/Close Capture File	214
CCR	Set Condition Code Register	215
CGMXCLK	Set CGMXCLK to Bus Clock Relationship	216
CHIPMODE	Choose Device for Simulation	218
CLEARMAP	Clear .MAP File	219
CLEARSYMBOL	Clear User Symbols	220
CODE	Show Disassembled Code	221
COLORS	Set Simulator Colors	222
CYCLES or CY	Set Cycles Counter	223
DASM	Disassemble Memory	224
DDRA	Set Port A Direction Register	225
DDRB	Set Port B Direction Register	226
DDRC	Set Port C Direction Register	227
DDRD	Set Port D Direction Register	228
DDRE	Set Port E Direction Register	229
DUMP	Dump Memory to Screen	230
EVAL	Evaluate Expression	231

EXIT or QUIT	Exit/Quit Application	232
G, GO, or RUN	Begin Program Execution	233
GOEXIT	Execute Without Breakpoints/Debugger	234
GONEXT	Execute Past Subroutine/Interrupt	235
GOMACRO	Execute Macro after Break	236
GOTIL	Execute Until Address	237
GOTOCYCLE	Execute to Cycle Counter Value	238
H or HREG	Set/Clear Half-Carry Bit	239
HELP	Open Help	240
HX	Set H:X Index Register Pair	241
I	Set/Clear Interrupt Mask	242
INFO	Display Line Information	243
INPUTA	Set Port A Inputs	244
INPUTB	Set Port B Inputs	245
INPUTC	Set Port C Inputs	246
INPUTD	Set Port D Inputs	247
INPUTE	Set Port E Inputs	248
INPUTS	Show Port Inputs	249
INT or IRQ	Set IRQ Pin State	250
LF OR LOGFILE	Open/Close Logfile	251
LISTOFF	Turn Off Step Listing	252
LISTON	Turn On Step Listing	253
LOAD	Load S Records	254
LOADALL	Load S Records and Map File	255
LOADDESK	Load Desktop Settings	256
LOADMAP	Load Map File	257
LOADV	Load and Verify	258
LOAD_BIN	Load Binary File	259
LOADV_BIN	Load and Verify Binary File	260
MACRO	Execute Batch File	261
MACROEND	Stop Saving Commands to Batch File	263
MACROSTART	Save Debug Commands to Batch File	264
MACS	List Macros	265
MAP	Show Information in Map File	266
MD or MD1	Display Memory at Address	267
MD2	Display Memory (Window 2) at Address	268
MM or MEM	Modify Memory	269
N	Set/Clear Negative Bit	271

NOBR	Remove Breakpoints 272
NOMAP	Clear .MAP File. 273
NOSYMBOL	Clear User Symbols 274
PC	Set Program Counter 275
POD	Change Serial Port 276
PORTA or PRTA	Set Port A Output Latches 277
PORTB or PRTB	Set Port B Output Latches 278
PORTC or PRTC	Set Port C Output Latches 279
PORTD or PRTD	Set Port D Output Latches 280
PORTE or PRTE	Set Port E Output Latches 281
QUIET	Toggle Window Refresh 282
R	Use Register Files 283
REG	Show Registers 284
REM	Place Comment in Batch/Macro File. 285
RESET	Simulate Processor Reset. 286
RESETGO	Reset and Restart MCU 287
SCCLR	Clear SCI I/O Buffers 288
SCDI	Input SCI Data 289
SCDO	View SCI Outputs 290
SAVEDESK	Save Desktop Settings 291
SHOWBREAKS	Display Breakpoint Window 292
SHOWCODE	Display Code at Address 293
SHOWMAP	Show Information in Map File 294
SHOWPC	Display Code at PC Address 295
SHOWTRACE	Display Trace Window 296
SIM08	Switch Simulation Mode 297
SNAPSHOT	Save Window Data to Logfile 298
SOURCEPATH	Set Path for Source Code. 299
SP	Set Stack Pointer 300
SPCLR	Clear SPI I/O Buffers. 301
SPDI	Enter SPI Inputs. 302
SPDO	View SPI Outputs 303
SPFREQ	Set Input Clock for SPI Slave 304
SS	Execute Source Step(s) 305
ST or STEP or T	Execute Single Step 306
STACL	Show Stack Window 307
STATUS	Show Registers 308
STEPFOR	Step Forever. 309

STEPTIL	Step Until Address	310
SYMBOL	Add Symbol	311
TRACE	Enable/Disable Tracing	312
UPLOAD_SREC	Upload S Record to Screen	313
V	Set or Clear V Bit (CCR)	314
VAR	Display Variable	315
VERIFY	Verify S-Record File	316
VERSION or VER	Display Software Version	317
WAIT	Wait for n Cycles	318
WHEREIS	Display Symbol Value	319
X or XREG	Set X Register Value	320
Z	Set/Clear Zero Bit	321

10.2 Command Descriptions

The debugging command set is arranged alphabetically by command name in this section. The entries describe the commands in detail.

Set Accumulator Value — A or ACC

The ACC command sets the accumulator to a specified value. The value entered with the command is shown in the CPU window. The ACC and A commands are identical.

Syntax: `ACC <n>`

Where:

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

Use with: ICS08GPW and ICD08SW

Example:	`A 10`	Set the accumulator to $10.

ADCLR — Clear A/D Input Buffer

The ADCLR command can be used to flush the input buffer for A/D simulation. This will reset the circular buffer and clear all values. Notice that if the A/D is using a value currently, this command will not prevent the A/D from using it. See ADDI for accessing the input buffer of the A/D interface.

If the ICS08 circuit board is connected, A/D inputs are received from the board, so this command has no effect.

Syntax: `ADCLR <n>`

Where:

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

Use with: ICS08GPW only

Example:	`ADCLR`	Clear the input buffer for A/D simulation.

Input Data to A/D Converter ADDI

The ADDI command allows the user to input data into the A/D converter. If a data parameter is given, the value is placed into the next slot in the circular input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the A/D. The maximum number of input values is 256 bytes.

If the ICS08 circuit board is connected, A/D inputs are received from the board, so this command has no effect.

Syntax: `ADDI <n>`

Where:

`<n>`	The value to be entered into the next location in the input buffer

Use with: ICS08GPW only

Example:

`ADDI $55`	Set the next input value to the ADDI to $55.
`ADDI`	Pull up the data window with all the input values.

ASCIIF3 and ASCIIF6 — Toggle ASCII Display

The ASCIIF3 and ASCIIF6 commands toggle the memory windows between displaying data only and data and ASCII characters.

ASCIIF3 toggles memory window 1. ASCIIF6 toggles memory window 2.

Syntax: `ASCIIF3`

Use with: ICD08SW only

Example: `ASCIIF3` Toggles memory window 1 between displaying data only and data with ASCII characters

Assemble Instructions ASM

The ASM command assembles M68HC08 Family 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. Enter the new instruction in the **New Instruction** text box. Press the ENTER key to assemble the new instruction, store and display the resulting machine code, then move to the next memory location, where the user will be prompted for another instruction.

If there is an error in the instruction format, the address stays at the current address and an assembly error flag appears. To exit assembly, press the EXIT button.

Syntax: `ASM [<address>]`

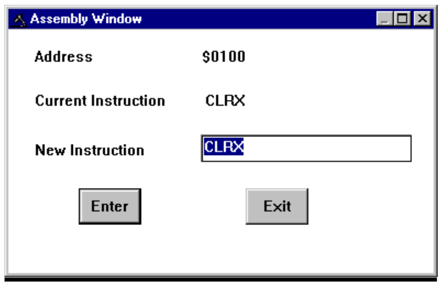
Where:

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

Use with: ICS08GPW and ICD08SW

Example: With an address argument: `ASM 100`

The Assembly window with the ASM command and no argument is shown in **Figure 10-1**.



Address $0100
Current Instruction CLRX
New Instruction CLRX

Figure 10-1. Assembly Window – ASM Command

BELL — Sound PC Bell

The BELL command sounds the PC bell the specified number of times. With no argument, the bell sounds once. 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

Use with: ICS08GPW and ICD08SW

Example:

`BELL 3`	Ring PC bell three times.

Block Fill Memory — BF or FILL

The BF or FILL command fills a block of memory with a specified byte, word, or long value. The optional argument specifies whether to fill the block in bytes (.B, the default, 8 bits) or in words (.W, 16 bits).

The ICD08SW debugger can also supply the argument as type long (.L, 32 bits).

Syntax: `BF [.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, word, or long value to be stored in the specified block • If the byte variant (.B) is used , then `<n>` must be an 8-bit value. • If the word variant (.W) is used , then `<n>` must be a 16-bit value. • If the long variant (.L) is used , then `<n>` must be a 32-bit value (ICD08SW only)

Use BF with: ICS08GPW and ICD08SW

Use FILL with: ICD08SW only

Examples:

`BF C0 CF FF`	Store FF in bytes at addresses C0–CF.
`BF.W 300 31F 4143`	Store word value 4143 at addresses 300–31F.

BR Set Instruction Breakpoint

The BR command displays or sets instruction breakpoints, according to its parameter values:

- If no parameter is entered, the BR command displays a list of all current breakpoints in the status window.
- If an `<address>` value is entered, the BR command sets a breakpoint at the specified address.

An optional value `<n>` may be entered with the address to specify a break count. The BR command sets a breakpoint at the specified address, but code execution does not break until the nth time it arrives at the breakpoint.

NOTE: *The maximum number of breakpoint addresses is 64. Each BR, BREAKA, BREAKSP, or BREAKHX command that includes an address value uses an additional breakpoint address, unless the address is a duplicate. For example, if 64 BR commands already have taken up 64 addresses, the only way to include an address value in a BREAKA, BREAKSP, or BREAKHX command is to duplicate one of those 64 addresses.*

If source code is displayed in either code window, mouse or keyboard commands can be used to set, remove, or clear all breakpoints:

1. Position the cursor on the line of code where the breakpoint will be set.
2. Click the left mouse button to select the line.
3. Press the right mouse button once to open the **Code Window Shortcut** menu.
4. Select **Toggle Breakpoint at Cursor** option. If there is no current breakpoint set at this line of code, a breakpoint will be set. If there is a current breakpoint set at this line of code, the breakpoint will be removed.

To remove all breakpoints, enter the NOBR command in the Status window command line.

Set Instruction Breakpoint (continued) BR

Syntax:	`BR [<address> [<n>]]`	;set a breakpoint
	`BR`	;list current breakpoints

Where:

`<address>`	The address for a breakpoint.
`<n>`	Break after value: code execution passes through the breakpoint n-1 times, then breaks the nth time it arrives at the breakpoint.

Use with: ICS08GPW and ICD08SW

Examples:	`BR 300`	Set a breakpoint at address 300.
	`BR 330 8`	Set a breakpoint at address 330, break on eighth arrival at 330.

BREAKA Set Accumulator Breakpoint

The BREAKA command sets an accumulator breakpoint to halt code execution when the value of the accumulator equals the specified `n` value.

- With an `n` value, the command forces a break in execution as soon as the accumulator value equals `n`.
- With `n` and `address` values, the command forces a break in execution when the accumulator value equals `n` and execution arrives at the specified address. If the accumulator value changes from `n` by the time execution arrives at the address, no break occurs.

NOTE: *The maximum number of breakpoint addresses is 64. Each BR, BREAKA, BREAKSP, or BREAKHX command that includes an address value uses an additional breakpoint address, unless the address is a duplicate. For example, if 64 BR commands already have taken up 64 addresses, the only way to include an address value in a BREAKA, BREAKSP, or BREAKHX command is to duplicate one of those 64 addresses.*

If the BREAKA command is entered without an `address` value, the halt in code execution clears the accumulator breakpoint. To cancel the accumulator breakpoint before the halt occurs, enter the BREAKA command without any parameter values. If the BREAKA command is entered without an `address` value, the accumulator breakpoint does not show in the BREAKPOINT WINDOW.

If the BREAKA command is entered with an address value, the accumulator breakpoint may be cleared by one of these methods:

- Enter the NOBR command.
- Position the cursor on that address in the code window. Then press the right mouse button and select **Toggle Breakpoint at Cursor** menu item.

Set Accumulator Breakpoint (continued) BREAKA

Syntax: `BREAKA [<n> [<address>]]`

Where:

`<n>`	Accumulator value that triggers a break in execution
`<address>`	Optional address for the break in execution (provided that the accumulator value equals `n`)

Use with: ICS08GPW only

Examples:

`BREAKA 55`	Break execution when the accumulator value equals 55
`BREAKA`	Cancel the accumulator breakpoint.
`BREAKA 55 300`	Break execution at address 300 if accumulator value equals 55

BREAKHX Set HX Register Breakpoint

The BREAKHX command sets an HX register breakpoint and breaks code execution when the value of the HX register equals the specified `n` value.

With an `n` value, the command forces a break in execution as soon as the accumulator value equals `n`.

With `n` and `address` value, the command forces a break in execution when the accumulator value equals `n` and execution arrives at the specified address. If the accumulator value changes from `n` by the time execution arrives at the address, no break occurs.

NOTE: *The maximum number of breakpoint addresses is 64. Each BR, BREAKA, BREAKSP, or BREAKHX command that includes an address value uses an additional breakpoint address, unless the address is a duplicate. For example, if 64 BR commands already have taken up 64 addresses, the only way to include an address value in a BREAKA, BREAKSP, or BREAKHX command is to duplicate one of those 64 addresses.*

If the BREAKHX command is entered without an address value, the break in code execution clears the accumulator breakpoint. To cancel the accumulator breakpoint before the break occurs, enter the BREAKHX command without any parameter values. If the BREAKHX command is entered without an address value, the accumulator breakpoint does not show in the Breakpoint window.

If the BREAKHX command is entered with an address value, the accumulator breakpoint may be cleared by:

1. Enter the NOBR command.
2. Position the cursor on that address in the code window. Then press the right mouse button and select the **Toggle Breakpoint at Cursor** menu item.

Set HX Register Breakpoint (continued) BREAKHX

Syntax: `BREAKHX [<n> [<address>]]`

Where:

`<n>`	Index register value that triggers a break in execution
`<address>`	Optional address for the break in execution (when the index register value equals `n`)

Use with: ICS08GPW only

Examples:

`BREAKHX A9`	Break execution when the HX register value equals A9
`BREAKHX`	Cancel the HX register breakpoint.
`BREAKHX A9 400`	Break execution at address 400 if HX register value equals A9

BREAKSP Set Stack Pointer Breakpoint

The BREAKSP command sets a stack pointer breakpoint to halt code execution when the value of the stack pointer equals a specified value.

- With an `n` value, the command forces a break in execution as soon as the stack pointer value equals `n`.
- With `n` and `address` values, the BREAKSP command forces a halt in execution when the stack pointer value equals `n` and execution arrives at the specified address. If the stack pointer value changes from `n` by the time execution arrives at the address, no break occurs.

NOTE: *The maximum number of breakpoint addresses is 64. Each BR, BREAKA, BREAKSP, or BREAKHX command that includes an address value uses an additional breakpoint address, unless the address is a duplicate. For example, if 64 BR commands already have taken up 64 addresses, the only way to include an address value in a BREAKA, BREAKSP or BREAKHX command is to duplicate one of those 64 addresses.*

If the BREAKSP command is entered without an `address` value, the halt in code execution clears the stack pointer breakpoint. To cancel the stack pointer breakpoint before the halt occurs, enter the BREAKSP command without any parameter values. If the BREAKSP command is entered without an `address` value, the stack pointer breakpoint does not show in the Breakpoint window.

If the BREAKSP command is entered with an address value, the stack pointer breakpoint may be cleared by one of these methods:

- Enter the NOBR command.
- Position the cursor on that address in the code window. Then press the right mouse button and select **Toggle Breakpoint at Cursor** menu item.

Set Stack Pointer Breakpoint (continued) BREAKSP

Syntax: `BREAKSP [<n> [<address>]]`

Where:

`<n>`	Stack pointer value that triggers a break in execution
`<address>`	Optional address for the break in execution (when the stack pointer value equals `n`)

Use with: ICS08GPW only

Examples:

`BREAKSP E0`	Break execution when the stack pointer (SP) value equals E0
`BREAKSP`	Cancel the SP breakpoint.
`BREAKSP E0 300`	Break execution at address 300 if SP value equals E0

C — Set/Clear Carry Bit

The C command sets or clears the C bit of the condition code register (CCR).

NOTE: *The CCR bit designators are in the lower portion 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`

Use with: ICS08GPW and ICD08SW

Examples:

`C 0`	Clears the C bit of the CCR
`C 1`	Sets the C bit of the CCR

Capture Changed Data CAPTURE

The CAPTURE command specifies locations to be monitored for changes in value. If the value of such a location changes and if a capture file is open, the file records the change in value. (See the CAPTUREFILE or CF command for more information about capture files.)

To stop monitoring a location, specify that same location in another CAPTURE command or close the capture file. Closing the capture file undoes the specifications for all monitoring locations.

NOTE: *Before entering the CAPTURE command, open a capture file via the CAPTUREFILE or CF command. The CAPTURE command has no effect unless a capture file is open.*

Syntax: `CAPTURE <address> [<address>...]`

Where:

`<address>`	Location to be monitored for a change in value

Use with: ICS08GPW only

Examples:

`CAPTURE PORTA`	Monitor location PORTA for any value changes.
`CAPTURE C0`	Monitor RAM location C0 for any value changes.
`CAPTURE D0 D1 D2`	Monitor for any value changes in an array of locations.

CAPTUREFILE or CF — Open/Close Capture File

The CAPTUREFILE command opens a capture file to record changed values. If the specified file does not yet exist, this command creates the file. 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 parameter is omitted, a prompt asks for this overwrite/append choice.

The command interpreter does not assume a filename extension for the capture file. To close the capture file, enter this command without any parameter values.

The CF and CAPTUREFILE commands are identical. If no CAPTURE command has specified locations to be monitored, the CF and CAPTUREFILE commands have no effect.

NOTE: *The CAPTURE command specifies the location to be monitored for value changes. Closing the capture file deletes the location specification. The simulator continues writing to an open capture file. The capture file must be closed within a reasonable time to prevent the file from growing too large.*

Syntax:

```
CAPTUREFILE [<filename> [R | A]]
```

Where:

`<filename>`	Name of the capture file

Use with: ICS08GPW only

Examples:

`CAPTUREFILE TEST.CAP`	Open capture file TEST.CAP.
`CF TEST4.CAP A`	Open capture file TEST4.CAP; append new entries.

Set Condition Code Register — CCR

The CCR command sets the condition code register (CCR in the CPU) to the specified hexadecimal value. The value entered with the command displays in the CPU window.

NOTE: *The CCR bit designators are in the lower portion of the CPU window. The CCR binary 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 is set; a period means that the corresponding bit is clear.*

Syntax: `CCR <n>`

Where:

`<n>`	The new hexadecimal value for the CCR

Use with: ICS08GPW and ICD08SW

Example:

`CCR E4`	Assign the value E4 to the CCR. This makes the binary pattern 11100100; the N bit set, other bits clear.

CGMXCLK Set CGMXCLK to Bus Clock Relationship

Several peripherals can be clocked by either the bus clock or the CGMXCLK. To properly simulate these modules with respect to the bus clock, properly set the CGMXCLK to bus clock relationship. The CGMXCLK command allows the user to specify how many bus cycles are equal to a certain number of CGMXCLK cycles. Changing these values automatically resets the processor to ensure proper peripheral initialization. This value needs to be specified only once per debug session, regardless of the number of resets. To consistently use a different setting, place the initialization in the startup macro STARTUP.08.

The following peripherals use or can use the CGMXCLK:

- Timebase module
- A/D (analog-to-digital) converter
- COP (computer operating properly) watchdog
- SCI serial communications interface

The default is 2 CGMXCLK cycles = 1 bus cycle in time. This would be the case when running the M68HC908GP Family of devices from a 4.9152-MHz clock oscillator with a 2.4576-MHz internal bus. When running a 32.768-kHz crystal with a 2.4576-MHz internal bus, set 75 bus cycles = 1 CGMXCLK cycle. Do this by using the command:

```
CGMXCLK 75 1
```

Try to keep the numbers involved as low as possible for more accurate simulation. For example, 1 and 4 are better than 201 and 800.

Syntax: `CGMCLK <n1> <n2>`

Where:

`<n1>`	Number of bus cycles
`<n2>`	Time equivalent CGMXCLK cycles

Set CGMXCLK to Bus Clock Relationship (continued) CGMXCLK

Use with: ICS08GPW only

Examples:

`CGMXCLK 75 1`	Bus/crystal = 2.4576 MHz bus / 32.768 kHz crystal
`CGMXCLK 1 2`	Bus/crystal = 2.4576 MHz bus / 4.9512 kHz crystal
`CGMXCLK`	Pops up a window asking for values

CHIPMODE — Choose Device for Simulation

The CHIPMODE command brings up a pop-up window containing all of the HC08 devices that can be simulated with ICS08GPW for Windows. The device can be selected from the window.

NOTE: *The selection of a new chip does not take effect until the next debugging session.*

Syntax: `CHIPMODE`

Use with: ICS08GPW only

Example:

`CHIPMODE`	Brings up a window for selecting the device for simulation

Clear .MAP File CLEARMAP

The CLEARMAP command removes the current MAP file from memory, forcing the debugger to show disassembled code in the Code windows instead of source code. Symbols defined using the SYMBOL command are not affected by this command.

For the ICS08GPW, the NOMAP command is identical to CLEARMAP.

Syntax: `CLEARMAP`

Use with: ICS08GPW and ICD08SW

Example: `CLEARMAP` Clears symbols and their definitions

CLEARSYMBOL — Clear User Symbols

The CLEARSYMBOL command removes all the user-defined symbols (created with the SYMBOL command). Debug information from MAP files, used for source level debugging, is not affected by the CLEARSYMBOL command.

NOTE: *List the current user-defined symbols using the SYMBOL command.*

Syntax:	`CLEARSYMBOL`	
Use with:	ICS08GPW and ICD08SW	
Example:	`CLEARSYMBOL`	Clears user-defined symbols

Show Disassembled Code CODE

The CODE command shows disassembled code in the Code window, starting at address `add`. Specifying an address in the middle of an intended instruction may cause improper results.

Syntax:	`CODE <add>`	
	Where:	
	`<add>`	The user code's starting address
Use with:	ICD08SW only	
Example:	`CODE 100`	Shows the disassembled code in the code window starting at hex address 100

COLORS — Set Simulator Colors

The COLORS command opens the **Change Window Colors** dialog that allows choosing the text and background colors for windows in the ICS08GPW simulator and ICD08SW debugger. After setting the colors options for the windows, save the changes using the SAVEDESK command. For more information about using the **Change Window Colors** dialog, see **6.21.2 Change Colors**.

Syntax: `COLORS`

Use with: ICS08GPW and ICD08SW

Example: `COLORS` Open the colors window.

Set Cycles Counter — CYCLES or CY

The CYCLES command changes the value of the cycle counter. The cycle counter counts the number of processor cycles that have passed during execution. The Cycle window shows the cycle counter. The cycle count can be useful for timing procedures.

If no parameter is specified, the current cycle count is displayed in the Status window. This is useful for capturing the cycle count to a logfile.

Syntax: `CYCLES [<n>]`

Where:

`<n>`	Integer value for the cycles counter

Use with: ICS08GPW only

Examples:

`CYCLES 0`	Reset cycles counter.
`CY 1000`	Set cycle-counter value to 1000.

DASM Disassemble Memory

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

- If the command includes an address value, one disassembled instruction is shown, beginning at that address.
- If a 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. In this case, enter the LF command to record the disassembled instructions in a logfile or use the scroll bars in the status window.*

Syntax:

```
DASM [<address> | <startrange> <endrange>]
```

Where:

`<address>`	First address of three instruction opcodes to be disassembled
`<startrange>`	Starting address for a range of instructions to be disassembled
`<endrange>`	Ending address for a range of instructions to be disassembled

Use with: ICS08GPW and ICD08SW

Examples:

```
DASM 300
0300                A6E8    LDA #0E8
DASM 200 208
0200                5F      CLRX
0201                A680    LDA #80
0203                B700    STA PORTA
0205                A6FE    LDA #FE
0207                B704    STA DDRA
```

Set Port A Direction Register — DDRA

The DDRA command assigns the specified byte value to the port A data direction register (DDRA). Bits assigned 0 denote input pins; bits assigned 1 denote output pins.

Syntax: `DDRA <n>`

Where:

`<n>`	The byte value to be placed into DDRA

Use with: ICS08GPW only

Examples:

`DDRA FF`	Set all port A pins to be outputs.
`DDRA 00`	Set all port A pins to be inputs.

DDRB — Set Port B Direction Register

The DDRB command assigns the specified byte value to the port B data direction register (DDRB). Bits assigned 0 denote input pins; bits assigned 1 denote output pins.

Syntax: `DDRB <n>`

Where:

`<n>`	The byte value to be placed into DDRB

Use with: ICS08GPW only

Examples:

`DDRB 03`	Set the lower two bits of port B pins as outputs; set the others to be inputs.
`DDRB FF`	Set all port B pins to be outputs.

Set Port C Direction Register DDRC

The DDRC command assigns the specified byte value to the port C data direction register (DDRC). Bits assigned 0 denote input pins; bits assigned 1 denote output pins.

Syntax: `DDRC <n>`

Where:

`<n>`	The byte value to be placed into DDRC

Use with: ICS08GPW only

Examples:

`DDRC 03`	Set the lower two bits of port C pins as outputs; set the others to be inputs.
`DDRC FF`	Set all port C pins to be outputs.

DDRD — Set Port D Direction Register

The DDRD command assigns the specified byte value to the port D data direction register (DDRD). Bits assigned 0 denote input pins; bits assigned 1 denote output pins.

Syntax: `DDRD <n>`

Where:

`<n>`	The byte value to be placed into DDRD

Use with: ICS08GPW only

Examples:

`DDRD 03`	Set the lower two bits of port D pins as outputs; set the others to be inputs.
`DDRD FF`	Set all port D pins to be outputs.

Set Port E Direction Register — DDRE

The DDRE command assigns the specified byte value to the port E data direction register (DDRE). Bits assigned 0 denote input pins; bits assigned 1 denote output pins.

Syntax: `DDRE <n>`

Where:

`<n>`	The byte value to be placed into DDRE

Use with: ICS08GPW only

Examples:

`DDRE 03`	Set the lower two bits of port E pins as outputs; set the others to be inputs.
`DDRE FF`	Set all port E pins to be outputs.

DUMP — Dump Memory to Screen

The DUMP command sends contents of a block of 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: *Sometimes the DUMP command causes the memory contents to scroll through the debug window too rapidly to view. In this case, enter the LF command, to record the memory locations in a logfile, or use the scroll bars in the status window.*

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

Where:

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

Use with: ICS08GPW and ICD08SW

Examples:

`DUMP C0 CF`	Dump array of RAM values in bytes
`DUMP.W 300 37S`	Dump ROM code in address 300-37S in words
`DUMP.B 200 300`	Dump contents of addresses 200-300 in rows of eight bytes

Evaluate Expression EVAL

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: *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 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

Use with: ICS08GPW and ICD08SW

Examples:

```
EVAL 45 + 32
0077H 119T 0001670 0000000001110111Q  "w"
EVAL 100T
0064H 100T 0001440 0000000001100100Q  "d"
```

EXIT or QUIT — Exit/Quit Application

The EXIT command terminates the software and closes all windows. The QUIT command is identical to EXIT.

Syntax: `EXIT`

Use with: ICS08GPW and ICD08SW

Example: `EXIT` Finish working with the program.

Begin Program Execution — G, GO, or RUN

The identical G, GO, and RUN commands start execution of code at the current program counter (PC) address or at an optional specified address.

If only one address is entered, that address is the starting address. Execution continues until a key is pressed, until it arrives at a breakpoint, or until an error occurs.

If a second address is entered, execution stops at that address.

In the ICS08GPW simulator, this command causes the host computer to simulate instructions as fast as it can. However, execution will be much slower than real-time execution.

In the ICD08W debugger, this command starts real-time execution by the MC68HC908GP20 or MC68HC908GP32 processors.

NOTE: *To see the windows updated with information during execution of code, use the STEPFOR command.*

Syntax:

```
G [[startaddr] [endaddr]]
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

Use with: ICS08GPW and ICD08SW

Examples:

`GO`	Begin code execution at the current PC value.
`GO 346`	Begin code execution at address 346.
`G 300 371`	Begin code execution at address 300. End code execution just before the instruction at address 371.
`RUN 300`	Begin code execution at address 300.

GOEXIT Execute Without Breakpoints/Debugger

The GOEXIT command is similar to the GO command, except that the target is left running without any breakpoints and the debugger software is terminated.

Syntax: `GOEXIT <addr>`

Where:

`<addr>`	Starting address of user code

Use with: ICD08SW only

Example:

`GOEXIT 100`	Sets the program counter to location 100 hex, runs the program, and exits from background debug mode.

Execute Past Subroutine/Interrupt GONEXT

The GONEXT command executes from the current PC address until the next instruction is reached. It is used to execute past a subroutine call or past intervening interrupts. Some debuggers refer to this functionality as "step over."

Syntax: `GONEXT`

Use with: ICD08SW only

Example: `GONEXT`

GOMACRO — Execute Macro after Break

The GOMACRO command executes the program in the simulator beginning at the address in the program counter (PC). Execution continues until a key is pressed, until it arrives at a breakpoint, or until an error occurs. Afterward, it runs the specified macro file just like the MACRO command.

Syntax: `GOMACRO <filename>`

Where:

`<filename>`	The name of a script file to be executed, with or without extension .MAC, or a pathname that includes an asterisk (*) wildcard character. When the asterisk is entered, the command displays a list of appropriate files, from which the required file can be selected.

Use with: ICS08GPW only

Example: `GOMACRO AVCALC.MAC` Begin code execution at the current PC value; at breakpoint execute macro AVCALC.MAC.

Execute Until Address GOTIL

The GOTIL command executes code beginning at the address in the program counter (PC). Execution continues until the program counter contains the specified ending address, until a key or the STOP button on the ICS08GPW toolbar is pressed, until it reaches a breakpoint, or until an error occurs.

Syntax: `GOTIL <endaddr>`

Where:

`<endaddr>`	The address at which execution stops

Use with: ICS08GPW and ICD08SW

Example:	`GOTIL 2F0`	Executes code up to address 2F0

GOTOCYCLE — Execute to Cycle Counter Value

The GOTOCYCLE command executes the program in the simulator beginning at the address in the program counter (PC). Execution continues until the cycle counter is equal to or greater than the specified value, until a key or the STOP button on the ICS08GPW toolbar is pressed, until it reaches a breakpoint, or until an error occurs.

Syntax: `GOTOCYCLE <n>`

Where:

`<n>`	Cycle-counter value at which execution stops

Use with: ICS08GPW only

Example:	`GOTOCYCLE 100`	Execute the program until the cycle counter equals 100.

Set/Clear Half-Carry Bit — H or HREG

The H or HREG command sets or clears the H bit in the condition code register (CCR).

NOTE: *The CCR bit designators are in the lower portion 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: `H 0|1`

Use with: ICS08GPW and ICD08SW

Examples:

`H 1`	Sets the H bit in the CCR
`H 0`	Clears the H bit of the CCR

HELP — Open Help

The HELP command opens the Windows help file for the program. An alternative way to open the help system is to press the `F1` key. If entered with an optional parameter, help appears for the specified topic. If no parameter is entered, the entire help file appears.

Syntax: `HELP <topic>`

Where:

`<topic>`	A debug command or assembly instruction.

Use with: ICS08GPW and ICD08SW

Examples:

`HELP`	Open the help file.
`HELP asm`	Displays help for the ASM debugging command.

Set H:X Index Register Pair — HX

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

Use with: ICS08GPW and ICD08SW

Example:

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

I — Set/Clear Interrupt Mask

The I command sets or clears the I bit of the condition code register (CCR).

The CCR bit designators are in the lower portion 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: `I <0 or 1>`

Use with: ICS08GPW and ICD08SW

Examples:

`I 1`	Set the I bit in the CCR.
`I 0`	Clear the I bit of the CCR.

Display Line Information INFO

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

Before executing this command, select a line of code in the Code window by clicking on it with the left mouse button.

Syntax: `INFO`

Use with: ICS08GPW and ICD08SW

Example:

`INFO`	Display information about the cursor line.
`Filename:PODTEST.ASM`	Line number:6
`Address:$0100`	
`Disassembly:START   5F    CLRX`	

INPUTA Set Port A Inputs

The INPUTA command sets the simulated inputs to port A. The CPU reads this input value when port A is set as an input port.

NOTE: *If the ICS08GP circuit board is connected, port A inputs come from the board, so this command has no effect.*

Syntax: `INPUTA <n>`

Where:

`<n>`	Eight-bit simulated value for port A

Use with: ICS08GPW only

Example:

`INPUTA AA`	Simulate the input AA on port A.

Set Port B Inputs INPUTB

The INPUTB command sets the simulated inputs to port B. The CPU reads this input value when port B is set as an input port.

NOTE: *If the ICS08GP circuit board is connected, port B inputs come from the board, so this command has no effect.*

Syntax: `INPUTB <n>`

Where:

`<n>`	Eight-bit simulated value for port B

Use with: ICS08GPW only

Example:

`INPUTB 01`	Simulate the input 01 on port B.

INPUTC Set Port C Inputs

The INPUTC command sets the simulated inputs to port C. The CPU reads this input value when port C is set as an input port.

NOTE: *If the ICS08GP circuit board is connected, port C inputs come from the board, so this command has no effect.*

Syntax: `INPUTC <n>`

Where:

`<n>`	Eight-bit simulated value for port C

Use with: ICS08GPW only

Example: `INPUTC 01` Simulate the input 01 on port C.

Set Port D Inputs INPUTD

The INPUTD command sets the simulated inputs to port D. The CPU reads this input value when port D is set as an input port.

NOTE: *If the ICS08GP circuit board is connected, port D inputs come from the board, so this command has no effect.*

Syntax: `INPUTD <n>`

Where:

`<n>`	Eight-bit simulated value for port D

Use with: ICS08GPW only

Example: `INPUTD 01` Simulate the input 01 on port D.

INPUTE — Set Port E Inputs

The INPUTE command sets the simulated inputs to port E. The CPU reads this input value when port E is set as an input port.

NOTE: *If the ICS08GP circuit board is connected, port E inputs come from the board, so this command has no effect.*

Syntax: `INPUTE <n>`

Where:

`<n>`	Eight-bit simulated value for port E

Use with: ICS08GPW only

Example:

`INPUTE 01`	Simulate the input 01 on port E.

Show Port Inputs INPUTS

The INPUTS command shows the simulated input values to port A and B (entered via the INPUTA or INPUTB commands).

NOTE: *If the ICS08GP circuit board is connected, this command shows input values from the board.*

Syntax: `INPUTS`

Use with: ICS08GPW only

Example: `INPUTS` Show I/O port input values.

Port A — AA

Port B — 01

INT or IRQ — Set IRQ Pin State

The IRQ command assigns the state value of the MCU IRQ pin. To see the current simulated value on the pin, enter this command without any parameter value. The external interrupt is simulated as a level or edge/level triggered interrupt, depending on the IRQ bit in the MOR (mask option register).

NOTE: *If the M68ICS08GP20 hardware pod is connected, then the IRQ pin level comes from the circuit board, and this command cannot be used to modify its value.*

Syntax: `IRQ [0 | 1]`

Use with: ICS08GPW and ICD08SW

Examples:

`INT 0`	Assign 0 to the IRQ pin.
`IRQ 1`	Assign 1 to the IRQ pin.

Open/Close Logfile — LF or LOGFILE

The LOGFILE command opens an external file to receive log entries of the commands entered in the command line of the Status window and the system responses to those commands that appear in the Status window message area.

- If the specified file does not exist, this command creates the file.
- If the specified file exists, an optional parameter can be entered to specify whether to overwrite existing contents (R, the default) or to append the log entries (A). If this parameter is omitted, a prompt window asks whether to overwrite the existing file or append information to the existing file.

While logging is in effect, any line appended to the Status window is also written to the logfile.

Logging continues until another LOGFILE or LF command is entered without any parameter values. This second command disables logging and closes the logfile.

Data can be captured by opening a logfile, executing a command that dumps data to the Status window, and then closing the logfile.

The command interpreter does not assume a filename extension.

Syntax: `LF [<filename>  [<R | A>]]`

Where:

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

Use with: ICS08GPW and ICD08SW

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 logfile.

LISTOFF Turn Off Step Listing

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`	
Use with:	ICS08GPW only	
Example:	`LISTOFF`	Do not show step information.

Turn On Step Listing LISTON

The LISTON command turns on the screen listing of the step-by-step information during stepping. The register values and program instructions are displayed in the Status window while running 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`

Use with: ICS08GPW only

Example: `LISTON` Show step information.

LOAD — Load S Records

The LOAD command loads an S-record (*.S19) object file into the debugger. Entering this command without a filename brings up a list of .S19 files in the current directory. Select a file for loading from this list. Upon loading, if the reset vector is defined in the code, the debugger sets the PC to that address.

When used with ICS08GPW, the associated map file is also loaded. When used with ICD08SW, the map file is *not* loaded; use the LOADALL or LOADMAP command for this purpose.

Syntax: `LOAD [<filename>]`

Where:

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

Use with: ICS08GPW and ICD08SW

Examples:

`LOAD PROG1.S19`	Load file PROG1.S19 and its map file into the simulator at the load addresses in the file.
`LOAD PROG2`	Load file PROG2.S19 and its map file into the simulator 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 S Records and Map File — LOADALL

The LOADALL command loads both the S19 object file and the map file into the debugger. It is equivalent to executing both the LOAD and LOADMAP commands.

Syntax: `LOADALL [<filename>]`

Where:

`<filename>`	Filename of the user source code

Use with: ICD08SW only

Example:

`LOADALL myprog`	Loads both the S19 object file and the map file

LOADDESK — Load Desktop Settings

The LOADDESK command loads the debugger window (desktop) settings for window position, size, and visibility, allowing a choice of how to set up the windows for the project.

This command executes automatically whenever the debugger is started.

Use the SAVEDESK command to save the debugger window settings to the desktop file.

Syntax:	`LOADDESK`	
Use with:	ICS08GPW and ICD08SW	
Example:	`LOADDESK`	Get window settings from desktop file.

Load Map File LOADMAP

The LOADMAP command loads into the debugger a map file that contains source-level debug information. 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.

This command does *not* load an object file. It is useful in ICD08SW when the object file has been previously programmed into FLASH memory.

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 list of all files in the specified directory that have the .MAP extension.

Use with: ICS08GPW and ICD08SW

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:`	Display the names of the .MAP files on the diskette in drive A:.
`LOADMAP`	Display the names of the .MAP files in the current directory.

LOADV — Load and Verify

The LOADV command first executes the LOAD command, then automatically executes the VERIFY command.

Syntax: `LOADV [<filename>]`

Where:

`<filename>`	Filename of user source code.

Use with: ICD08SW only

Example:

`LOADV myprog`	Loads the S19 into the target. The contents of the S19 file on the target board are then compared with the file myprog.

Load Binary File LOAD_BIN

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

Syntax: `LOAD_BIN <filename> <add>`

Where:

`<filename>`	Name of the binary file
`<add>`	Starting address

Use with: ICD08SW only

Example: `LOAD_BIN myfile 100` Loads a binary myfile of bytes starting at hex address 100.

LOADV_BIN — Load and Verify Binary File

The LOADV_BIN command first executes the LOAD_BIN command, then automatically executes the VERIFY command.

Syntax: `LOADV_BIN <filename> <add>`

Where:

`<filename>`	Name of the binary file
`<add>`	Starting address

Use with: ICD08SW only

Example: `LOADV_BIN myfile 100` — Loads a binary myfile of bytes starting at hex address 100, then executes a VERIFY command.

Execute Batch File — MACRO

The MACRO command executes a macro file, which is a text file that contains a sequence of debug commands. Executing the macro file has the same effect as executing the individual commands, one after another. The SCRIPT command is identical.

Entering this command without a filename value brings up a list of macro (.MAC) files in the current directory. The file for execution can be selected directly from this list.

NOTE: *A macro file can contain the MACRO command, allowing nested macro files up to 16 levels deep.*

The most common use of the REM and WAIT commands is within macro files. The REM command displays comments while the macro file executes. The WAIT command establishes a pause between the execution of the macro file commands.

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

To create a macro file, use either a text editor or the MACROSTART and MACROEND commands.

Syntax: `MACRO <filename>`

Where:

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

Use with: ICS08GPW and ICD08SW

MACRO — Execute Batch File (continued)

Examples:	`MACRO INIT.MAC`	Execute commands in file INIT.MAC.
	`SCRIPT *`	Display names of all .MAC files, then execute the selected file.
	`MACRO A:*`	Display names of all .MAC files in drive A, then execute the selected file.
	`MACRO`	Display names of all .MAC files in the current directory, then execute the selected file.

Stop Saving Commands to Batch File MACROEND

The MACROEND command stops recording of the macro file in which the software has saved debug commands. (The MACROSTART command opened the macro file). The recorded macro file is closed and left ready for use by the MACRO command.

Syntax: `MACROEND`

Use with: ICS08GPW and ICD08SW

Example: `MACROEND` Stop saving debug commands to the macro file, then close the file.

MACROSTART — Save Debug Commands to Batch 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 may 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.

Use with: ICS08GPW and ICD08SW

Example: `MACROSTART TEST.MAC` Save debug commands in macro file TEST.MAC

List Macros MACS

The MACS command brings up a window with a list of macros. These are files with the extension .ICD (such as the STARTUP.ICD macro). Use the arrow keys and the ENTER key or the mouse to select. Cancel with the ESC key.

Syntax: `MACS`

Use with: ICD08SW only

Example: `MACS` Displays a window with a list of macros

MAP — Show Information in Map File

The MAP command allows viewing of information from the current map file stored in 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 and SHOWMAP commands are identical.

Syntax: `MAP`

Use with: ICS08GPW and ICD08SW

Example:

Example:	`MAP`	Shows symbols from the loaded map file and their values

Display Memory at Address — MD or MD1

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 displayed. If a logfile is open, this command also writes the first 16 bytes to the logfile.

For ICS08GPW, the SHOW command is identical.

For ICD08SW, the display is in the *first* memory window.

Syntax: `MD <address>`

Where:

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

Use MD with: ICS08GPW and ICD08SW

Use MD1 with: ICD08SW only

Examples:

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

MD2 Display Memory (Window 2) at Address

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:

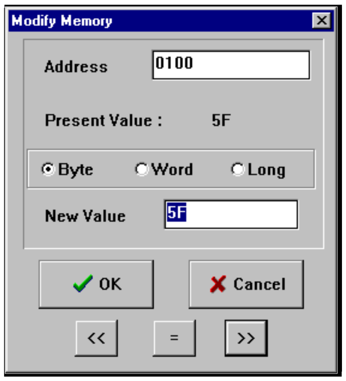
`<address>`	The starting memory address for display in the upper left corner of the memory window.

Use with: ICD08SW only

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

Modify Memory MM or MEM

The MM or MEM 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, however, the command has only an address value, the **Modify Memory** dialog (see **Figure 10-2**) appears, showing the specified address and its present value. Use the dialog to enter a new value for the address or to modify the address type by selecting 8-bit bytes, 16-bit words, or 32-bit longs. To modify several memory locations from this dialog, enter the new value in the **New Value** text box and click the >> button to increment the current address, the << button to decrement the current address, or the = button to display the same address. For the ICD08SW, the MEM command is identical.


Modify Memory

Address 0100

Present Value : 5F

[✓ Byte] Word Long

New Value 5F

Figure 10-2. Modify Memory Dialog Window

If macro recording is on, all the values written to memory are recorded and will be properly written to memory when the macro is played back. See the MACROSTART command.

If the MM command includes optional data values, the software assigns the values to the specified addresses sequentially, then the command ends. No window appears in this case.

MM or MEM Modify Memory (continued)

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)

Use with: ICS08GPW and ICD08SW

Examples:

`MM 90`	Start memory modify at address $90.
`MM 300 00`	Assign value 00 to address $300.
`MM 100 00 01 10 11`	Assign values 00–11 to bytes 100–103.
`MM.L 200 123456`	Place long value $123456 at address $200.

Set/Clear Negative Bit N

The N command sets or clears the N bit of the condition code register (CCR).

NOTE: *The CCR bit designators are in the lower portion 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: `N 0|1`

Use with: ICS08GPW and ICD08SW

Example:

Example:	`N 1`	Set the N bit of the CCR.
	`N 0`	Clear the N bit of the CCR.

NOBR — Remove Breakpoints

The NOBR command removes one or all active breakpoints. If this command is entered with an address value, it removes the breakpoint at that address. Without the address parameter, it removes all current breakpoints. To set breakpoints, use the BR command.

An alternative way for clearing a breakpoint in the Code window is to position the cursor on a line of code, click the left mouse button to select the line, then press the right mouse button and select the **Toggle Breakpoint at Cursor** menu item. This removes the breakpoint from the line.

Syntax: `NOBR [<address>]`

Where:

`<address>`	Optional address of a single breakpoint to be removed

Use with: ICS08GPW and ICD08SW

Examples:

`NOBR`	Remove all current instruction breakpoints.
`NOBR 120`	Remove the instruction breakpoint at address 120.

Clear .MAP File NOMAP

The NOMAP command removes the current MAP file from memory, forcing the debugger to show disassembled code in the Code windows instead of source code. Symbols defined using the SYMBOL command are not affected by this command.

For the ICS08GPW, the NOMAP command is identical to CLEARMAP.

Syntax: `NOMAP`

Use with: ICS08GPW and ICD08SW

Example:	`NOMAP`	Clears symbols and their definitions

NOSYMBOL Clear User Symbols

The NOSYMBOL command removes all user-defined symbols created using the SYMBOL from memory. Symbols are created using the SYMBOL command. Symbols defined via a loaded MAP file are not affected.

Syntax: `NOSYMBOL`

Use with: ICS08GPW and ICD08SW

Example: `NOSYMBOL` Clears user-defined symbols and their definitions

Set Program Counter PC

The PC command assigns the specified value to the MCU program counter. 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; the code windows change accordingly. The value entered with the command is displayed in the CPU Window.

An alternative way for setting the PC in a code window is to position the cursor on a line of code, click the left mouse button to select the line, 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

Use with: CS08GPW and ICD08SW

Example: `PC 0200` Sets the PC value to 0200

POD Change Serial Port

The POD command connects to the ICS08GP circuit board through the specified serial (COM) port. If successful, this command responds with the current status of ports, reset, and IRQ pins on the board. The command also shows the version of the board.

This command is used to change from stand-alone mode (no hardware pod attached to the host computer) to in-circuit simulation mode (pod attached). To change back to stand-alone mode, use the SIM08 command.

Syntax: `POD <n>`

Where:

`<n>`	The number (1...8) of a serial port (COM1 through COM8) on the PC

Use with: ICS08GPW only

Example: `POD 1` Connect to serial port COM1.

Port A — 80

Port B — 00

Reset — 1

IRQ — 1

Version — 01

Set Port A Output Latches — PORTA or PRTA

The PORTA command assigns the specified value to the port A output register latches. The PRTA command is an alternate form of the PORTA command.

NOTE: *If the M68ICS08GP20 hardware pod is connected, the system sends the n parameter value of this command to the board.*

Syntax: `PORTA <n>`

Where:

`<n>`	The new value for the port A output latches

Use with: ICS08GPW only

Example: `PORTA FF` Set all port A output latches high.

PORTB or PRTB — Set Port B Output Latches

The PORTB command assigns the specified value to the port B output register latches. The PRTB command is an alternate form of the PORTB command.

NOTE: *If the M68ICS08GP20 hardware pod is connected, the system sends the n parameter value of this command to the board.*

Syntax: `PORTB <n>`

Where:

`<n>`	The new value for the port B output latches

Use with: ICS08GPW only

Example:

`PORTB 03`	Set the port B output latches to 03.

Set Port C Output Latches — PORTC or PRTC

The PORTC command assigns the specified value to the port C output register latches. The PRTC command is an alternate form of the PORTC command.

NOTE: *If the M68ICS08GP20 hardware pod is connected, the system sends the n parameter value of this command to the board.*

Syntax: `PORTC <n>`

Where:

`<n>`	The new value for the port C output latches

Use with: ICS08GPW only

Example:

`PORTC 03`	Set the port C output latches to 03.

PORTD or PRTD — Set Port D Output Latches

The PORTD command assigns the specified value to the port D output register latches. The PRTD command is an alternate form of the PORTD command.

NOTE: *If the M68ICS08GP20 hardware pod is connected, the system sends the n parameter value of this command to the board.*

Syntax: `PORTD <n>`

Where:

`<n>`	The new value for the port D output latches

Use with: CS08GPW only

Example:

`PORTD 03`	Set the port D output latches to 03.

Set Port E Output Latches — PORTE or PRTE

The PORTE command assigns the specified value to the port E output register latches. The PRTE command is an alternate form of the PORTE command.

NOTE: *If the M68ICS08GP20 hardware pod is connected, the system sends the n parameter value of this command to the board.*

Syntax: `PORTE <n>`

Where:

`<n>`	The new value for the port E output latches

Use with: ICS08GPW only

Example: `PORTE 03` Set the port E output latches to 03.

QUIET — Toggle Window Refresh

The QUIET command toggles the refresh of memory-based windows on or off. The default is ON. This command can be used on the startup command line.

Turning refresh off dramatically increases the stepping rate, since the data windows are not continually updated.

Syntax:	`QUIET`	
Use with:	ICD08SW only	
Example:	`QUIET`	Turns refresh of memory-based windows on or off

Use Register Files R

The R command pulls up windows for the register files and starts interactive setup of such system registers as the I/O, timer, and COP.

Entering this command opens the register files window, which can present a list of peripheral modules for the MC68HC908GP20 or MC68HC908GP32 MCUs. The user can view any of the registers, modify their values, and store the results back into memory.

An alternate way to bring up the register files window is to press the REGISTER FILES speed button.

Syntax:	R	
Use with:	ICS08GPW and ICD08SW	
Example:	R	Start interactive system register setup.

REG — Show Registers

The REG command displays the contents of the CPU registers in the Status window. The STATUS command is identical to the REG command.

This command is useful for capturing the CPU state to an open logfile.

Syntax: `REG`

Use with: ICS08GPW and ICD08SW

Example: `REG` Displays the contents of the CPU registers

Place Comment in Batch/Macro File — REM

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

This command is useful for recording comments in a logfile.

Syntax: `REM <text>`

Where:

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

Use with: ICS08GPW and ICD08SW

Example:

`REM Program executing;`	Display the message `Program executing` during macro file execution.

RESET — Simulate Processor Reset

The RESET command resets the MCU and sets the program counter (PC) to the contents of the reset vector. This command does not start execution of user code.

Syntax:	`RESET`	
Use with:	ICS08GPW and ICD08SW	
Example:	`RESET`	Reset the MCU.

Reset and Restart MCU RESETGO

The RESETGO command simulates a reset of the MCU, sets the program counter (PC) to the contents of the reset vector, and then starts execution from that address.

Syntax: `RESETGO`

Use with: ICS08GPW only

Example:	`RESETGO`	Simulate reset of the MCU and start execution of code.

SCCLR Clear SCI I/O Buffers

The SCCLR command can be used to flush the input and output buffers for SCI simulation. This will reset the circular buffers and clear out all values. Notice that if the SCI is currently shifting a value, this command will not prevent the SCI from finishing that transfer. See SCDI and SCDO for accessing the input and output buffers of the SCI interface.

If the M68ICS08GP20 hardware pod is connected, SCI inputs and outputs are directed from the pod and this command has no effect.

Syntax: `SCCLR`

Use with: ICS08GPW only

Example: `SCCLR` Clear input and output buffers for SCI simulation.

Input SCI Data — SCDI

The SCDI command allows the user to input data into the SCI. If a data parameter is given, the value is placed into the next slot in the circular input buffer. If no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SCI. The maximum number of input values is 256 bytes.

If the M68ICS08GP20 hardware pod is connected, SCI inputs and outputs are directed from the pod, so this command has no effect.

ICS08GPW simulates the correct number of cycles for one character, the character appears in the SCI data register, and the appropriate flags are set.

Syntax: `SCDI [<n>]`

Where:

`<n>`	The value to be entered into the next location in the input buffer

Use with: ICS08GPW only

Example:

`SCDI $55`	Set the next input value to the SCI to $55.
`SCDI`	Pull up the data window with all the input values.

SCDO — View SCI Outputs

The SCDO command displays the output circular buffer from the SCI. A window is opened that shows all the data that the SCI has shifted out. An arrow is used to point to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

If the ICS08 circuit board is connected, SCI inputs and outputs are directed from the board, so this command has no effect.

Syntax:	`SCDO`	
Use with:	ICS08GPW only	
Example:	`SCDO`	View data from the output buffer for SCI simulation.

Save Desktop Settings SAVEDESK

The SAVEDESK command saves the window size/position and desktop settings for the application when it is first opened or for use with the LOADDESK command. Opening the application or entering the LOADDESK command loads the saved settings.

Syntax: `SAVEDESK`

Use with: ICS08GPW and ICD08SW

Example: `SAVEDESK` Save window settings for the application.

SHOWBREAKS — Display Breakpoint Window

The SHOWBREAKS command brings up the Breakpoint window that displays the breakpoints used in the current debugging session. Breakpoints can be modified through this window.

Syntax: `SHOWBREAKS`

Use with: ICS08GPW only

Example: `SHOWBREAKS` Open the breakpoint window.

Display Code at Address — SHOWCODE

The SHOWCODE command displays code in the Code window 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

Use with: CS08GPW and ICD08SW

Example: `SHOWCODE 200` Show code starting at location $200.

SHOWMAP — Show Information in Map File

The SHOWMAP command allows viewing of information from the current map file stored in 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 and SHOWMAP commands are identical.

Syntax: `SHOWMAP`

Use with: ICS08GPW and ICD08SW

Example:	`SHOWMAP`	Show symbols from the loaded map file and their values.

Display Code at PC Address SHOWPC

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. All values, registers, and code currently displayed are recorded in the logfile.

This command is often useful immediately after the SHOWCODE command.

Syntax:	`SHOWPC`	
Use with:	ICS08GPW and ICD08SW	
Example:	`SHOWPC`	Show code from the PC address value.

SHOWTRACE Display Trace Window

The SHOWTRACE command displays the trace window, showing the last 1024 instructions that were executed after the TRACE command is used.

Syntax: `SHOWTRACE`

Use with: ICS08GPW only

Example: `SHOWTRACE` Open the trace window.

Switch Simulation Mode SIM08

The SIM08 command allows switching from in-circuit simulation (*with* the M68ICS08GP20 hardware pod connected to the host computer) to stand-alone simulation (*without* the pod connected).

Syntax: `SIM08`

Use with: ICS08GPW only

Example:	`SIM08`	Switch from in-circuit simulation to stand-alone simulation.

SNAPSHOT — Save Window Data to Logfile

The SNAPSHOT command sends textual information about the debugger windows to the open logfile. If no logfile is open, the command has no effect. This command is useful for documentation and testing.

Syntax: `SNAPSHOT`

Use with: ICS08GPW and ICD08SW

Example:

`LOGFILE SNAPSHOT`	Opens a logfile named SNAPSHOT.LOG and stores all information that appears in the Status window.
`SNAPSHOT`	Takes a snapshot of all open windows and stores it in the file SNAPSHOT.LOG.
`LF`	Close the SNAPSHOT.LOG file.

The SNAPSHOT.LOG file can now be opened with any text editor.

Set Path for Source Code SOURCEPATH

The SOURCEPATH command sets the default path for source code that is not in the current directory.

Syntax: `SOURCEPATH <pathname>`

Where:

`<pathname>`	The full path and filename of the source file

Use with: ICD08SW only

Example: `SOURCEPATH` d:\mysource\myfile.asm

SP Set Stack Pointer

The SP command assigns the specified value to the stack pointer (SP) used by the CPU. The value entered with the command should be reflected in the CPU window.

Syntax: `SP <n>`

Where:

`<n>`	The new stack pointer value

Use with: ICS08GPW and ICD08SW

Example:

`SP $E0`	Set the stack pointer value to $E0.

Clear SPI I/O Buffers SPCLR

The SPCLR command can be used to flush the input and output buffers for SPI simulation. This will reset the circular buffers and clear out all values. Notice that if the SPI is currently shifting a value, this command will not prevent the SPI from finishing that transfer. See SPDI and SPDO for accessing the input and output buffers of the SPI interface.

If the ICS08 circuit board is connected, SPI inputs and outputs are directed from the board, so this command has no effect.

Syntax:	`SPCLR`	
Use with:	ICS08GPW only	
Example:	`SPCLR`	Clear input and output buffers for SPI simulation.

SPDI — Enter SPI Inputs

The SPDI command allows the user to enter input data into the SPI. If a data parameter is given, the value is placed into the next slot in the circular input buffer. If no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SPI. The maximum number of input values is 256 bytes.

The input characters are shifted in after the proper number of cycles have elapsed for each character's reception.

If the ICS08 circuit board is connected, SPI inputs and outputs are directed from the board, so this command has no effect.

Syntax: `SPDI [<n>]`

Where:

`<n>`	The value to be entered into the next location in the input buffer

Use with: ICS08GPW only

Examples:

`SPDI $55`	Set the next input value to the SPI to $55.
`SPDI`	Pull up the data window with all the input values.

View SPI Outputs SPDO

The SPDO command displays the output circular buffer from the SPI. A window is opened that shows all the data that the SPI has shifted out. An arrow is used to point to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

If the ICS08 circuit board is connected, SPI inputs and outputs are directed from the board, so this command has no effect.

Syntax: `SPDO`

Where:

`<n>`	The number of cycles for the period of the input clock

Use with: ICS08GPW only

Example: `SPDO` View data from the output buffer for SPI simulation.

SPFREQ Set Input Clock for SPI Slave

The SPFREQ command lets the user set the frequency of the SPI slave input clock. If the SPI is configured for slave mode, this command allows the user to enter the number of cycles (`n`) that the period of the input clock will be. If no value is given, a pop-up window will appear and the user will be prompted for a value. If this command is not used, then clocking is assumed to be set by the SPI control register.

If the ICS08 circuit board is connected, SPI inputs and outputs are directed from the board, so this command has no effect.

Syntax: `SPFREQ [<n>]`

Where:

`<n>`	The number of cycles for the period of the input clock

Use with: ICS08GPW only

Example: `SPFREQ 8` Set the period of the input slave clock to eight cycles (total shift = 8*8 = 64 cycles).

Execute Source Step(s) SS

The SS command steps through a specified number of source code instructions, beginning at the current program counter (PC) address value, then halts. All windows are refreshed as each instruction is executed. This makes the SS command useful for high level language compilers (such as C) so that the user can step through compiler source code instead of assembly instructions.

If the number argument is omitted, one source instruction is executed. If the SS command is entered with an *n* value, the command steps through `n` source instructions.

Syntax: `SS [<n>]`

Where:

`<n>`	Number of instructions to step through

Use with: ICS08GPW and ICD08SW

Examples:

`SS`	Step through the instruction at the PC address value.
`SS 8`	Step through eight instructions, starting at the current PC address value.

ST or STEP or T — Execute Single Step

The identical ST, STEP, and T commands step through a specified number of assembly instructions, beginning at the current program counter (PC) address, then halt. All windows are refreshed as each instruction is executed. If the number argument is omitted, one instruction is executed. If the command is entered with a parameter value, the command steps through that many instructions.

Syntax: `STEP [<n>]`

Where:

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

Use with: CS08GPW and ICD08SW

Examples:

`STEP`	Execute the assembly instruction at the PC address value.
`ST 2`	Execute two assembly instructions, starting at the PC address value.

Show Stack Window STACK

The STACK command opens the HC08 Stack window, which shows the stack pointer (SP) value, data stored on the stack, and results of an RTS or RTI instruction.

Syntax:	`STACK`	
Use with:	CS08GPW only	
Example:	`STACK`	Open the stack window.

STATUS Show Registers

The STATUS command displays the contents of the CPU registers in the Status window. The STATUS command is identical to the REG command.

This command is useful for saving the CPU state to a logfile.

Syntax: `STATUS`

Use with: ICS08GPW and ICD08SW

Example:	`STATUS`	Display the contents of the CPU registers.

Step Forever STEPFOR

The STEPFOR command continuously executes instructions, one at a time, beginning at the current program counter (PC) address. Execution continues until an error condition occurs, until it reaches a breakpoint, or until a key or the STOP button on the ICS08GPW toolbar is pressed. All windows are refreshed as each instruction is executed.

Syntax: `STEPFOR`

Use with: ICS08GPW and ICD08SW

Example: `STEPFOR` Step through instructions continuously.

STEPTIL — Step Until 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 continues to the specified address or until a key or the STOP button on the debugger's toolbar is pressed, or it reaches a breakpoint, or until an error occurs.

Syntax: `STEPTIL <address>`

Where:

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

Use with: ICS08GPW and ICD08SW

Example:

`STEPTIL 0200`	Execute instructions continuously until PC value is 0200.

Add Symbol 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 commands and replaces all addresses in the Code (when displaying disassembly) and Variables windows.

Syntax:

```
SYMBOL [<label> <value>]
```

Where:

`<label>`	The ASCII-character string label of the new symbol
`<value>`	The value of the new symbol (label)

Use with: ICS08GPW and ICD08SW

Examples:

`SYMBOL`	Show the current user-defined symbols.
`SYMBOL timer_control $08`	Define new symbol timer_control with value $08. Subsequently, to modify the value of timer_control, enter the command: MM timer_control new_value
`MM timer_control $33`	Write $33 to address $08.

TRACE — Enable/Disable Tracing

The TRACE command enables or disables instruction captures. When tracing is enabled, the debugger records instructions in a 1024-element circular buffer. Note that tracing slows execution somewhat.

The debugger disassembles captured information when buffer contents are viewed through the Trace window. To view tracing results, use the SHOWTRACE command. If tracing is not enabled or if a trace slot is empty, the Trace window will display the message `No Trace Available`. To clear the Trace window, toggle tracing OFF and then ON using the TRACE command.

Syntax:	`TRACE`	
Use with:	ICS08GPW only	
Example:	`TRACE`	Enable (or disable) instruction tracing.

Upload S Record to Screen UPLOAD_SREC

The UPLOAD_SREC command uploads the contents of the specified memory block (range), in .S19 object file format, displaying the contents in the status window. If a logfile is opened, UPLOAD_SREC puts the information into the logfile as well.

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.*

This command is particularly useful in ICD08SW, where data from the real MCU can be captured to a logfile.

Syntax: `UPLOAD_SREC <startrange> <endrange>`

Where:

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

Use with: ICS08GPW and ICD08SW

Example:

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

V — Set or Clear V Bit (CCR)

The V command sets (1) or clears (0) the V bit in the condition code register (CCR).

NOTE: *The CCR bit designators are at the lower right of the CPU window. The CCR pattern is 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: `V [<value>]`

Where:

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

Use with: ICS08GPW and ICD08SW

Examples:

`V 0`	Clear the CCR V bit.
`V 1`	Set the CCR V bit.

Display Variable VAR

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.

In ASCII displays of variables, control characters or other non-printing characters appear as periods (.). Byte, word, long, or string variants determine the display format:

Byte (.B) — Hexadecimal and binary (the default)

Word (.W) — Hexadecimal and decimal

Long (.L) — Hexadecimal and decimal

String (.S) — ASCII characters

The optional <*n*> parameter specifies the number of string characters to be displayed; the default value is 1. 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

Use with: ICS08GPW and ICD08SW

Examples:

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

VERIFY — Verify S-Record File

The VERIFY command compares the contents of program memory with an S-record file. The name of the file is prompted.

The comparison stops at the first memory location that differs from the file.

Syntax: `VERIFY`

Use with: ICD08SW only

Examples:

`LOADALL`	Test.s19
`VERIFY`	Displays the message `Verifying....verified`

Display Software Version — VERSION or VER

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

Syntax: `VERSION`

Use with: ICS08GPW and ICD08SW

Examples:	`VERSION`	Display version and date of the software.
	`VER`	Display version and date of the software.

WAIT — Wait for *n* Cycles

The WAIT command delays simulator command execution by the specified number of cycles. This command is used in MACRO files to control when inputs come into the simulator. If a WAIT command is encountered, control is passed back to the keyboard. Then the macro file execution waits for a command to be entered, such as GO or STEP, which starts MCU execution once again. As soon as the number of cycles that pass is equal to the `<n>` value of the WAIT command, the simulator resumes executing commands of the macro file until another WAIT is encountered or the two mentioned conditions happens again.

Syntax: `WAIT <n>`

Where:

`<n>`	The hexadecimal number of cycles to wait

Use with: ICS08GPW only

Example: `WAIT A` Delay command execution for 10 MCU cycles.

Display Symbol Value — WHEREIS

The WHEREIS command displays the value of the specified symbol. Symbol names are defined through source code or the SYMBOL command. Alternatively, this command returns the symbol at a specified address.

Syntax: `WHEREIS <symbol> | <address>`

Where:

`<symbol>`	A symbol listed in the symbol table
`<address>`	Address for which a symbol is defined

Use with: ICS08GPW and ICD08SW

Examples:

`WHEREIS START`	Display the symbol START and its value.
`WHEREIS 0300`	Display the value 0300 and its symbol name if any.

X or XREG — Set X Register Value

The X command sets the index (X) register to the specified value. The value entered with the command is displayed in the CPU window. The X command is identical to the XREG command.

Syntax: `X <value>`

Where:

`<value>`	The new value for the X register

Use with: ICS08GPW and ICD08SW

Examples:

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

Set/Clear Zero Bit Z

The Z command sets or clears the Z bit in the condition code register (CCR).

NOTE: *The CCR bit designators are in the lower portion of the CPU window. The CCR pattern is 111HINZC (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: `Z 0|1`

Use with: ICS08GPW and ICD08SW

Examples:

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

Section 11. Example Project

11.1 Contents

11.2 Introduction 323

11.3 Setting Up a Sample Project 323
11.3.1 Set Up the Environment 324
11.3.2 Create the Source Files 325
11.3.3 Assemble the Project 326

11.2 Introduction

This section provides information to guide a user through a first-time use of the ICS08GPW software and through a typical setup of the WinIDE.

11.3 Setting Up a Sample Project

To demonstrate how source code to be assembled is handled using the ICS08GPW simulator, WinIDE editor, and CASM08W assembler software as an integrated development environment, consider as an example the following typical project.

NOTE: *The sample files referred to here are referenced for illustration purposes only and are not provided with the software. Create personal *.ASM files for projects using the ICS08GPW software components. For information about using files created by other assemblers, see* ***5.5.5 Files from Other Assemblers****.*

11.3.1 Set Up the Environment

To begin the project, start the WinIDE editor and establish the desktop and environment settings for the project:

1. Start the WinIDE editor by selecting the icon from the Windows 95 (or later version) **Start** menu.
2. In the WinIDE editor, choose the **Setup Environment** option from the **File** menu to open the **Environment Settings** dialog.
3. Enter environment options for the modules of the WinIDE development environment, represented by the **General Editor**, **General Environment**, **EXE1**, **EXE2**, **EXE3**, and **EXE4 Assembler/Compiler** tabs. For the example project:
   a. In the **General Environment** tab, choose the preferred environment options. In the **%FILE% Parameter passed to external program is** text box, enter the path and filename (MAIN.ASM).
   b. In the **General Editor** tab, choose the editing options preferred.
   c. In the **EXE1** tab, make sure the **EXE Path** text box points to the ICS08GPW.EXE path and filename, and that the **Options** text box indicates the proper communications port (if the pod is to be used).
   d. In the **Assembler/Compiler** tab, make sure these options are selected:
      i. The **EXE Path** text box indicates the path and filename for the CASM08W.EXE.
      ii. The **Type** text box specifies the P&E CASM08W assembler.
      iii. To view the CASM08W window during assembly, check the **Show Assembler Progress** option in the **Assembly Preferences** section of the tab.
4. Press the OK button to save the settings made in the **Environment Settings** dialog tabs and close the dialog.

The environment is now set for the project. To save it for later use:

1. In the WinIDE, select the **Save Project As** option from the **Environment** menu.
2. In the dialog box, enter a path and a descriptive project filename with the .PPF extension. Place the project file in the directory where the source files will be located.

11.3.2 Create the Source Files

Create new or edit existing source code files using the WinIDE editor:

1. From the **File** menu, choose the **New File** option to create a blank source window in which to enter source code (or open an existing file using the **Open File** option). All the source code files in the WinIDE editor can be worked on individually.
2. After creating the new file or editing the existing file, from the **File** menu, choose the **Save File** option to assign a path and filename to the source file (or choose the **Save File As** option to assign a new path and filename to an existing file).
3. Create all the source code files required for the project.

The example project consists of 12 source code files created in the WinIDE editor. The files are then assembled into *.ASM files using WinIDE ASSEMBLE/COMPILE toolbar button. The 12 files are then listed in a separate file MAIN.ASM.

The MAIN.ASM file consists of `$INCLUDE` functions, each followed by the filename for the source code file, followed by an optional comment describing the function of the code in that file. Using the `$INCLUDE` function in a main file lets the user organize source code logically into a number of small files, ultimately making it easier to develop, manage, and work with the source code. For more information about using the `$INCLUDE` function, see **5.7.4 INCLUDE**.

The example MAIN.ASM file:

```
****************************************************
Include files
****************************************************
$include "equates.asm"
$include "init.asm"
$include "charge.asm"
$include "dcharge.asm"
$include "options.asm"
$include "misc.asm"
$include "readv.asm"
$include "isr.asm"
$include "display.asm"
$include "eeprom.asm"
$include "text.asm"
$include "vectors.asm"
```

11.3.3 Assemble the Project

The project is ready for assembly. In the WinIDE, follow these steps:

1. With the MAIN.ASM file in the active window, press the ASSEMBLE/COMPILE FILE button (third button from the left) on the WinIDE toolbar to start the assembler from the WinIDE editor. Alternately, press the F4 hotkey.
2. If the assembler encounters errors during assembly, the assembler stops and the first error is displayed highlighted in red in the source file. To correct the errors, click on the DEBUGGER (EXE1) toolbar button (left-most button) on the WinIDE toolbar to open or move to the ICS08GPW simulator to debug the source code. When you have finished debugging the code in the ICS08GPW simulator, return to the WinIDE editor by clicking the BACK TO EDITOR button (the left-most button) in the ICS08GPW toolbar.
3. Continue assembling, debugging, and editing the source files until the assembly completes successfully.
4. Based on the **Output Control** options selected in the **Assembler/Compiler** tab of the **Environment Settings** dialog, the

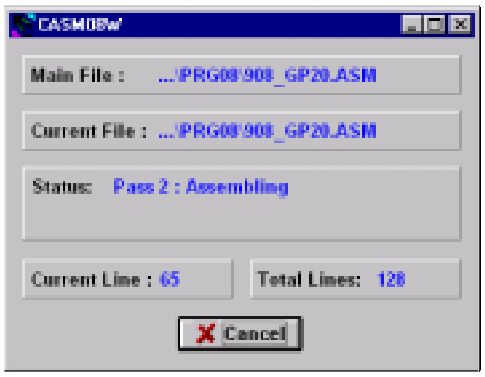
assembler creates additional output files with the filename of the main file and an extension which indicates the file type. The S19 and MAP files are required; the LST file is optional.

a. MAIN.S19 — Motorola S-record (S19) object code file that can be downloaded into the simulator

b. MAIN.MAP — Map file containing information necessary for source level debugging

c. MAIN.LST — Listing file.

d. The CASM08W window displays during assembly, showing the files and progress of the assembler in the **Status** area. When assembly completes successfully, the assembler window appears like the one shown in **Figure 11-1**.



Main File : ...\PRG08\908_GP20.ASM
Current File : ...\PRG08\908_GP20.ASM
Status: Pass 2 : Assembling
Current Line : 65
Total Lines: 128

Figure 11-1. CASM08W Window

Section 12. Using the MON08 Interface

12.1 Contents

12.2 Introduction. 329

12.3 Header Placement and Layout 329

12.4 Connecting to the In-Circuit Simulator. 331

12.5 Disabling the Target-System Interface 332

12.2 Introduction

The MON08 debugging interface may be used to debug and program a target system's MCU directly. The target must be connected to the ICS08GP20 board's MON08 interface connector. This chapter explains how to connect to the MON08 interface on the target board.

12.3 Header Placement and Layout

Two headers must be placed on the target board:

- P1 — 1-pin header such as Berg Electronics part number 68001-601
- P2 — 16-pin header such as Berg Electronics part number 67997-616

Table 12-1 and **Table 12-2** show the target-system interconnections for P1 and P2. **Figure 12-1** shows the pin layouts for P1 and P2. Additional information about the connections on the ICS08GP20 board can be found in **Appendix B. Technical Reference and Troubleshooting**.

Table 12-1. MON08 Target System Connector P1

Pin No.	ICS08GP20 Label	Direction	Target System Connection
1	$\overline{\text{RST}}$	Bidirectional	Connect to MCU $\overline{\text{RST}}$ pin and P2 pin 4. No other target-system logic should be tied to this signal. It will swing from 0 to +8.5 Vdc.

Table 12-2. MON08 Target System Connector P2

Pin No.	ICS08GP20 Label	Direction	Target System Connection
1	$\overline{\text{RST}}$_OUT	Out to target	Connect to logic that is to receive the $\overline{\text{RST}}$ signal.
2	GND	Ground	Connect to ground (V_{SS}).
3	$\overline{\text{RST}}$_IN	In from target	Connect to all logic that generates resets.
4	$\overline{\text{RST}}$	Bidirectional	Connect to MCU $\overline{\text{RS}}$ pin and P1 pin 1. No other target-system logic should be tied to this signal. It will swing from 0 to +8.5 Vdc.
5	TGT_$\overline{\text{IRQ}}$	In from target	Connect to logic that generates interrupts.
6	$\overline{\text{IRQ}}$	Out to target	Connect to MCU IRQ pin. No other target-system logic should be tied to this signal. It will swing from 0 to +8.5 Vdc.
7	TGT_PTA0	Bidirectional	Connect to user circuit that normally would be connected to PTA0 on the MCU. This circuit will not be connected to the MCU when the in-circuit simulator is being used.
8	PTA0	Bidirectional	Connect to MCU PTA0 pin. No other target-system logic should be tied to this signal. Host I/O present on this pin.
9	TGT_PTA7	Bidirectional	Connect to user circuit that normally would be connected to PTA7 on the MCU.
10	PTA7	Bidirectional	Connect to MCU PTA7 pin. No other target-system logic should be tied to this signal. Grounded during reset and for 256 cycles after reset.
11	TGT_PTC0	Bidirectional	Connect to user circuit that normally would be connected to PTC0 on the MCU.
12	PTC0	Bidirectional	Connect to MCU PTC0 pin. No other target-system logic should be tied to this signal. Held at +5 Vdc during reset.
13	TGT_PTC1	Bidirectional	Connect to user circuit that normally would be connected to PTC1 on the MCU.
14	PTC1	Bidirectional	Connect to MCU PTC1 pin. No other target-system logic should be tied to this signal. Grounded during reset.

Table 12-2. MON08 Target System Connector P2 (Continued)

Pin No.	ICS08GP20 Label	Direction	Target System Connection
15	TGT_PTC3	Bidirectional	Connect to user circuit that normally would be connected to PTC3 on the MCU.
16	PTC3	Bidirectional	Connect to MCU PTC3 pin. No other target-system logic should be tied to this signal. Grounded during reset.

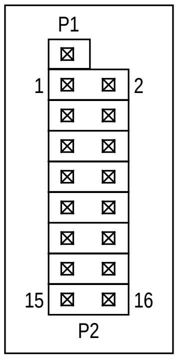


Figure 12-1. MON08 Target-System Connector Layout

12.4 Connecting to the In-Circuit Simulator

Using the 16-pin cable provided with the M68ICS08GP20 kit, connect one end of the cable to the ICS08GP20 board at J6. Connect the other end to connector P2 on the target-system board. The pin-1 indicators on each cable end must correspond to the pin-1 indicators on the headers. P1 is not used when connecting to the ICS08GP20 board.

12.5 Disabling the Target-System Interface

To use the target system in a stand-alone fashion (without the M68ICS08GP20 hardware pod connected), it is necessary to jumper the pins on the target board's connectors, as shown in **Figure 12-2**. This reconnects the target MCU to the appropriate circuits on the target system.

For production boards, a further enhancement of this scheme would be to include cutable traces between the pins of P1 and P2, as shown in **Figure 12-2**. The traces may be cut when debugging is necessary. To return the board to stand-alone use, jumpers may be installed as shown.

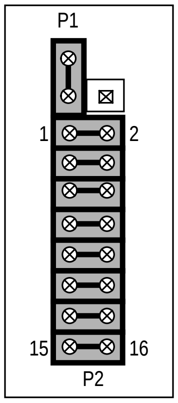


Figure 12-2. Target System Stand-Alone Connection

Appendix A. S-Record Information

A.1 Contents

A.2 Introduction. 333

A.3 S-Record Content 334

A.4 S-Record Types. 335

A.5 S-Record Creation. 336

A.6 S-Record Example 336
A.6.1 S0 Header Record 337
A.6.2 First S1 Record. 338
A.6.3 S9 Termination Record 339
A.6.4 ASCII Characters 339

A.2 Introduction

The Motorola S-record format was devised for the purpose of encoding programs or data files in a printable format for transport between computer platforms. The format also provides for editing of the S records and monitoring the cross-platform transfer process.

A.3 S-Record Content

Each S record is a character string composed of several fields which identify:

- Record type
- Record length
- Memory address
- Code/data
- Checksum

Each byte of binary data is encoded in the S record as a 2-character hexadecimal number:

- The first character represents the high-order four bits of the byte.
- The second character represents the low-order four bits of the byte.

The five fields that comprise an S record are shown in the **Table A-1**.

Table A-1. S-Record Fields

TYPE	RECORD LENGTH	ADDRESS	CODE/DATA	CHECKSUM

The S-record fields are described in **Table A-2**.

Each record may be terminated with a CR/LF/NULL. Additionally, an S record may have an initial field to accommodate other data such as line number generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

Table A-2. S-Record Field Contents

Field	Printable Characters	Contents
Type	2	S-record type — S0, S1, etc.
Record length	2	Character pair count in the record, excluding the type and record length
Address	4, 6, or 8	2-, 3-, or 4-byte address at which the data field is to be loaded into memory
Code/Data	0-2n	From 0 to *n* bytes of executable code, memory loadable data, or descriptive information. For compatibility with teletypewriter, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S record)
Checksum	2	Least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields

A.4 S-Record Types

Eight types of S records have been defined to accommodate the several needs of the encoding, transport, and decoding functions. The various Motorola upload, download, and other record transport control programs, as well as cross assemblers, linkers, and other file-creating or debugging programs, utilize only those S records which serve the purpose of the program. For specific information on which S records are supported by a particular program, consult the user manual for the program.

NOTE: *The ICS08GPW supports only the S0, S1, and S9 record types. All data before the S1 record is ignored. Thereafter, all records must be S1 type until the S9 record, which terminates data transfer.*

An S-record format may contain the record types in **Table A-3**.

Only one termination record is used for each block of S records. Normally, only one header record is used, although it is possible for multiple header records to occur.

Table A-3. S-Record Types

Record Type	Description
S0	Header record for each block of S records. The code/data field may contain any descriptive information identifying the following block of S records. The address field is normally zeroes.
S1	Code/data record and the 2-byte address at which the code/data is to reside
S2–S8	Not applicable to ICS08GPW
S9	Termination record for a block of S1 records. Address field may optionally contain the 2-byte address of the instruction to which control is to be passed. If not specified, the first interplant specification encountered in the input will be used. There is no code/data field.

A.5 S-Record Creation

S-record format programs may be produced by dump utilities, debuggers, cross assemblers, or cross linkers. Several programs are available for downloading a file in the S-record format from a host system to an 8- or 16-bit microprocessor-based system.

A.6 S-Record Example

A typical S-record format, as printed or displayed, is shown in this example:

Example:

```
S00600004844521B
S1130000285F245F2212226A00042429008237C2A
S11300100002000800082529001853812341001813
S113002041E900084#42234300182342000824A952
S107003000144ED492
S9030000FC
```

In the example, the format consists of:

- An S0 header
- Four S1 code/data records
- An S9 termination record

A.6.1 S0 Header Record

The S0 header record is described in **Table A-4**.

Table A-4. S0 Header Record

Field	S-Record Entry	Description
Type	S0	S-record type S0, indicating a header record
Record length	06	Hexadecimal 06 (decimal 6), indicating six character pairs (or ASCII bytes) follow
Address	0000	4-character, 2-byte address field, zeroes
Code/Data	48 44 52	Descriptive information identified these S1 records: ASCII H, D, and R — “HDR”
Checksum	18	Checksum of S0 record

A.6.2 First S1 Record

The first S1 record is described in **Table A-4**.

Table A-5. S1 Header Record

Field	S-Record Entry			Description	
Type	S1			S-record type S1, indicating a code/data record to be loaded/verified at a 2-byte address	
Record length	13			Hexadecimal 13 (decimal 19), indicating 19 character pairs, representing 19 bytes of binary data, follow.	
Address	0000			4-character, 2-byte address field; hexadecimal address 0000, indicates location where the following data is to be loaded	
Code/Data	**Opcode**			**Instruction**	
	28	5F		BHCC	$f0161
	24	5F		BCC	$0163
	22	12		BHI	$0118
	22	6A	2	BHI	$0172
	00	04	4	BRSET	0, $04, $012F
	29	00		BHCS	$010D
	08	23	7	BRSET	4, $23, $018C
Checksum	2A			Checksum of the first S1 record	

The 16 character pairs shown in the code/data field of **Table A-5** are the ASCII bytes of the actual program.

The second and third S1 code/data records each also contain $13 (19) character pairs and are ended with checksum 13 and 52, respectively. The fourth S code/data record contains 07 character pairs and has a checksum of 92.

A.6.3 S9 Termination Record

The S9 termination record is described in **Table A-6**.

Table A-6. S-9 Header Record

Field	S-Record Entry	Description
Type	S9	S-record type S9, indicating a termination record
Record length	03	Hexadecimal 04, indicating three character pairs (three bytes) follow
Address	0000	4-character, 2-byte address field, zeroes
Code/Data		No code/data in an S9 record
Checksum	FC	Checksum of S9 record

A.6.4 ASCII Characters

Each printable ASCII character in an S record is encoded in binary. **Table A-6** gives an example of encoding for the S1 record. The binary data is transmitted during a download of an S-record from a host system to a 9- or 16-bit microprocessor-based system.

Appendix B. Technical Reference and Troubleshooting

B.1 Contents

B.2 Introduction. 341

B.3 Functional Description 342
B.3.1 ICS08GP20 Board 342
B.3.2 SPGMR08 Serial Programmer Base Unit 344

B.4 Troubleshooting the Quick Start. 344

B.5 Troubleshooting MON08 Mode 351

B.6 Connector Pin Assignments 355

B.7 Target-Cable Pin Assignments 364

B.8 Parts List 367

B.9 Board Layout and Schematic Diagram. 369

B.2 Introduction

This appendix provides technical support information for the M68ICS08GP in-circuit simulator kit, including:

- Functional description of the kit
- Troubleshooting the quick-start procedure
- Troubleshooting MON08 mode
- Connector and cable pin assignments
- Schematic diagrams
- Parts list
- Board layout diagram

ESD CAUTION: *Ordinary amounts of static electricity from clothing or the work environment can damage or degrade electronic devices and equipment. For example, the electronic components installed on printed circuit boards are extremely sensitive to electrostatic discharge (ESD). Wear a grounding wrist strap whenever handling any printed circuit board. This strap provides a conductive path for safely discharging static electricity to ground.*

B.3 Functional Description

The M68ICS08GP hardware pod consists of two components:

- ICS08GP20 board
- SPGMR08 serial programmer base unit

B.3.1 ICS08GP20 Board

The core component of the board is the MCU (either the MC68HC908GP20 or MC68HC908GP32). This MCU resides either on the ICS08GP20 board or on a target system.

When the MCU resides on the board, the pod may be used as an in-circuit emulator or simulator for the MC68HC908GP20. For this configuration, a target cable is run from the pod to the target system. The ICS08GP20 supports two kinds of target cables:

- A 40-pin ribbon cable (Motorola part number 01-RE91138W01), terminating in a 40-pin male DIP (dual in-line package) header
- A flexible target head adapter cable (Motorola part number M68CBL05C), terminating in connectors for one of two target head adapters. Two target head adapters (THAs) are available for use with the flex cable:
  - For a 40-pin DIP-package MCU on the target system, use Motorola THA part number M68TC08GP20P40.
  - For a 44-pin QFP-package (quad flat pack) MCU on the target system, use Motorola THA part number M68TC08GP20FB44.

Using a target cable is optional, and the board may be utilized with flying leads to other circuits. The MCU can be either the 44-pin QFP version or the 40-pin DIP version. On the ICS08GP20 board, socket XU1 supports the 40-pin DIP package, and socket XU3 supports the 44-pin QFP package.

When the MCU resides on a target system, the ICS08GP20 board can communicate with the MCU over a 16-pin MON08 cable (Motorola part number 01-RE91008W01). Either version of the MCU is supported when using the MON08 cable.

When using the ICS08GPW simulation software, the MCU provides the required input/output information that lets the host computer simulate code, performing all functions except for maintaining port values. The internal FLASH memory on the device is downloaded with a program that generates the appropriate port values. The ICS08GPW software on the host computer lets the host computer become a simulator. When the ICS requires port data, the computer requests the data through the host's serial connection to the core MCU. The core MCU responds by sending the data to the host via the serial connection. It is this arrangement that allows a real-world interface for the in-circuit simulator. The clock runs the MCU at a 4.9512-MHz external clock rate.

NOTE: *The simulation speed will be slower than this rate, because the host computer is the simulator.*

When using the ICD08SW debugging software, the user’s code can be run directly out of the MCU’s internal FLASH at real-time speeds.

NOTE: *The ICS08GP20’s emulation of the on-board MCU is limited. Port A bit 0 (PTA0) is used for host-to-MCU communication. The port bit is not available for connection to a target system. Setting DDRA bit 0 to 1 will stop communications with the simulation or debugger software and will require a system reset to regain communication with the MCU. Port bits PTA7, PTC0, PTC1, and PTC3 are temporarily disconnected from the target system during reset. Emulation of the MC68HC908GP’s $\overline{RST}$ signal is also limited in that the signal is not a bi-directional, open-drain signal. It is emulated as either an input or an output (determined by jumper header W2) when using the target connectors, or as two pins (one input and one output) when using the MON08 cable.*

When using the PROG08SW programming software, the MCU's FLASH memory can be programmed. Socket XU3 supports the 44-pin QFP version of the MC68HC908GP20FB or MC68HC908GP32FB, and socket XU1 supports the 40-pin DIP version of the MC68HC908GP20P or MC68HC908GP32P. Only one part may be programmed at a time. The pod also supports in-circuit programming of either version of the part through the MON08 cable.

B.3.2 SPGMR08 Serial Programmer Base Unit

The SPGMR08 provides +5 Vdc power, +8.5 Vdc power for the V_{TST} voltage required to enter monitor mode, a 4.9152-MHz clock signal, and host PC RS-232 level translation for the ICS08GP20 board. This is the same SPGMR08 base unit that is used with numerous programming adapters for other Motorola HC08 MCUs.

The socket power LED on some older SPGMR08s lights dimly when used with the ICS08GP20. This is normal and does not affect the functionality of the assembled pod.

B.4 Troubleshooting the Quick Start

The quick-start installation procedure in **1.8 Quick Start Instructions** describes how to prepare the pod for use in the modes where the MCU is installed on the ICS08GP20 board. These modes include:

- Using the pod as an in-circuit simulator/emulator with a target cable
- Using the pod as a programmer
- Using the pod as a stand-alone system without a target board

If difficulties are experienced when quick-starting the kit using the procedure outlined in **1.8 Quick Start Instructions**, follow these steps:

1. Do not use the MON08 cable to a target system in these modes. The MON08 cable connection is to be used only when the MCU is on the target system. Troubleshooting information for the MON08 modes may be found in **B.5 Troubleshooting MON08 Mode**.

2. Disconnect any target cables from the board. These troubleshooting steps assume that no target system connections are present.

3. Make sure that the MCU is installed correctly. Verify that only one MCU is installed in either XU1 or XU3. Insert the MC68HC908GP32FB QFP version of the part into the XU3 socket with the part pins down and with the orientation notch and pin 1 to the upper left. Alternately, insert the DIP version of the part (MC68HC908GP32P) into the XU1 socket with the part pins down and with the orientation notch to the top of the board.

4. Make sure the board is getting power:

   a. Check the power at the output of the adapter. First disconnect the pod from the power supply, then measure the power at the wall adapter's output connector to confirm that it produces 5 Vdc. The outer barrel of the connector is ground, and the inner sleeve is +5 Vdc. If there is no power at the connector, verify that the adapter is getting power from the AC power outlet.

   b. Check the power at the SPGMR. First remove the ICS08GP20 board from the SPGMR, then plug the adapter's output connector into the SPGMR. The system power LED should light. Check for 5 Vdc at the SPGMR's board socket P5 pin 7. This socket is the 10-pin SIP socket located nearest the LEDs. If the LED does not light, or if 5 Vdc is not present on P5 pin 7, check the fuse in the SPGMR. If more than 6.2 Vdc or reverse voltage is applied to the SPGMR, the fuse will blow.

   c. Check the pod system power with the ICS08GP20 board attached. Disconnect the SPGMR from the power supply and disconnect the SPGMR from the host PC. Configure the ICS08GP20 board to the factory defaults and attach the ICS08GP20 board to the SPGMR. Reconnect the power supply to the SPGMR. The system power LED should light. If the LED does not light, there may be a problem with the ICS08GP20 causing too much of a drain on the 5 Vdc supply. Check the system power at J7 pin 7. The J7 connector is the top connector that connects to the SPGMR near the LEDs. The pins of the J7 connector may be probed from the top side of the board. Using J7 pin 1 as the ground reference, check for +5 Vdc at J7 pin 7.

d. Check the ICS08GP20 board's VSW_OUT power with the host disconnected. With the ICS08GP20 board installed on the SPGMR, the SPGMR powered, and no host connection to the SPGMR, check for the following voltages on the J7 connector of the ICS08GP20 board, using J7 pin 1 as the ground reference:

   Approximately 8.5 Vdc at J7 pin 9

   Approximately 1.25 Vdc at J7 pin 6

   If these voltages are not present when the system power LED is lit, there may be a problem with the SPGMR's internal step-up power supply. There may also be a problem with the R1-R2 voltage divider on the ICS08GP20 board.

e. Check the ICS08GP20 board's VSW_OUT and V_{TST} power with the host connected. First, exit any ICS08GPW software that may be running on the host PC. Then disconnect power from the SPGMR. Ensure that the ICS08GP20 board is installed on the SPGMR and that it is configured for the factory default settings. Ensure that there is an MCU in either XU1 or XU3 and that it is inserted correctly. Connect the serial cable between the host PC and the SPGMR. Apply power to the pod. At this point, the system power LED should be lit, and the socket power LED should be off. If the socket power LED is on, there may be a problem with the host PC's serial port or the serial cable. (See step 5 for communications problems.) If the socket power LED is off, start the ICS05GPW simulator software as described in sections 3 and 4 of the quick-start instructions while watching the socket power LED.

   If the socket power LED does not light at all, there may be a problem with the host PC communicating with the board. Refer to step 5.

   If the socket power LED flickers a few times and then goes out, the host PC is able to control the power to the ICS08GP20 board but communications may still not be established with the MCU. As the flickering of the socket power LED indicates, the host PC is applying and removing power to the ICS08GP20 board during this period. Use an oscilloscope to view the voltages on J7 as the software tries to establish communication with the MCU. The user will need to restart or retry the ICS08GPW software while looking at the signals. Using J7 pin 1 as the ground reference, check for a signal that varies between 0 and +5 Vdc at J7 pin 10 (board V_{CC}) and J7 pin 4 (SP_$\overline{\text{RESET}}$). If these voltages are present, the power is good, but communication problems should be investigated as described in step 5.

If the socket power LED comes on and stays on, communication was probably established with the MCU. Check for the following voltages at J7, using J7 pin 1 as the ground reference:

i. Approximately 8.5 Vdc at J7 pin 3 and J7 pin 9
ii. Approximately 1.25 Vdc at J7 pin 6
iii. 5 Vdc at J7 pin 4 and J7 pin 10

If these voltages are present, the power is good, and the problem lies elsewhere.

5. Make sure that the host PC can communicate with the MCU:

   a. The MCU's PTA0 pin is used for host communications. DDRA bit 0 should never be set to 1 as this interrupts monitor-mode communications. The target connector PTA0 pins (J2 pin 15, J4 pin 10, and J6 pin 7) are never connected to the MCU's PTA0 pin; they are wired only for probing purposes.

   b. Make sure that the serial cable is correctly attached to the pod and to the correct serial port on the host computer.

   c. Make sure that the cable is a straight-through cable supporting all nine pins of the serial port connection.

   d. Make sure that no hardware security key or other devices are attached to the serial port or cable.

   e. Make sure that the host PC supports the minimum speed requirements of the ICS08GPW software.

   f. Make sure to use the correct security code to access the MCU. If the security bytes have been programmed previously, the part will not unlock and enter monitor mode unless the correct security code is sent to the MCU.

g. Check for data at the pod end of the serial cable. Pin 3 of this connector carries RS-232 data into the pod; pin 2 carries RS-232 data out of the pod. Pin 4 controls the socket power. Pin 5 is ground. While the ICS08GPW software is trying to establish communications, pins 3 and 4 should both toggle between +10 Vdc and –10 Vdc (or +12 Vdc and –12 Vdc). If these signals are not seen at the cable end, the problem is on the PC and cable side of the system. When connected to the pod, a +10 Vdc signal on pin 4 should activate the socket power and socket power LED.

h. Make sure the serial data is getting to the MCU's PTA0 pin. First, exit any ICS08GPW software that may be running on the host PC. Then disconnect power from the SPGMR. Ensure that the ICS08GP20 board is installed on the SPGMR and that it is configured for the factory default settings. Ensure that there is an MCU in either XU1 or XU3 and that it is inserted correctly. Connect the serial cable between the host PC and the SPGMR. Apply power to the pod. Start the ICS05GPW simulator software as described in sections 3 and 4 of the quick-start instructions. Probe the PTA0 pin (XU1 pin 33 or XU3 pin 32) for the serial data. Since the board power is turned off and on several times during the connecting phase, the data observed at the MCU's PTA0 pin is also affected.

6. Make sure that the MCU has a good clock source. Use an oscilloscope to check the OSC1 input at the MCU (XU1 pin 5 or XU3 pin 44). Set the oscilloscope to 0.1 µs per division. The oscillator should run when socket power is on. Approximately two divisions per cycle should be observed. This corresponds to a 4.9152-MHz signal; the frequency required for a 9600-baud communications rate. If the clock signal is not present, check to see that a jumper is installed on W1. This selects the SPGMR as the source of the OSC1 signal.

7. Make sure that the MCU can enter and remain in monitor mode. For this to happen, the following conditions must occur:

a. At the rising edge of $\overline{\text{RST}}$, $\overline{\text{IRQ}}$ must be at V_{TST} (8.5 Vdc). Using a dual-trace oscilloscope, trigger channel 1 on the rising edge of $\overline{\text{RST}}$ (XU1 pin 6 or XU3 pin 1) and read the $\overline{\text{IRQ}}$ pin (XU1 pin 14 or XU3 pin 11) with channel 2. Start the ICS08GPW software as described in **1.8 Quick Start Instructions** and verify that the signal is approximately 8.5 Vdc when $\overline{\text{RST}}$ rises. If $\overline{\text{IRQ}}$ is not at 8.5 Vdc, there may be a problem with the ICS08GP20 board's IRQ circuit. Check R3, Q2, U6, R6, and U5 for the proper signals to keep $\overline{\text{IRQ}}$ at 8.5 Vdc during the period where $\overline{\text{RST}}$ is low.

b. At the rising edge of $\overline{\text{RST}}$, PTA0, PTA7, PTC0, PTC1, and PTC3 must be held at logic values 1, 0, 1, 0, and 0 respectively. The logic levels are 5-volt CMOS logic levels. Using a dual-trace oscilloscope, trigger channel 1 on the rising edge of $\overline{\text{RST}}$ (XU1 pin 6 or XU3 pin 1), and read the corresponding MCU pin with channel 2. PTA0 (XU1 pin 33 or XU3 pin 32) is the serial data pin to and from the host PC and should be around 5 Vdc at the rising edge of $\overline{\text{RST}}$. PTA7 (XU1 pin 40 or XU3 pin 39), PTC0 (XU1 pin 7 or XU3 pin 2), PTC1 (XU1 pin 8 or XU3 pin 3), and PTC3 (XU1 pin 10 or XU3 pin 5) are controlled by analog switch U4 and should be approximately 0 V, 5 V, 0 V, and 0 V, respectively, at the rising edge of $\overline{\text{RST}}$. Port pins PTA7, PTC0, PTC1, and PTC3 are connected to the target connector pins after the rising edge of $\overline{\text{RST}}$ and are then available for target system connections. The MCU's PTA0 pin is never connected to the target pins, as it is used for host communication.

c. PTA7 must be held low for at least 24 bus cycles after $\overline{\text{RST}}$ goes high. The counter circuit comprised of U5, U6, and U7 keeps the PTA7 pin near 0 V for 256 OSC1 clock cycles. This corresponds to 64 bus cycles. Using a dual-trace oscilloscope, trigger channel 1 on the rising edge of $\overline{\text{RST}}$ (XU1 pin 6 or XU3 pin 1), and read PTA7 (XU1 pin 40 or XU3 pin 39) with channel 2. Verify that PTA7 is held low for at least 256 OSC1 clock cycles (about 52 μs at 4.9152 MHz). PTA7 is reconnected to the target connector pins when this delay expires.

d. Either $\overline{\text{RST}}$ or $\overline{\text{IRQ}}$ must remain at 8.5 Vdc to hold the MCU in monitor mode. The ICS08GP20 board has an interrupt lockout feature to keep $\overline{\text{IRQ}}$ at 8.5 Vdc when the $\overline{\text{RST}}$ or $\overline{\text{RST\_IN}}$ signal is asserted (low) and keep it at 8.5 Vdc for at least 256 OSC1 clock cycles after $\overline{\text{RST}}$ goes high. The TGT_$\overline{\text{IRQ}}$ signal is allowed to control the signal when $\overline{\text{RST}}$ is not asserted and the delay has expired.

8. Make sure that external circuitry does not interfere with the monitor-mode communications. When connecting external circuitry to the ICS08GP20 board, use only the target system connectors J2, J3, and J4. This ensures that the target system will not interfere with the communications and setup of the MCU's monitor mode by allowing the pod to disconnect some target system components during monitor mode entry.

9. When connecting to a target system, observe the setting of W2 (target RST direction). W2 is provided to allow the user to select whether the target system can reset the MCU on the pod (jumper between pins 2 and 3) or whether the target system receives a reset signal from the pod (jumper between pins 1 and 2). $\overline{\text{RST}}$ is *not* a bidirectional, open-drain signal at the target connectors. Removing the jumper leaves the $\overline{\text{RST}}$_IN signal pulled up to +5 V.

10. When connecting to a target system, observe the setting of W3 (target V_{DD} disconnect). W3 is provided to allow the user to select whether the pod powers the target system's MCU and external circuitry (W3 jumper on) or whether the target provides the power for its MCU and circuitry (W3 jumper off). The only target system V_{DD} supported is +5 Vdc in either case.

CAUTION: *Remove the W3 jumper when the target system is powered by a source other than the pod. Failure to remove the jumper in this case will cause the two power-supply outputs to be connected together, possibly causing large currents to flow over the target cable. If the pod is to provide power to the target system, ensure that the current drain on the target connector's V_{DD} pins is kept under 100 mA.*

B.5 Troubleshooting MON08 Mode

This section describes the troubleshooting steps for the instances where the MCU is installed on a target system and the pod is used to interact with the target system through the MON08 cable. These instances include in-circuit simulation/emulation and FLASH memory programming through the MON08 cable.

1. Disconnect the target system and make sure that the pod operates correctly when configured as described in the quick-start instructions (**1.8 Quick Start Instructions**). Refer to **B.4 Troubleshooting the Quick Start**.

2. If the quick start works, the pod should be functioning well enough to place the MCU on the target system into monitor mode.

3. Prepare the pod for use with the MON08 cable. Turn off the power to the target system. Exit the ICS08GP software. Remove the power plug from the SPGMR. *Remove any MCU from sockets XU1 and XU3*. Install a jumper on W1 so that the ICS08GP20's delay circuit has an oscillator input. Jumper selections on W2 and W3 have no effect when using the MON08 cable.

4. Connect the 16-pin cable from J6 on the pod to the target system's MON08 connector. Details on designing a MON08 connector for the target system are given in **Section 12. Using the MON08 Interface**. If cutable jumpers were used on the target board, the jumpers must be cut before using the MON08 cable.

5. The target system (including the MCU) must be *externally* powered. The target system's MCU V_{DD} must be +5 Vdc to communicate with the pod. Do not apply power to the target system at this time.

6. Exit any ICS08GPW software that may be running on the host PC. Connect the serial cable between the host PC and the SPGMR. Apply power to the pod by connecting the wall adapter's output jack to the SPGMR. At this point, the system power LED should be lit, and the socket power LED should be off. If the socket power LED is on, there may be a problem with the host PC's serial port or the serial cable. Refer to step 9 for information on host communications.

7. Apply power to the target system. At this point, the target MCU should be powered. Check for +5 Vdc at the MCU's V_{DD} pin. The pod should leave the target MCU in reset with approximately 0 Vdc at the MCU's $\overline{\text{RST}}$ pin. Verify this at the target MCU's $\overline{\text{RST}}$ pin and at J6 pin 4. If $\overline{\text{RST}}$ floats too high, the MCU may start up and begin executing code out of its FLASH memory. The pod should reset the MCU again in step 8 when the software is started.

8. Start the ICS05GPW simulator software as described in sections 3 and 4 of the quick-start instructions while watching the socket power LED.

   If the socket power LED does not light at all, there may be a problem with the host PC communicating with the pod. Continue with step 9.

   If the socket power LED flickers a few times and then goes out, the host PC is able to control the pod but communications may still not be being established with the MCU on the target system. As the flickering of the socket power LED indicates, the host PC is applying and removing power to the ICS08GP20 board during this period. Use an oscilloscope to view the voltages on J7 as the software tries to establish communication with the MCU. Restart or retry the ICS08GPW software while looking at the signals. Using J7 pin 1 as the ground reference, check for a signal that varies between 0 and +5 Vdc at J7 pin 10 (pod V_{CC}) and J7 pin 4 (SP_$\overline{\text{RESET}}$). If these voltages are present the pod power is good, but the MCU is not being placed in monitor mode. Continue with step 9.

   If the socket power LED comes on and stays on, communication is probably established with the MCU. Check for the following voltages at J7 on the ICS08GP20 board, using J7 pin 1 as the ground reference:

      Approximately 8.5 Vdc at J7 pin 3 and J7 pin 9

      Approximately 1.25 Vdc at J7 pin 6

      5 Vdc at J7 pin 4 and J7 pin 10

   If these voltages are present, the pod power is good. Continue with step 9.

9. Make sure the host PC can communicate with the MCU:

   a. The MCU's PTA0 pin is used for host communications. DDRA bit 0 should never be set to 1, as this interrupts monitor-mode communications. The MON08 pin TGT_PTA0 (J6 pin 7) is never connected to the MCU's PTA0 pin. It is wired to J2 pin 15, J4 pin 10, and J6 pin 7 for probing purposes. On the MON08 connector J6, pin 8 is wired to the MCU's PTA0 pin. Driving this signal with external logic on the target system will interrupt communications.

   b. Make sure that the MON08 cable is properly installed between the pod and the target system. Pin 1 of each connector on the cable must go to pin 1 of the headers on the pod and target system.

   c. Make sure that the serial cable is correctly attached to the pod and to the correct serial port on the host computer.

   d. Make sure that the cable is a straight-through cable supporting all nine pins of the serial-port connection.

   e. Make sure that no hardware security key or other device is attached to the serial port or cable.

   f. Make sure that the host PC supports the minimum speed requirements of the ICS08GPW software.

   g. Make sureto use the correct security code to access the MCU. If the security bytes are programmed already, the part will not unlock and enter monitor mode unless the correct security code is sent to the MCU.

   h. Make sure the serial data is getting to the MCU's PTA0 pin. Re-start the ICS05GPW simulator software as described in sections 3 and 4 of the quick-start instructions. Probe the PTA0 pin of the target MCU for the serial data. Since the board power is turned off and on several times during the connecting phase, the data observed at the MCU's PTA0 pin is also affected.

10. Make sure that the target MCU has a good clock source with a clock rate that gives a 9600-baud serial communications rate for monitor mode on the target system. Use an oscilloscope to check the OSC2 output at the MCU.

11. Make sure that the MCU can enter and remain in monitor mode. For this to happen, the following conditions must occur:

   a. At the rising edge of $\overline{\text{RST}}$, the target MCU's $\overline{\text{IRQ}}$ pin must be at V_{TST} (8.5 Vdc). Using a dual-trace oscilloscope, trigger channel 1 on the rising edge of the MCU's $\overline{\text{RST}}$ pin and read the $\overline{\text{IRQ}}$ pin with channel 2. Start the ICS08GPW software as described in **1.8 Quick Start Instructions** and verify that the $\overline{\text{IRQ}}$ signal is approximately 8.5 Vdc when $\overline{\text{RST}}$ rises.

   b. At the rising edge of $\overline{\text{RST}}$, PTA0, PTA7, PTC0, PTC1, and PTC3 on the MCU must be held at logic values 1, 0, 1, 0, and 0 respectively. The logic levels are 5-volt CMOS logic levels. Using a dual-trace oscilloscope, trigger channel 1 on the rising edge of $\overline{\text{RST}}$ and read the corresponding MCU pin with channel 2. PTA0 is the serial data pin to and from the host PC and should be around 5 Vdc at the rising edge of $\overline{\text{RST}}$. MCU pins PTA7, PTC0, PTC1, and PTC3 are controlled by analog switch U4 and should be approximately 0 V, 5 V, 0 V, and 0 V respectively at the rising edge of $\overline{\text{RST}}$. After the rising edge of $\overline{\text{RST}}$, the MCU pins PTA7, PTC0, PTC1, and PTC3 are connected (by the pod) to the MON08 connector pins TGT_PTA7, TGT_PTC0, TGT_PTC1, and TGT_PTC3. The MCU's PTA0 pin is never connected to the TGT_PTA0 pin, as it is used for host communication.

   c. The MCU's PTA7 pin must be held low for at least 24 bus cycles after the $\overline{\text{RST}}$ pin goes high. The counter circuit comprised of U5, U6, and U7 keeps the MCU's PTA7 pin near 0 V for 256 OSC1 clock cycles. This corresponds to 64 bus cycles. Using a dual-trace oscilloscope, trigger channel 1 on the rising edge of $\overline{\text{RST}}$ and read the MCU's PTA7 pin with channel 2. Verify that PTA7 is held low for at least 256 OSC1 clock cycles (about 52 µs at 4.9152 MHz). PTA7 is reconnected to the TGT_PTA7 pin and the target system circuitry when this delay expires.

   d. Either $\overline{\text{RST}}$ or $\overline{\text{IRQ}}$ must remain at 8.5 Vdc to hold the MCU in monitor mode. The ICS08GP20 board has an $\overline{\text{IRQ}}$ lockout feature to keep $\overline{\text{IRQ}}$ at 8.5 Vdc when the $\overline{\text{RST}}$ or $\overline{\text{RST\_IN}}$ signal is asserted (low) and keep it at 8.5 Vdc for at least 256 OSC1 clock cycles after $\overline{\text{RST}}$ goes high. The TGT_$\overline{\text{IRQ}}$ signal is allowed to control the $\overline{\text{IRQ}}$ signal when $\overline{\text{RST}}$ is not asserted and the delay has expired.

12. Make sure that the target circuitry does not interfere with the monitor-mode communications. When connecting target circuitry to the MCU, be sure to connect the circuits through the pod by connecting to the $\overline{\text{RST}}$_OUT, $\overline{\text{RST}}$_IN, TGT_$\overline{\text{IRQ}}$, TGT_PTA0, TGT_PTA7, TGT_PTC0, TGT_PTC1, and TGT_PTC3 pins of the MON08 connector. These signals will be connected by the pod to the corresponding pins of the MCU through the corresponding MON08 connector pins $\overline{\text{RST}}$, $\overline{\text{IRQ}}$, PTA7, PTC0, PTC1, and PTC3 after monitor mode is established. TGT_PTA0 is never connected to PTA0, as the PTA0 signal is being used for host communications.

B.6 Connector Pin Assignments

The tables in this section describe the pin assignments for the connectors on the ICS08GP20 board.

Table B-1. Target DIP Connector J2

Pin No.	Mnemonic	Schematic NET	Direction	Signal Description
1	PTA7/KBD7	TGT_PTA<7>	Bidirectional	Port A I/O / keyboard interrupt — bit 7
2	V_{DDA}	TGT_VDD	—	PLL analog power
3	PTA6/KBD6	PTA<6>	Bidirectional	Port A I/O / keyboard interrupt — bit 6
4	V_{SSA}	GND	—	PLL analog ground
5	PTA5/KBD5	PTA<5>	Bidirectional	Port A I/O / keyboard interrupt — bit 5
6	CGMXFC	CGMXFC	Analog	External filter capacitor
7	PTA4/KBD4	PTA<4>	Bidirectional	Port A I/O / keyboard interrupt — bit 4
8	OSC2	OSC2	Out	Crystal amplifier output
9	PTA3/KBD3	PTA<3>	Bidirectional	Port A I/O / keyboard interrupt — bit 3
10	OSC1	OSC1	In	Crystal amplifier input
11	PTA2/KBD2	PTA<2>	Bidirectional	Port A I/O / keyboard interrupt — bit 2
12	$\overline{RST}$	TGT_$\overline{RST}$	In or out	External reset
13	PTA1/KBD1	PTA<1>	Bidirectional	Port A I/O / keyboard interrupt — bit 1
14	PTC0	TGT_PTC<0>	Bidirectional	Port C I/O — bit 0
15	PTA0/KBD0	TGT_PTA<0>	Bidirectional	Port A I/O / keyboard interrupt — bit 0 Unavailable MCU connection
16	PTC1	TGT_PTC<1>	Bidirectional	Port C I/O — bit 1
17	V_{SSAD}/V_{REFL}	GND	—	ADC analog ground / voltage reference low
18	PTC2	PTC<2>	Bidirectional	Port C I/O — bit 2
19	V_{DDAD}/V_{REFH}	TGT_VDD	—	ADC analog power / voltage reference high
20	PTC3	TGT_PTC<3>	Bidirectional	Port C I/O — bit 3
21	PTB7/AD7	PTB<7>	Bidirectional	Port B I/O / ADC input — bit 7
22	PTC4	PTC<4>	Bidirectional	Port C I/O — bit 4
23	PTB6/AD6	PTB<6>	Bidirectional	Port B I/O / ADC input — bit 6
24	PTE0/TxD	PTE<0>	Bidirectional	Port E I/O — bit 0 / SCI transmit data
25	PTB5/AD5	PTB<5>	Bidirectional	Port B I/O / ADC input — bit 5
26	PTE1/TxD	PTE<1>	Bidirectional	Port E I/O — bit 1 / SCI receive data
27	PTB4/AD4	PTB<4>	Bidirectional	Port B I/O / ADC input — bit 4

Table B-1. Target DIP Connector J2 (Continued)

Pin No.	Mnemonic	Schematic NET	Direction	Signal Description
28	$\overline{\text{IRQ}}$	TGT_$\overline{\text{IRQ}}$	In	External interrupt
29	PTB3/AD3	PTB<3>	Bidirectional	Port B I/O / ADC input — bit 3
30	PTD0/$\overline{\text{SS}}$	PTD<0>	Bidirectional	Port D I/O — bit 0 / SPI slave select
31	PTB2/AD2	PTB<2>	Bidirectional	Port B I/O / ADC input — bit 2
32	PTD1/MISO	PTD<1>	Bidirectional	Port D I/O — bit 1 / SPI master in/slave out
33	PTB1/AD1	PTB<1>	Bidirectional	Port B I/O / ADC input — bit 1
34	PTD2/MOSI	PTD<2>	Bidirectional	Port D I/O — bit 2 / SPI master out/slave in
35	PTB0/AD0	PTB<0>	Bidirectional	Port B I/O / ADC input — bit 0
36	PTD3/SPSCK	PTD<3>	Bidirectional	Port D I/O — bit 3 / SPI serial clock
37	PTD5/T1CH1	PTD<5>	Bidirectional	Port D I/O — bit 5 / timer 1 channel 1
38	V_{SS}	GND	—	MCU ground
39	PTD4/T1CH0	PTD<4>	Bidirectional	Port D I/O — bit 4 / timer 1 channel 0
40	V_{DD}	TGT_VDD	—	MCU power

Table B-2. Target A Connector J3

Pin No.	Mnemonic	Schematic NET	Direction	Signal Description
1	GND	GND	—	Flex cable shield ground
2	CGMXFC	CGMXFC	Analog	External filter capacitor
3	OSC2	OSC2	Out	Crystal amplifier output
4	GND	GND	—	Flex cable shield ground
5	OSC1	OSC1	In	Crystal amplifier input
6	$\overline{\text{RST}}$	TGT_$\overline{\text{RST}}$	In or out	External reset
7	PTC0	TGT_PTC<0>	Bidirectional	Port C I/O — bit 0
8	PTC1	TGT_PTC<1>	Bidirectional	Port C I/O — bit 1
9	PTC2	PTC<2>	Bidirectional	Port C I/O — bit 2
10	PTC3	TGT_PTC<3>	Bidirectional	Port C I/O — bit 3
11	PTC4	PTC<4>	Bidirectional	Port C I/O — bit 4
12	PTC5	PTC<5>	Bidirectional	Port C I/O — bit 5
13	PTC6	PTC<6>	Bidirectional	Port C I/O — bit 6
14	PTE0/TxD	PTE<0>	Bidirectional	Port E I/O — bit 0 / SCI transmit data
15	PTE1/TxD	PTE<1>	Bidirectional	Port E I/O — bit 1 / SCI receive data
16	$\overline{\text{IRQ}}$	TGT_$\overline{\text{IRQ}}$	In	External interrupt
17	PTD0/$\overline{\text{SS}}$	PTD<0>	Bidirectional	Port D I/O — bit 0 / SPI slave select
18	PTD1/MISO	PTD<1>	Bidirectional	Port D I/O — bit 1 / SPI master in/slave out
19	GND	GND	—	Flex cable shield ground
20	PTD2/MOSI	PTD<2>	Bidirectional	Port D I/O — bit 2 / SPI master out/slave in
21	PTD3/SPSCK	PTD<3>	Bidirectional	Port D I/O — bit 3 / SPI serial clock
22	NC	None	—	No connection
23	V_{SS}	GND	—	MCU ground
24	GND	GND	—	Flex cable shield ground
25	NC	(none)	—	No connection
26	V_{DD}	TGT_VDD	—	MCU power
27	PTD4/T1CH0	PTD<4>	Bidirectional	Port D I/O — bit 4 / timer 1 channel 0
28	PTD5/T1CH1	PTD<5>	Bidirectional	Port D I/O — bit 5 / timer 1 channel 1

Table B-2. Target A Connector J3 (Continued)

Pin No.	Mnemonic	Schematic NET	Direction	Signal Description
29	PTD6/T2CH0	PTD<6>	Bidirectional	Port D I/O — bit 6 / timer 2 channel 0
30	PTD7/T2CH1	PTD<7>	Bidirectional	Port D I/O — bit 7 / timer 2 channel 1
31	PTB0/AD0	PTB<0>	Bidirectional	Port B I/O / ADC input — bit 0
32	PTB1/AD1	PTB<1>	Bidirectional	Port B I/O / ADC input — bit 1
33	PTB2/AD2	PTB<2>	Bidirectional	Port B I/O / ADC input — bit 2
34	PTB3/AD3	PTB<3>	Bidirectional	Port B I/O / ADC input — bit 3
35	PTB4/AD4	PTB<4>	Bidirectional	Port B I/O / ADC input — bit 4
36	PTB5/AD5	PTB<5>	Bidirectional	Port B I/O / ADC input — bit 5
37	PTB6/AD6	PTB<6>	Bidirectional	Port B I/O / ADC input — bit 6
38	GND	GND	—	Flex cable shield ground
39	PTB7/AD7	PTB<7>	Bidirectional	Port B I/O / ADC input — bit 7
40	GND	GND	—	Flex cable shield ground

Table B-3. Target B Connector J4

Pin No.	Mnemonic	Schematic NET	Direction	Signal Description
1	V_{DDAD}/V_{REFH}	TGT_VDD	—	ADC analog power / voltage reference high
2	NC	—	—	No connection
3	NC	—	—	No connection
4	NC	—	—	No connection
5	NC	—	—	No connection
6	NC	—	—	No connection
7	V_{SSAD}/V_{REFL}	GND	—	ADC analog ground / voltage reference low
8	NC	—	—	No connection
9	GND	GND	—	Flex cable shield ground
10	PTA0/KBD0	TGT_PTA<0>	Bidirectional	Port A I/O / keyboard interrupt — bit 0 Unavailable MCU connection
11	PTA1/KBD1	PTA<1>	Bidirectional	Port A I/O / keyboard interrupt — bit 1
12	PTA2/KBD2	PTA<2>	Bidirectional	Port A I/O / keyboard interrupt — bit 2
13	PTA3/KBD3	PTA<3>	Bidirectional	Port A I/O / keyboard interrupt — bit 3
14	GND	GND	—	Flex cable shield ground
15	PTA4/KBD4	PTA<4>	Bidirectional	Port A I/O / keyboard interrupt — bit 4
16	PTA5/KBD5	PTA<5>	Bidirectional	Port A I/O / keyboard interrupt — bit 5
17	PTA6/KBD6	PTA<6>	Bidirectional	Port A I/O / keyboard interrupt — bit 6
18	PTA7/KBD7	TGT_PTA<7>	Bidirectional	Port A I/O / keyboard interrupt — bit 7
19	V_{DDA}	TGT_VDD	—	PLL analog power
20	V_{SSA}	GND	—	PLL analog ground
21	NC	—	—	No connection
22	NC	—	—	No connection
23	NC	—	—	No connection
24	NC	—	—	No connection
25	NC	—	—	No connection
26	NC	—	—	No connection
27	GND	GND	—	Flex cable shield ground

Table B-3. Target B Connector J4 (Continued)

Pin No.	Mnemonic	Schematic NET	Direction	Signal Description
28	NC	—	—	No connection
29	NC	—	—	No connection
30	NC	—	—	No connection
31	NC	—	—	No connection
32	GND	GND	—	Flex cable shield ground
33	NC	—	—	No connection
34	NC	—	—	No connection
35	NC	—	—	No connection
36	NC	—	—	No connection
37	NC	—	—	No connection
38	GND	GND	—	Flex cable shield ground
39	NC	—	—	No connection
40	NC	GND	—	Flex cable shield ground

Table B-4. SPGMR Connectors J7 and J8

Pin No.	Schematic NET	Direction	Signal Description
1	GND	—	System ground
2	SP_OSC	In	Output of SPGMR08 4.9152-MHz oscillator
3	VTST	—	Switched variable voltage power supply output from SPGMR08 (8.5 Vdc)
4	SP_$\overline{\text{RST}}$	In	Reset signal from SPGMR08
5	GND	—	System ground
6	COMPARE	Analog	Input to SPGMR08 variable switching power supply regulation circuit
7	NC	—	No connection
8	PTA<0>	Bidirectional	SPGMR08 host communication line to/from MCU sockets, MON08 connector. No connection to target cable connectors
9	VSW_OUT	Analog	Un-switched variable voltage power supply output from SPGMR08
10	VDD	—	Switched +5 Vdc system power from SPGMR08

Table B-5. Auxiliary Port C Logic Analyzer Connector J1

Pin No.	MCU Mnemonic	Schematic NET	Direction	Signal Description
1	PTC5	PTC<5>	Bidirectional	Port C I/O — bit 5
2	PTC6	PTC<6>	Bidirectional	Port C I/O — bit 6

Table B-6. Auxiliary Port D Logic Analyzer Connector J5

Pin No.	MCU Mnemonic	Schematic NET	Dir	Signal Description
1	PTD6/T2CH0	PTD<6>	Bidirectional	Port D I/O — bit 6 / timer1 2 channel 0
2	PTD7/T2CH1	PTD<7>	Bidirectional	Port D I/O — bit 7 / timer 2 channel 1

Table B-7. MON08 Connector J6

Pin No.	Mnemonic	Schematic NET	Direction	Signal Description
1	$\overline{\mathrm{RST}}$_OUT	$\overline{\mathrm{RST}}$_OUT	Out	Reset signal to target system — 0 to +5 Vdc output reflecting state of MCU $\overline{\mathrm{RST}}$ signal
2	GND	GND	—	System ground
3	$\overline{\mathrm{RST}}$_IN	$\overline{\mathrm{RST}}$_IN	In	Reset signal from target system — 0 to +5 Vdc input to control state of MCU $\overline{\mathrm{RST}}$ signal
4	$\overline{\mathrm{RST}}$	$\overline{\mathrm{RST}}$	Bidirectional	External reset — Held at +8.5 Vdc out of reset
5	TGT_$\overline{\mathrm{IRQ}}$	TGT_$\overline{\mathrm{IRQ}}$	In	Reset signal from target system — 0 to +5 Vdc input to control state of MCU $\overline{\mathrm{IRQ}}$ signal
6	$\overline{\mathrm{IRQ}}$	$\overline{\mathrm{IRQ}}$	Out	External interrupt — Held at +8.5 Vdc in reset and when TGT_$\overline{\mathrm{IRQ}}$ not asserted (low)
7	PTA0/KBD0	TGT_PTA<0>	Bidirectional	Port A I/O / keyboard interrupt — bit 0 Unavailable MCU connection
8	PTA0/KBD0	PTA<0>	Bidirectional	Port A I/O / keyboard interrupt — bit 0 Host I/O present on this pin
9	PTA7/KBD7 after reset	TGT_PTA<7>	Bidirectional	Port A I/O / keyboard interrupt — bit 7
10	PTA7/KBD7	PTA<7>	Bidirectional	Port A I/O / keyboard interrupt — bit 7 Grounded during reset and for 256 cycles after reset
11	PTC0 after reset	TGT_PTC<0>	Bidirectional	Port C I/O — bit 0
12	PTC0	PTC<0>	Bidirectional	Port C I/O — bit 0 Held at +5 Vdc during reset
13	PTC1 after reset	TGT_PTC<1>	Bidirectional	Port C I/O — bit 1
14	PTC1	PTC<1>	Bidirectional	Port C I/O — bit 1 Grounded during reset
15	PTC3 after reset	TGT_PTC<3>	Bidirectional	Port C I/O — bit 3
16	PTC3	PTC<3>	Bidirectional	Port C I/O — bit 3 Grounded during reset

B.7 Target-Cable Pin Assignments

The tables in this section describe the pin assignments for these cables:

- Flex target cable for use with the QFP target head adapter
- MON08 target cable

Table B-8. Flex Target Cable (M68CBL05C) and QFP Target Head Adapater (MCTC08GP20FB44) (Sheet 1 of 3)

QFP Package Pin Number	ICS08GP20 Board Label	Target Head Adapter Pin Number	ICS08GP20 Pin Number
1	$\overline{\text{RST}}$	A12	J3–6
2	PTC0	A15	J3–7
3	PTC1	A16	J3–8
4	PTC2	A19	J3–9
5	PTC3	A20	J3–10
6	PTC4	A23	J3–11
7	PTC5	A24	J3–12
8	PTC6	A27	J3–13
9	PTE0	A28	J3–14
10	PTE1	A31	J3–15
11	$\overline{\text{IRQ}}$	A32	J3–16
12	PTD0	A35	J3–17
13	PTD1	A36	J3–18
14	PTD2	A40	J3–20
15	PTD3	B37	J3–21
16	V_{SS}	B33	J3–23
17	V_{DD}	B30	J3–26
18	PTD4	B25	J3–27
19	PTD5	B26	J3–28
20	PTD6	B21	J3–29
21	PTD7	B22	J3–30
22	PTB0	B17	J3–31
23	PTB1	B18	J3–32
24	PTB2	B13	J3–33

Table B-8. Flex Target Cable (M68CBL05C) and QFP Target Head Adapater (MCTC08GP20FB44) (Sheet 2 of 3)

QFP Package Pin Number	ICS08GP20 Board Label	Target Head Adapter Pin Number	ICS08GP20 Pin Number
25	PTB3	B14	J3–34
26	PTB4	B9	J3–35
27	PTB5	B10	J3–36
28	PTB6	B5	J3–37
29	PTB7	B1	J3–39
30	V_{DDAD}	A1	J4–1
31	V_{SSAD}	A13	J4–7
32	PTA0	A18	J4–10
33	PTA1	A21	J4–11
34	PTA2	A22	J4–12
35	PTA3	A25	J4–13
36	PTA4	A29	J4–15
37	PTA5	A30	J4–16
38	PTA6	A33	J4–17
39	PTA7	A34	J4–18
40	V_{DDA}	A37	J4–19
41	V_{SSA}	A38	J4–20
42	XFC	A4	J3–2
43	OSC2	A7	J3–3
44	OSC1	A11	J3–5
NC	GND	A3	J3–1
NC	GND	A8	J3–4
NC	GND	A17	J3–19
NC	GND	A26	J3–24
NC	GND	A39	J3–38
NC	GND	B34	J3–40
NC	GND	B27	J4–9
NC	GND	B20	J4–14
NC	GND	B8	J4–27
NC	GND	B6	J4–32

Table B-8. Flex Target Cable (M68CBL05C) and QFP Target Head Adapater (MCTC08GP20FB44) (Sheet 3 of 3)

QFP Package Pin Number	ICS08GP20 Board Label	Target Head Adapter Pin Number	ICS08GP20 Pin Number
NC	GND	B4	J4–38
NC	GND	B2	J4–40
NC	NC	B38	J3–22
NC	NC	B29	J3–25
NC	NC	A2	J4–2
NC	NC	A5	J4–3
NC	NC	A6	J4–4
NC	NC	A9	J4–5
NC	NC	A10	J4–6
NC	NC	A14	J4–8
NC	NC	B39	J4–21
NC	NC	B40	J4–22
NC	NC	B35	J4–23
NC	NC	B36	J4–24
NC	NC	B31	J4–25
NC	NC	B32	J4–26
NC	NC	B28	J4–28
NC	NC	B23	J4–29
NC	NC	B24	J4–30
NC	NC	B19	J4–31
NC	NC	B15	J4–33
NC	NC	B16	J4–34
NC	NC	B11	J4–35
NC	NC	B12	J4–36
NC	NC	B7	J4–37
NC	NC	B3	J4–39

Table B-9. Target MON08 Cable

ICS08GP20 and Target Pin No.	ICS08GP20 Board Label	ICS08GP20 and Target Pin No.	ICS08GP20 Board Label
1	$\overline{\text{RST}}$_OUT	9	TGT_PTA7
2	GND	10	PTA7
3	$\overline{\text{RST}}$_IN	11	TGT_PTC0
4	$\overline{\text{RST}}$	12	PTC0
5	TGT_$\overline{\text{IRQ}}$	13	TGT_PTC1
6	$\overline{\text{IRQ}}$	14	PTC1
7	TGT_PTA0	15	TGT_PTC3
8	PTA0	16	PTC3

B.8 Parts List

The parts list for the ICS08GP20 board is given in **Table B-10**.

Table B-10. ICS08GP20 B0ard Parts List

Part Number	Description	Manufacturer	Reference Designator
84-RE21123W01 Rev. C	Printed wiring board (bare)	Multek, Inc.	—
SSB58	Bag, antistatic	Prime	—
C410C104M5U5CA	Capacitor, .1 µF 20% 50 V	KEMET Electronics Corporation	C1, C3–C11
T351E106M025AS	Capacitor, 10 µF 20% 25 V tantalum	KEMET Electronics Corporation	C2, C12
1N5817	Schottky power rectifier	Motorola	CR1
BS170	Transistor, TMOS FET switching	Motorola	Q1, Q2
MF-25-B-10.7K	Resistor, 10.7 K 1% 1/4 W	Yageo Corporation/DigiKey Corporation	R2
MF-25-B-61.9K	Resistor, 61.9 K 1% 1/4 W	Yageo Corporation/DigiKey Corporation	R1
CR-25-B-3.3K	Resistor, 3.3 K 5% 1/4 W	Yageo Corporation/DigiKey Corporation	R3, R7

Table B-10. ICS08GP20 B0ard Parts List (Continued)

Part Number	Description	Manufacturer	Reference Designator
CR-25-B-47K	Resistor, 47 K 5% 1/4 W	Yageo Corporation/DigiKey Corporation	R4–R6
MAX394CPP	IC, low-voltage, quad SPST CMOS analog switch	Maxim Integrated Products, Inc.	U4
MC74HC04AN	IC, hex inverter	Motorola	U5
MC74HC02AN	IC, quad 2-input NOR gate	Motorola	U6
MC74HC4040AN	IC, 12-stage binary ripple counter	Motorola	U7
69192-640	Connector, 2 x 20 pin header straight-retentive leg	Berg Electronics, Inc.	J2-J4
69192-616	Connector, 16-pin header, dual row, straight	Berg Electronics, Inc.	J6
SSW-110-01-G-S	Connector, 10-pin gold header, single row, straight	Samtec, Inc.	J7, J8
24-872	Connector, shunt 2 position	Krista	W1, W2, W3
69192-602	Connector, 2-pin header straight-retentive leg	Berg Electronics, Inc.	J1, J5, W1, W3
69190-603	Connector, 3-pin header straight-retentive leg	Berg Electronics, Inc.	W2
613-7400316	Socket, 40-pin DIP	WELLS-CTI	XU1
IC120-0444-206	Socket, 44-pin PLCC	Yamaichi Electronics USA, Inc.	XU2
IC51-467.KS-11247 or IC51-0444-467	Socket, 44-pin PQFP	Yamaichi Electronics USA, Inc.	XU3
MC68HC908GP32CFB or MC68HC908GP32CP	MCU, 44-pin QFP 68HC908GP20 or MCU, 40-pin DIP 68HC908GP20	Motorola	U3 U1

B.9 Board Layout and Schematic Diagram

Figure B-1 shows the ICS08GP20 board layout and component locations.

The ICS08GP20 schematic diagram occupies the six un-numbered pages that follow.

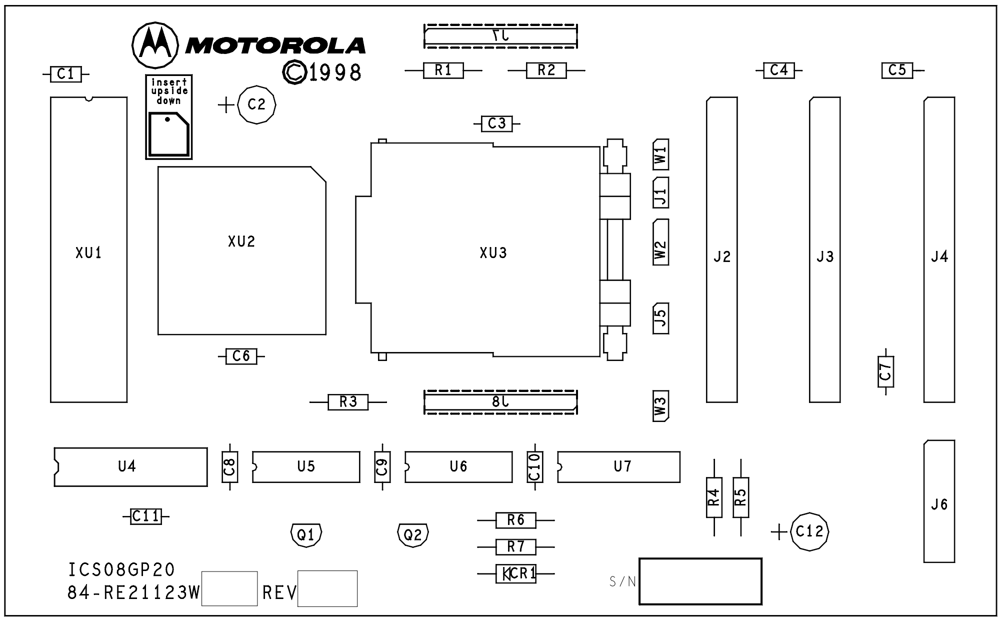


Figure B-1. ICS08GP20 Board Layout

TABLE OF CONTENTS

1	TITLE PAGE
2	SPGMR08 CONN / MONO08 CONN + ENTRY LOGIC
3	GP20 SOCKETS
4	TARGET ADAPTER / LOGIC ANALYZER / CONFIGURATION HEADERS
5	SIGNAL CROSS-REFERENCE
6	UNIT CROSS-REFERENCE

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED
	0	ORIGINAL RELEASE	07-20-98	SM
ALL	A	LOCKED REFS. CHANGED R1 TO 61.9K FOR 8.5V VTST. CHANGED DIVIDER TAP ON U4 FOR 256 CYCLE DELAY. CHANGED JUMPER DESIGNATORS. SPLIT PAGE 3. REPLACED LIVE-BUG PLCC SOCKET WITH DEAD-BUG.	08-13-98	SM
4B2 TO 4D4, 2A2 TO 2D4	B	CHANGED REF DES: W2->J1, W3->W2, W4->J5, W5->W3, J1->J2, J2->J3, J3->J4, J4->J6, J5->J7, J6->J8, U1->U4, U2->U5, U3->U6, U4->U7.	09-08-98	SM
4A2, 4B2	C	MIRRORED J2 HEADER TO MATCH 01-RE91138W01 CABLE	09-18-98	SM

NOTES:

1. UNLESS OTHERWISE SPECIFIED:
   ALL RESISTORS ARE IN OHMS, 5%, 1/8 WATT.
   ALL CAPACITORS ARE IN UF.
   ALL VOLTAGES ARE DC.
2. INTERRUPTED LINES CODED WITH THE SAME LETTER OR LETTER COMBINATIONS ARE ELECTRICALLY CONNECTED.
3. DEVICE TYPE NUMBER IS FOR REFERENCE ONLY. THE NUMBER VARIES WITH THE MANUFACTURER.
4. SPECIAL SYMBOL USAGE:
   * DENOTES - ACTIVE LOW SIGNAL.
   <> DENOTES - VECTORED SIGNALS.
5. INTERPRET DIAGRAM IN ACCORDANCE WITH AMERICAN NATIONAL STANDARDS INSTITUTE SPECIFICATIONS, CURRENT REVISION, WITH THE EXCEPTION OF LOGIC BLOCK SYMBOLOGY.
6. CODE FOR SHEET TO SHEET REFERENCES IS AS FOLLOWS:

```
               5  C7   < >
      SHEET ——/  /      \ \—— OUTPUT
       ZONE ————/        \——— INPUT
```

7. VCC LOCATIONS
   UNLESS OTHERWISE SPECIFIED, VCC IS APPLIED TO:
   PIN 8 OF ALL 8-PIN ICS
   PIN 14 OF ALL 14-PIN ICS
   PIN 16 OF ALL 16-PIN ICS
   PIN 20 OF ALL 20-PIN ICS, ETC.
8. GROUND LOCATIONS
   UNLESS OTHERWISE SPECIFIED, GROUND IS APPLIED TO:
   PIN 4 OF ALL 8-PIN ICS
   PIN 7 OF ALL 14-PIN ICS
   PIN 8 OF ALL 16-PIN ICS
   PIN 10 OF ALL 20-PIN ICS, ETC.

TITLE PAGE

DESIGNED BY: STEVE MULANAX	DATE: 9/18/1998
DRAWN BY: STEVE MULANAX	DATE: 9/18/1998
CHECKED BY:	DATE:
MGR CHECKED BY:	DATE:

MOTOROLA

SEMICONDUCTOR PRODUCTS SECTOR

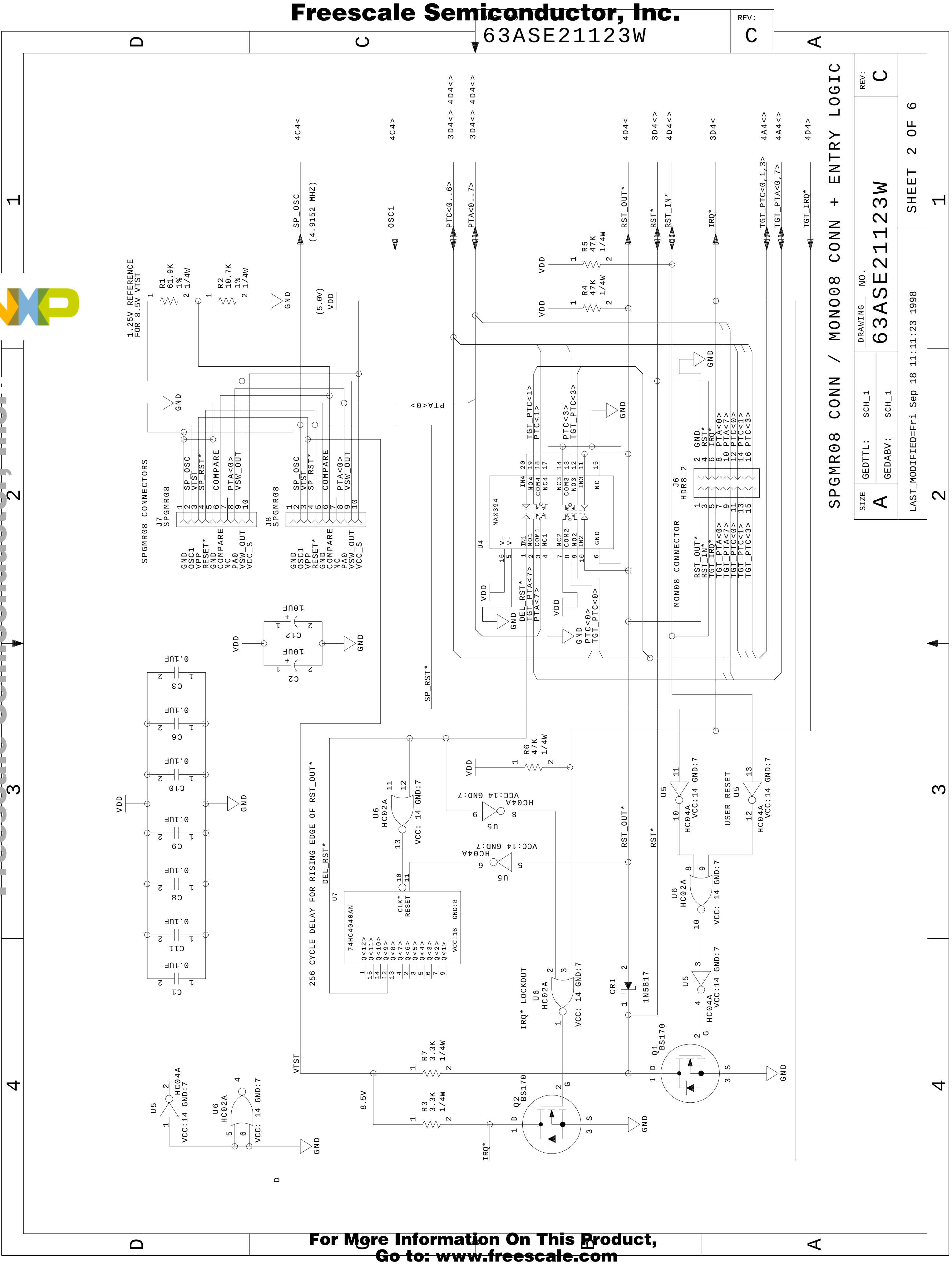
6501 WILLIAM CANNON DRIVE WEST AUSTIN, TEXAS 78735-8598 USA

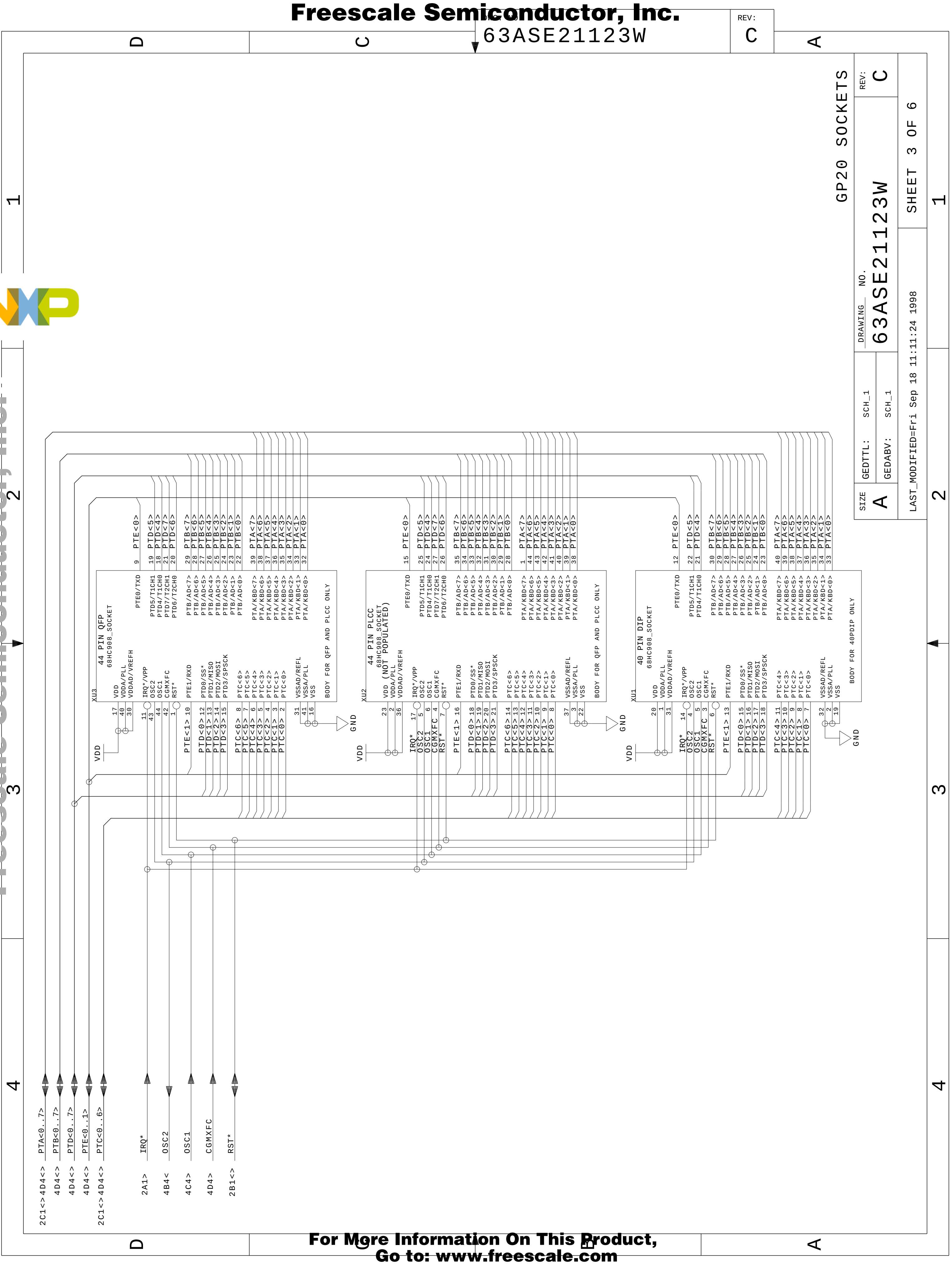
TITLE:
PROGRAMMING ADAPTER / ICS
ICS08GP20

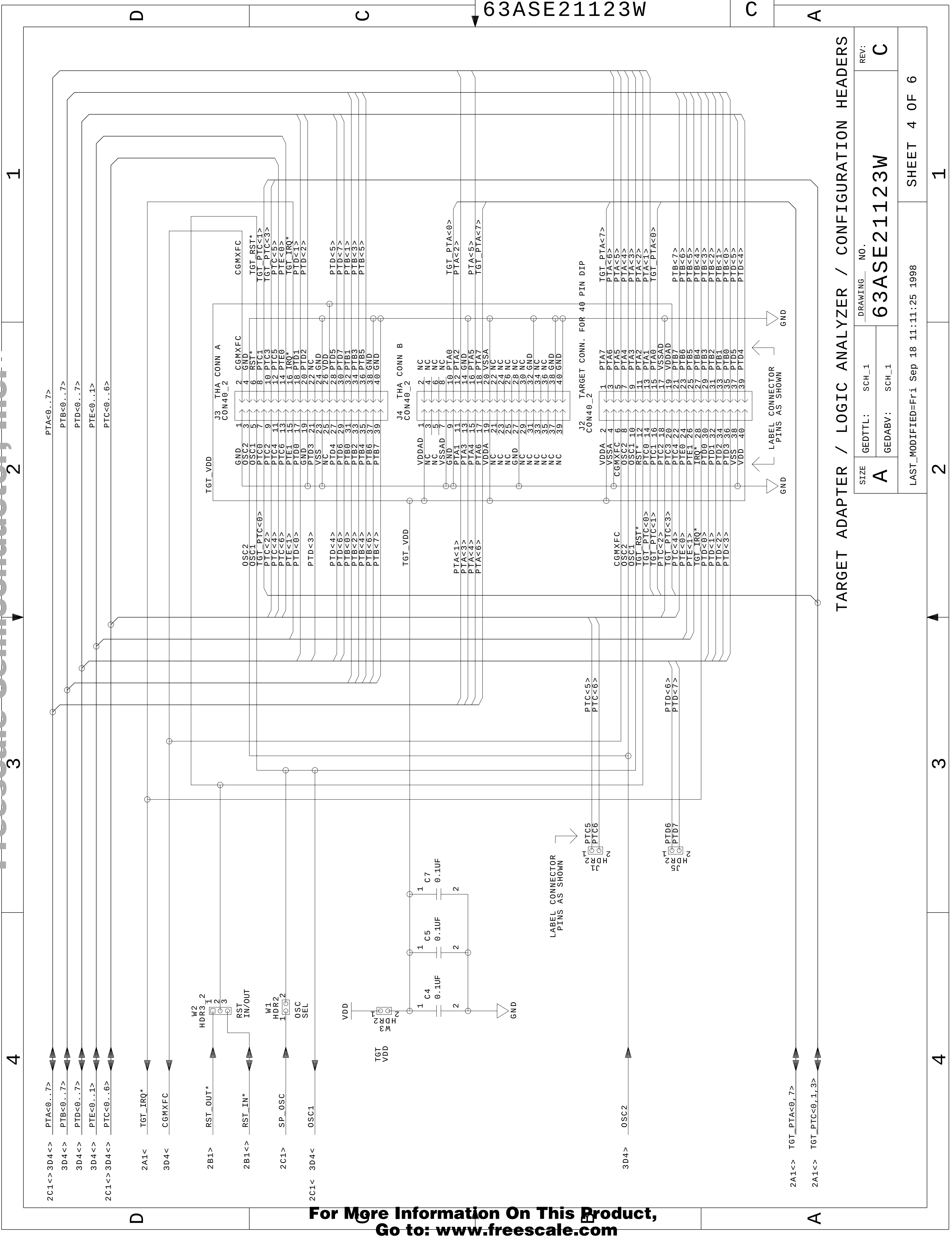
SIZE	GEDTTL: SCH_1 / GEDABV: SCH_1	DRAWING NO.	REV:
A	GEDTTL: SCH_1 GEDABV: SCH_1	63ASE21123W	C

LAST_MODIFIED=Fri Sep 18 11:11:48 1998

SHEET 1 OF 6


SPGMR08 CONN / MONO08 CONN + ENTRY LOGIC
SIZE A
GEDTTL: SCH_1
GEDABV: SCH_1
DRAWING NO. 63ASE21123W
REV: C
SHEET 2 OF 6
LAST_MODIFIED=Fri Sep 18 11:11:23 1998
SPGMR08 CONNECTORS
J7 SPGMR08
J8 SPGMR08
GND OSC1 VPP RESET* GND COMPARE NC PA0 VSW_OUT VCC_S
SP_OSC VTST SP_RST* COMPARE PTA<0> VSW_OUT
1.25V REFERENCE FOR 8.5V VTST
R1 61.9K 1% 1/4W
R2 10.7K 1% 1/4W
(5.0V) VDD
SP_OSC (4.9152 MHZ)
OSC1
PTC<0..6>
PTA<0..7>
RST_OUT*
RST*
RST_IN*
IRQ*
TGT_PTC<0,1,3>
TGT_PTA<0,7>
TGT_IRQ*
4C4< 4C4> 3D4<> 4D4<> 4D4< 3D4> 4D4> 3D4< 4A4<> 4D4>
R5 47K 1/4W
R4 47K 1/4W
U4 MAX394
TGT_PTC<1> PTC<1> PTC<3> TGT_PTC<3>
DEL_RST* TGT_PTA<7> PTA<7>
PTC<0> TGT_PTC<0>
J6 HDR8_2
MONO08 CONNECTOR
RST_OUT* RST_IN* TGT_IRQ* TGT_PTA<0> TGT_PTA<7> TGT_PTC<0> TGT_PTC<1> TGT_PTC<3>
GND RST* IRQ* PTA<0> PTA<7> PTC<0> PTC<1> PTC<3>
C12 10UF
C2 10UF
C1 C3 C6 C8 C9 C10 C11 0.1UF
SP_RST*
256 CYCLE DELAY FOR RISING EDGE OF RST_OUT*
DEL_RST*
U7 74HC4040AN
CLK* RESET
VCC:16 GND:8
U6 HC02A VCC: 14 GND:7
U5 HC04A VCC:14 GND:7
R6 47K 1/4W
USER RESET
IRQ* LOCKOUT
CR1 1N5817
Q1 BS170
Q2 BS170
R7 3.3K 1/4W
R3 3.3K 1/4W
VTST
8.5V
VDD
GND


63ASE21123W
REV: C
GP20 SOCKETS
DRAWING NO. 63ASE21123W
SHEET 3 OF 6
SIZE A
GEDTTL: SCH_1
GEDABV: SCH_1
LAST_MODIFIED=Fri Sep 18 11:11:24 1998
44 PIN QFP 68HC908_SOCKET
44 PIN PLCC 68HC908_SOCKET (NOT POPULATED)
40 PIN DIP 68HC908_SOCKET
BODY FOR QFP AND PLCC ONLY
BODY FOR 40PDIP ONLY
XU3
XU2
XU1
VDD
GND
PTA<0..7>
PTB<0..7>
PTD<0..7>
PTE<0..1>
PTC<0..6>
IRQ*
OSC2
OSC1
CGMXFC
RST*


TARGET ADAPTER / LOGIC ANALYZER / CONFIGURATION HEADERS
SIZE A
DRAWING NO. 63ASE21123W
REV: C
SHEET 4 OF 6
GEDTTL: SCH_1
GEDABV: SCH_1
LAST_MODIFIED=Fri Sep 18 11:11:25 1998
J3 THA CONN A CON40_2
J4 THA CONN B CON40_2
J2 CON40_2 TARGET CONN. FOR 40 PIN DIP
LABEL CONNECTOR PINS AS SHOWN
TGT_VDD
GND
W1 HDR2 OSC SEL
W2 HDR3_2 RST IN/OUT
W3 HDR2 TGT VDD
J1 HDR2 PTC5 PTC6
J5 HDR2 PTD6 PTD7
C4 0.1UF
C5 0.1UF
C7 0.1UF
PTA<0..7>
PTB<0..7>
PTD<0..7>
PTE<0..1>
PTC<0..6>
TGT_IRQ*
CGMXFC
RST_OUT*
RST_IN*
SP_OSC
OSC1
OSC2
TGT_PTA<0,7>
TGT_PTC<0,1,3>

```
      *** Signal Cross-Reference ***
         --- for the entire design --

   CGMXFC          3D4< 4D4>
   IRQ *          2A1> 3D4<
   OSC1            2C1< 3D4< 4C4>
   OSC2            3D4> 4B4<
   PTA <7..0>      2C1<> 3D4<> 4D4<>
   PTB <7..0>      3D4<> 4D4<>
   PTC <6..0>       2C1<> 3D4<> 4D4<>
   PTD <7..0>      3D4<> 4D4<>
   PTE <1..0>      3D4<> 4D4<>
   RST *           2B1<> 3D4<>
   RST_IN *        2B1<> 4D4<>
   RST_OUT *       2B1> 4D4<
   SP_OSC          2C1> 4C4<
   TGT_IRQ *      2A1< 4D4>
   TGT_PTA <?..?>  2A1<> 4A4<>
   TGT_PTC <?..?>  2A1<> 4A4<>
```

SIGNAL CROSS-REFERENCE

SIZE A	GEDTTL: SCH_1 / GEDABV: SCH_1	DRAWING NO. 63ASE21123W	REV: C
LAST_MODIFIED=Fri Sep 18 11:11:25 1998		SHEET 5 OF 6	

63ASE21123W REV: C

```
*** Unit Cross-Reference ***
   --- for the entire design --

C1     MOT_CAP    2D4
C2     MOT_TCAP   2C3
C3     MOT_CAP    2D3
C4     MOT_CAP    4C4
C5     MOT_CAP    4C4
C6     MOT_CAP    2D3
C7     MOT_CAP    4C3
C8     MOT_CAP    2D3
C9     MOT_CAP    2D3
C10    MOT_CAP    2D3
C11    MOT_CAP    2D4
C12    MOT_TCAP   2C2
CR1    MOT_SCHTKY 2B4
J1     HDR2       4B3
J2     CON40      4B2
J3     CON40      4D2
J4     CON40      4C2
J5     HDR2       4B3
J6     HDR8_2     2B2
J7     SKT10_1    2D2
J8     SKT10_1    2C2
Q1     BS170      2B4
Q2     BS170      2B4
R1     MOT_RES    2D1
R2     MOT_RES    2D1
R3     MOT_RES    2C4
R4     MOT_RES    2B1
R5     MOT_RES    2B1
R6     MOT_RES    2B3
R7     MOT_RES    2C4
U4     MAX394     2B2
U5     HC04A      2A3 2B3 2B3 2B4 2D4
U6     HC02A      2B3 2B4 2C3 2D4
U7     MC74HC4040A 2C3
W1     HDR2       4C4
W2     HDR3_1     4D4
W3     HDR2       4C4
XU1    HC908GP20  3B3
XU2    HC908GP20  3C3
XU3    HC908GP20  3D3
```

LAST USED	UNUSED
C9	
CR1	
J8	
Q2	
R7	
U7	
W3	
XU3	

63ASE21123W REV: C

UNIT CROSS-REFERENCE

SIZE	GEDTTL: SCH_1 / GEDABV: SCH_1	DRAWING NO.	REV:
A		63ASE21123W	C
LAST_MODIFIED=Fri Sep 18 11:11:26 1998		SHEET 6 OF 6	

Glossary

8-bit MCU — A microcontroller whose data is communicated over a data bus made up of eight separate data conductors. Members of the M68HC08 Family of microcontrollers are 8-bit MCUs.

A — Abbreviation for the accumulator of the M68HC08 MCU.

accumulator — An 8-bit register of the M68HC08 CPU. The contents of this register may be used as an operand of an arithmetic or logical instruction.

assembler — Software program that translates source code mnemonics into opcodes that can then be loaded into the memory of a microcontroller.

a**ssembly language** — Instruction mnemonics and assembler directives that are meaningful to programmers and can be translated into an object code program that a microcontroller understands. The CPU uses opcodes and binary numbers to specify the operations that make up a computer program. Humans use assembly language mnemonics to represent instructions. Assembler directives provide additional information such as the starting memory location for a program. Labels are used to indicate an address or binary value.

ASCII — American Standard Code for Information Interchange. A widely accepted correlation between alphabetic and numeric characters and specific 7-bit binary numbers.

breakpoint — During debugging of a program, it is useful to run instructions until the CPU gets to a specific place in the program and then enter a debugger program. A breakpoint is established at the desired address by temporarily substituting a software interrupt (SWI) instruction for the instruction at that address. In response to the SWI, control is passed to a debugging program.

byte — A set of exactly eight binary bits.

C — Abbreviation for “carry/borrow” in the condition codes register of the M68HC08. When adding two unsigned 8-bit numbers, the C bit is set if the result is greater than 255 ($FF).

CCR — Abbreviation for “condition codes register” in the M68HC08. The CCR has five bits (H, I, N, Z, and C) that can be used to control conditional branch instructions. The values of the bits in the CCR are determined by the results of previous operations. For example, after a load accumulator (LDA) instruction, Z will be set if the loaded value was $00.

clock — A square wave signal that is used to sequence events in a computer.

command set — The command set of a CPU is the set of all operations that the CPU knows how to perform. One way to represent an instruction set is with a set of shorthand mnemonics such as LDA meaning “load A.” Another representation of an instruction set is the set of opcodes that are recognized by the CPU.

condition codes register — The CCR have five bits (H, I, N, Z, and C) that can be used to control conditional branch commands. The values of the bits in the CCR are determined by the results of previous operations. For example, after a load accumulator (LDA) instruction, Z will be set if the loaded value was $00.

CPU — Central processor unit. The part of a computer that controls execution of instructions.

CPU cycles — A CPU clock cycle is one period of the internal bus-rate clock. Normally, this clock is derived by dividing a crystal oscillator source by two or more so the high and low times will be equal. The length of time required to execute an instruction is measured in CPU clock cycles.

CPU registers — Memory locations that are wired directly into the CPU logic instead of being part of the addressable memory map. The CPU always has direct access to the information in these registers. The CPU registers in an M68HC08 are A (8-bit accumulator), X (8-bit index register), CCR (condition code register containing the H, I, N, Z, and C bits), SP (stack pointer), and PC (program counter).

cycles — See CPU cycles.

data bus — A set of conductors that are used to convey binary information from a CPU to a memory location or from a memory location to a CPU; in the M68HC08, the data bus is 8-bits.

development tools — Software or hardware devices used to develop computer programs and application hardware. Examples of software development tools include text editors, assemblers, debug monitors, and simulators. Examples of hardware development tools include simulators, logic analyzers, and PROM programmers. An in-circuit simulator combines a software simulator with various hardware interfaces.

EPROM — Erasable, programmable read-only memory. A non-volatile type of memory that can be erased by exposure to an ultraviolet light source. MCUs that have EPROM are easily recognized by their packaging, a quartz window allows exposure to UV light. If an EPROM MCU is packaged in an opaque plastic package, it is termed a “one-time-programmable” OTP MCU, since there is no way to erase and rewrite the EPROM.

H — Abbreviation for “half-carry” in the condition code register of the M68HC08. This bit indicates a carry from the low-order four bits of an 8-bit value to the high-order four bits. This status indicator is used during BCD calculations.

I — Abbreviation for “interrupt mask bit” in the condition code register of the M68HC08.

index register — An 8-bit CPU register in the M68HC08 that is used in indexed addressing mode. The index register (X) can also be used as a general-purpose 8-bit register (in addition to the 8-bit accumulator).

input-output (I/O) — Interfaces between a computer system and the external world. For example, a CPU reads an input to sense the level of an external signal and writes to an output to change the level on an external signal.

instructions — Instructions are operations that a CPU can perform. Instructions are expressed by programmers as assembly language mnemonics. A CPU interprets an opcode and its associated operand(s) as an instruction.

listing — A program listing shows the binary numbers that the CPU needs alongside the assembly language statements that the programmer wrote. The listing is generated by an assembler in the process of translating assembly language source statements into the binary information that the CPU needs.

MCU — Microcontroller. A complete computer system including CPU, memory, clock oscillator, and I/O on a single integrated circuit.

memory location — In the M68HC08, each memory location holds one byte of data and has a unique address. To store information into a memory location the CPU places the address of the location on the address bus, the data information on the data bus, and asserts the write signal. To read information from a memory location the CPU places the address of the location on the address bus and asserts the read signal. In response to the read signal, the selected memory location places its data onto the data bus.

N — Abbreviation for “negative,” a bit in the condition code register of the M68HC08. In two’s-complement computer notation, positive signed numbers have a 0 in their MSB (most significant byte) and negative numbers have a 1 in their MSB. The N condition code bit reflects the sign of the result of an operation. After a load accumulator instruction, the N bit will be set if the MSB of the loaded value was a 1.

object code file — A text file containing numbers that represent the binary opcodes and data of a computer program. An object code file can be used to load binary information into a computer system. Motorola uses the S-record file format for object code files.

operand — An input value to a logical or mathematical operation.

opcode — A binary code that instructs the CPU to do a specific operation in a specific way. The M68HC08 CPU recognizes 210 unique 8-bit opcodes that represent addressing mode variations of 62 basic instructions.

OTPROM — A non-volatile type of memory that can be programmed but cannot be erased. An OTPROM is an EPROM MCU that is packaged in an opaque plastic package. It is called a “one-time-programmable” MCU because there is no way to expose the EPROM to a UV light.

PC — Abbreviation for program counter CPU register of the M68HC08.

program counter — The CPU register that holds the address of the next instruction or operand that the CPU will use.

RAM — Random access memory. Any RAM location can be read or written by the CPU. The contents of a RAM memory location remain valid until the CPU writes a different value or until power is turned off.

registers — Memory locations that are wired directly into the CPU logic instead of being part of the addressable memory map. The CPU always has direct access to the information in these registers. The CPU registers in the M68HC08 are A (8-bit accumulator), X (8-bit index register), CCR (condition code register containing the H, I, N, Z, and C bits), SP (stack pointer), and PC (program counter). Memory locations that hold status and control information for on-chip peripherals are called I/O and control registers.

reset — Reset is used to force a computer system to a known starting point and to force on-chip peripherals to known starting conditions.

S-record — A Motorola standard format used for object code files.

simulator — A computer program that copies the behavior of a real MCU.

source code — See source program

SP — Abbreviation for stack pointer CPU register in the M68HC08 MCU.

source program — A text file containing instruction mnemonics, labels, comments, and assembler directives. The source file is processed by an assembler to produce a composite listing and an object file representation of the program.

stack pointer — A CPU register that holds the address of the next available storage location on the stack.

V_{DD} — The positive power supply to a microcontroller (typically 5 volts dc).

V_{SS} — The 0 volt dc power supply return for a microcontroller.

Word — A group of binary bits. Some larger computers consider a set of 16 bits to be a word, but this is not a universal standard.

X — Abbreviation for “index register,” a CPU register in the M68HC08.

Z — Abbreviation for “zero,” a bit in the condition code register of the M68HC08. A compare instruction subtracts the contents of the tested value from a register. If the values were equal, the result of this subtraction would be zero so the Z bit would be set; after a load accumulator instruction, the Z bit will be set if the loaded value was $00.

Index

Symbols

%FILE% 55, 56, 70, 76, 79, 324

A

A command. . . . 197
ACC command 197
Accumulator value 136
ADCLR command 198
ADDI command 199
adding
  breakpoints 136
  variables 124
-address
  fields in listing file 96
address. . . . 105
ASCF6 command 200
ASCII
  characters 339
  constants 99
  files 117
  format 127
ASCIIF3 command. . . . 200
ASM command. . . . 201
assembler
  comments 99
  conditional assembly. . . . 102
  constants 98
  description. . . . 42
  directives. . . . 97, 99, 100, 102, 103, 111
  error messages. . . . 109
  files 44
  interface. . . . 92
  listing directives 104

operands 98
options. 66
third-party 97, 111
Assembler/Compiler
conditional assembly. 102
file button 92
interface. 92
options. 66
parameters 94
preferences 76
progress window 76
quick start 33
tab 73
Auto-Indentation 72
Auto-Save All Files. 70
Auto-Save Current Project 70

B

Back to Editor toolbar button 53
base address
setting 122
batch files
error output 78
EXE path. 74
BELL command 202
BF command. 203
BR command 204
branching. 105
BREAKHX command 208
Breakpoint Window 136
breakpoints 43
adding 136
deleting 137
editing 137
removing 137
setting 122
BREAKSP command 210
Browse 44
buttons, toolbar 56

C

C command 212
CAPTURE command 213
CAPTUREFILE command 214
CASM08W
  assembler comments 99
  assembler directives 99
  assembler options 98, 324
  assembler parameters 94
  conditional assembly. 102
  cycle adder 100
  description. 42, 92, 115
  environment 324
  error messages. 109
  files 44
  hotkeys 53
  INCLUDE directive 102
  installing 44
  MACRO directive. 103
  macros 104
  object files. 116
  outputs. 42, 95
  pseudo-operations 107
  quick-start 33
  sample project 323
  substituting 42, 97, 111
  toolbar buttons 53
  user interface. 93
  window 93, 327
CC value 105
CCR command 215
CF command. 214
CGMXCLK command 216
changes
  bases 125
  CPU information. 130
  restoring 64
  reverses 63
  save options. 69
  saving 61
  software startup 118
checksum. 334
child windows 50

CHIPMODE command 218
clearing
  Clear All 125
  markers 54
  variables 125
CLEARMAP command 219
CLEARSYMBOL command 220
client windows
  WinIDE 50
closing files
  current project file 68
  WinIDE 61
code timing 132
Code Window 122, 311
code/data 334
Color 81
column numbers 51
command
  buffer 140
  syntax 154
command-line
  entering commands 128
  parameters 55
comments 99
communications, pod-to-host 27, 45
compiler 74
  options 66
  third-party 117
  types 74
conditional assembly 102
configuration
  jumpers 36
configuring external programs in WinIDE 74
Confirm command line 77, 79
copying text 65
counting
  cycles 132
CPU
  registers 130
  results 132
Create Backup 72

creating
  new file
    WinIDE . . . 60
  script files . . . 117
  source files . . . 63
Currently edited filename . . . 70
customizing environments . . . 32
cycles
  adding . . . 100
  counter. . . . 105
  counting. . . . 132
  Cycle Adder . . . 100
  Cycle Cntr . . . 105
  CYCLES command . . . 223
  field in listing file . . . 96
  including information in list file . . . 75
CYCLES command . . . 221, 223

D

DASM command . . . 224
DDRA command . . . 225
DDRB command. . . . 226, 229
DDRC command. . . . 227
DDRD command . . . 228
debuggers
  files . . . 75, 92, 95, 117
  ICD08SW
    description. . . . 28, 43, 69, 115, 172
    limitations . . . 172
    starting. . . . 33, 78, 175
    tips. . . . 173
    using . . . 43, 176
  ICS08GPW
    description. . . . 115, 153
    routines . . . 148
    starting. . . . 33, 78
    using . . . 42, 43
  MON08 interface . . . 39, 329
  options. . . . 66
  security . . . 34
  third-party . . . 73

debugging commands
  detailed listings 196
  detailed listings
    See also individual command names 25
  ICD08SW
    command recall 186
    description 185
    summary 187
    syntax 185
  ICS08GPW
    entering 140
    summary 155
    syntax 154
deleting
  breakpoint 137
  text 65
  variables 125
desktop information 67
direct addressing mode 108
Direction 82
distribution media 44
drive space 27
DUMP command 230

E

editing
  breakpoints 137
  options, WinIDE 63, 71
  source files 63
  text 42
editor
  description 42
  files, saving 70
  options 71
  options, WinIDE 63
environment
  building 52
  customizing 32
  Environment Menu 32, 67
  options 66
  path 74

settings . . . 66, 67
storing settings . . . 66
EQU . . . 108
equate directive . . . 108
errors
files . . . 78, 97
format . . . 77
messages, assembler . . . 109
output batch file . . . 78
EVAL command . . . 231
examples
changing number format . . . 125
conditional assembly directives . . . 102
labels . . . 107
listing table . . . 106
macro directive . . . 103
S records . . . 336
EXE 1 (In-Circuit Simulator) tab . . . 78
EXE 2 (Programmer) tab . . . 78
EXE 3 (Debugger) tab . . . 78
EXE Path . . . 79
executable options . . . 66
executing
source code . . . 122
Exit Application . . . 70
EXIT command . . . 232
exiting
application . . . 70
WinIDE . . . 59, 62
exitingWinIDE . . . 70
Expand Includes in List . . . 75
Expand Macros in Listing . . . 75
extended addressing mode . . . 108
extension
specifying . . . 55
External Program toolbar buttons . . . 53
external programs . . . 78
configuring in WinIDE . . . 74
running . . . 73

F

FCB directive 108
FDB directive 109
features 28
fields
  listing files. 96
file options
  WinIDE. 59
filename
  storing as parameter 69
filename parameter 55
filename types 71, 73
files
  ASCII 117
  CASM08W 44
  ICD08SW 44
  ICS08GPW 44
  listing 327
  map 43
  object code 43
  printing 62
  PROG08SW 44
  S19 . . . . . 43, 116
  saving 61, 70
  script 117
  startup 119
  WinIDE. 44
file types
  WinIDE. 52
FILL command 203
Find Next button 82
Find what textbox 83
Fixed Tabs. 72
FLASH memory 26, 27, 28, 43, 164, 165, 173, 257, 343, 344, 351, 352
font settings. 66
Form Constant Byte 108
Form Double Byte. 109

G

G command 233
General Editor options 66
General Editor Tab 71
General Environment options 66
General Environment Tab 69
General Options 72
Give user option to save each file 71
GO command 233
GOEXIT command 234
GOMACRO command 236
GONEXT command 235
Gotil Address at Cursor 122
GOTIL command 237
GOTOCYCLE command 238

H

H command 239
hardware
  installation 35
  pod 25
  requirements 27
  specifications 29
HELP command 240
HEX format 127
hexadecimal number format 124
hexadecimal values
  field in listing file 96
hotkeys 28, 53, 92
HREG command 239
humidity 29
HX command 241

I

I command 242
I/O 26

ICD08SW 43
  Code Window 177
  Colors Window 183
  command recall 186
  CPU Window 184
  debugging commands 185
  description 26, 27, 28, 43, 49, 115, 172
  files 44
  hotkeys 53
  installing 44
  Memory Window 181
  MON08
    interface 172
    limitations 172
    tips 173
  parameters 175
  starting 33, 78, 175
  Status Window 176
  toolbar buttons 53
  user interface 176
  Variables Window 180
ICS08GP software
  See software 25
ICS08GP20 board 26, 31, 35, 37, 38, 43, 174, 329, 331, 342, 355
  parts list 367
ICS08GPW
  Breakpoint Window 135
  Code Windows 121
  CPU Window 130
  Cycles Window 132
  debugging commands 155
  description 27, 43
  Execute Options 148
  File Menu 143
  file options 143
  files 44, 146
  hotkeys 53
  installing 44
  Memory Window 126
  menu options
    Close Logfile 148
    Exit 148
    Go 149

Load S19 File 144
Multiple Step. 149
Open Logfile 146
Open Window 151
Play Macro 145
Record Macro 145
Reload Desktop. 152
Reload Last S19 144
Reset Processors 149
Save Desktop 152
Stop 150
Stop Macro 146
Stack Window. 132
stand-alone mode 43
starting. 33
Status Window 128
toolbar buttons 53, 140
Trace Window. 134
user interface. 115, 120
Variables Shortcut menu. 124
Variables Window 124
Window Options 150
INCLUDE. 325
INCLUDE directive 102
included files
expanding 75
using 70
indentation. 71
INFO command 243
INPUTA command 244
INPUTB command 245
INPUTC command 246
INPUTD command 247
INPUTE command 248
INPUTS command 249
installing
hardware pod. 37
software. 41, 44
INT command 250
integrated development environment 27
interactive mode 35

internal registers 130
Interrupt Stack. 133
IRQ command. 250

J

jumpers
  configuration 36

L

labels 104, 105, 107, 124
LED
  socket power 32, 344, 346, 348, 352
  system power 32, 38, 345, 346, 351
LF command 117
line count. 105
  field in list file. 96
line numbers 51
lines
  total 51
listing directives 104
listing files. 52, 75, 96, 105, 327
Listing Options 75
listing table 106
LISTOFF command 252
LISTON command 253
LOAD command. 254
Load S19 File 144
LOAD_BIN command 259
LOADALL command. 255
LOADDESK command 256
loading
  map files 144
LOADMAP command 257
LOADV command 258
LOADV_BIN command. 260
logfiles. 117
  opening 146
  specifying 146
LOGFILE command. 117

M

M68HC08 Serial Programmer
  See SPGMR . . . 25
M68ICS08GP in-circuit simulator kit
  components . . . 26
  description . . . 25
  features . . . 28
  hardware . . . 26, 29
  quick-start . . . 31
M68SPGMR08 Serial Programmer
  See SPGMR . . . 25
machine cycles . . . 105
MACRO command . . . 261
MACRO directive . . . 103
MACROEND command . . . 263
macros
  expanding . . . 75
  forward referencing . . . 104
  jumping from . . . 104
  recording . . . 145
  stopping . . . 146
MACROSTART command . . . 264
MACS command . . . 265
Main Filename . . . 70
Main Filename option
  WinIDE . . . 76
managing
  Code Window contents . . . 122
  open windows . . . 84
  project information . . . 66
  variables . . . 124
  WinIDE files . . . 59
MAP command . . . 266
Map Files . . . 95
map files . . . 43
  loading . . . 144
markers
  clearing . . . 54
  moving . . . 54, 55
  setting . . . 54
  using . . . 54
Match Case . . . 82, 83

Match Whole Word Only 82, 83
MCU 25, 26, 27, 133
MD command 267
MD2 command 268
media. 44
memory
  address. 105, 334
  map 127
  modifying 126
  system requirements 27
  viewing 126
Memory Window
  Shortcut menu 127
menus
  WinIDE. 58
microcontroller unit
  See MCU. 25
MM command. 126, 269
modifying
  memory 126
  memory bytes 126
MON08 . . . . 26, 39, 164, 169, 172, 173, 329, 343, 344, 351
monitor mode
  See MON08 25

N

N command. 271
navigating
  in IDE environment. 53
  in sour files 54
No Trace Available 134
NOBR command. 272
NOMAP command 273
NOSYMBOL command 274
number format. 124

O

object files. 43
opcode mnemonics 98

Open Logfile 146
Open Project 32
opening
  logfiles. 146
  WinIDE files 60
operating system 27
Options 79
ORG 109
originate directive 109
Other Assembler/Compiler 76
Output Debug File. 75
Output Listing File 75
Output S19 Object. 75

P

parameters
  command-line 55
  ICD08SW 175
  ICS08GPW 94
  PROG08SW 165
  WinIDE. 55
parts list. 367
pasting text 65
path, EXE 74
PC command. 275
PIF files. 74
Play Macro 145
pod
  communications 27, 32, 45, 118
  description. 26, 43, 52, 342, 344
  installation. 31, 35, 37
  power. 32, 38, 52
  troubleshooting 344, 345
POD command 276
PORTA command. 277
PORTB command. 278
PORTC command. 279
PORTD command. 280
PORTE command. 281

ports
- serial 27

power
- connector 37
- LED 32, 344, 345, 346, 348, 351, 352
- requirements 29
- supply,connecting 32, 38

power connector 37
PPF files 67
printing 59, 62
processor cycles
- viewing number 132

PROG08SW
- command usage and syntax 166
- commands
  - BM Blank-check Module 166
  - CM Choose Module 166
  - EM Erase Module 167
  - HE Help 169
  - PB Program Bytes 167
  - PM Program Module 167
  - QU Quit 169
  - RE Reset CPU 169
  - SM Show Module 167
  - SS Specify S record 168
  - UM Upload Module 168
  - UR Upload Range 168
  - VM Verify Module 168
  - VR Verify Range 168
- description 26, 28, 43, 69, 115, 164
- files 44
- hotkeys 53
- installing 44
- parameters 165
- programming example 169
- security 34
- starting 33, 165
- toolbar buttons 53

program counter
- setting 122

programming C devices
- equipment 73

project
    environment 52, 66
    files 67
    name 66
    sample 323
    saving 67
PRTA command 277
PRTB command 278
PRTC command 279
PRTD command 280
PRTE command 281
pseudo operations 107

Q

quick start 31
QUIET command 282
QUIT command 232

R

R command 283
RAM 27, 28, 43, 109, 164, 172, 173, 174
Real Tabs 73
record length 334
Record Macro 145
record type 334
recording macros 145
Recover Error from Compiler 77
Redo
    WinIDE 64
REG command 284
Reload Last S19 144
REM command 285
removing
    breakpoints 137
requirements
    hardware 27
    host 116
    host computer 27
    software 27

Reserve Memory Byte 109
reset
    microcontroller 148
RESET command 286
RESETGO command 287
reversing changes 63
RMB directive. 109
ROM 173
RS-232 serial connector 37
RTI 132
RTS 132
RUN command 233

S

S19 files 43, 116
Sample. 81
sample project 323
Save all files before running 80
Save files before Assembling 76, 77
SAVEDESK command. 291
saving
    changing options 69
    files 61
    projects 67
Saving the Project 70
SCCLR command. 288
SCDI command. 289
SCDO command 290
scratch pad files. 117
Script Files 117
security feature 34
security feature, M68HC908GP20 34
Select Source Module 122
selecting text 65
serial port
    connector. 31, 37, 38
    use 27
Serial Programmer
    See SPGMR 25
Set Base Address. 122

Set PC at Cursor 122
setting
  base address 122
  breakpoints 122
  markers 54
  program counter 122
Setup 44
setup
  ICS08GP software 31
Show as HEX and ASCII 127
Show as HEX Only 127
Show Assembler Progress 76
Show Cycles in Listing 75
Show Disassembly 123
Show Source/Disassembly 123
SHOWBREAKS command 135, 292
SHOWCODE command 293
SHOWMAP command 294
SHOWTRACE command 134, 296
SIM08 command 297
simulator 73
SLD map files 144
Smart Tabs 73
SNAPSHOT command 298
socket power LED 32
software
  CASM08W assembler
    See CASM08W 25
  ICD08SW in-circuit debugger
    See ICD08SW 25
  ICS08GP
    See specific programs 25
  ICS08GPW in-circuit simulator
    See ICS08GPW 25
  installing 44
  loading 41
  modifying startup 118
  PROG08SW FLASH programmer
    See PROG08SW 25
  requirements 27
  starting 45
  third-party 42, 43, 57, 73, 97, 111, 115, 117

WinIDE
See WinIDE . . . 25
Sound Bell on Error . . . 76
source code . . . 106
assembly mode . . . 121
disassembly mode . . . 121
editing . . . 42
executing . . . 122
files . . . 52
source files
creating . . . 63
editing . . . 63
preparing . . . 97
source window
WinIDE . . . 51
source-level debugging . . . 95
SOURCEPATH command . . . 299
SP command . . . 300
SPCLR command . . . 301
SPDI command . . . 302
SPDO command . . . 303
specifications
hardware . . . 29
specifying
ASCII constants . . . 99
SPFREQ command . . . 304
SPGMR . . . 26, 31, 32, 35, 36, 37, 38, 43, 164, 172, 342, 344, 345, 351
Split Bar . . . 89
Split Pointer . . . 89
S records
content . . . 334
creating . . . 336
overview . . . 333
S0 header record . . . 337
S0 record . . . 337
S1 record . . . 338
S9 record . . . 339
termination record . . . 339
types 335
SS command . . . 305
ST command . . . 306
STACK command . . . 307

stack data interpretations. 133
stack pointer value 132, 136
stack values
  viewing 132
stand-alone mode 35, 118
stand-alone mode, ICS08GPW 43
Start Menu. 44, 45
starting
  CASM08W 94
  ICD08SW 175
  ICS08GP software 45
  PROG08SW 165
  WinIDE. 52
startup files 119
STARTUP.08 119
status bar
  WinIDE. 51
STATUS command. 308
STEP command 306
STEPFOR command. 309
STEPTIL command 310
Stop Macro 146
storing
  desktop information 67
  environment settings. 66
  executable options. 66
Strikeout 81
Subroutine Stack. 133
switches
  compiler/assembler. 76
SYMBOL command. 124, 311
symbol table
  list file 96
system
  progress. 51
  requirements 116
  status 51
system power LED 38

T

T command 306
Tab Settings 72
tab settings 71
Tab Size 73
target system
  cables. 342, 343, 364
  connecting to 26, 35, 38, 39, 43, 329, 331, 343, 344
  connectors 329
  cycle timing. 132
  debugging 172, 173, 329
  description. 25
  I/O 115
  MCU 342, 343, 344
  MON08 interface 329, 344, 351
  oscillator 36
  power. 37
  programming. 164
  reset 37
  target head adapter 342
temperature
  operating 29
  storage 29
text files. 52
timing code 132
title bar
  WinIDE. 50
Toggle Breakpoint at Cursor. 122
toolbar
  ICS08GPW 140
  WinIDE. 56
TRACE command. 134, 312
tracing
  trace buffer 134
  trace buffer slot numbers 134
  viewing 134
TYPE. 74
Type 79

U

Underline 81
Undo
  WinIDE 63
UPLOAD_SREC command 313

V

V command 314
values on stack 132
VAR command 315
variables
  adding 124
  choosing types 125
  clearing 125
  deleting 125
  managing 124
Variables Window 311
VERIFY command 316
VERSION command 317
viewing
  breakpoints 135
  command results 128
  CPU information 130
  instructions during tracing 134
  memory 126

W

WAIT command 318
Wait for Assembler Result 76
Wait for compiler to finish 77
WHEREIS command 319
Window Base Address dialog 122
Windows 95 26, 27, 29, 45
WinIDE 27, 42
  Assembler/Compiler Tab 73
  closing files 61
  configuration parameters 67
  configuring external programs 74
  description 27, 48
  Edit options 63

Close/New Project 68
Copy 65
Cut. 64
Delete 65
Open Project 67
Paste 65
Redo 64
Save Project 67
Save Project As. 68
Select All. 65
Setup Environment 68
edit options
Undo 63
Environment Menu 67
Environment options. 66
Set Up Fonts 80
exiting 62, 70
file management 59
file options 59
Close File 61
Exit 62
New File 60
Open File. 60
Print. 62
Print Setup. 62
Save File 61
Save File As 61
files 44
file types 52
listing 52
source code 52
text. 52
font information 66
General Environment Tab. 69
INI file. 66
main window. 50
menus 58
printing 59
saving files 61

Search options . . . 81
  Find . . . 82
  Find Next . . . 83
  Go to Line . . . 84
shortcut buttons . . . 56
source directory . . . 66
source window . . . 51
starting . . . 52
status bar . . . 51
title bar . . . 50
toolbar . . . 56
user interface . . . 48
Window options . . . 84
  Arrange Icons . . . 87
  Cascade . . . 85
  Minimize All . . . 88
  Split . . . 89
  Tile . . . 86
windows . . . 49
WINIDE.INI file . . . 32, 66
word wrap . . . 71, 72

X

X command . . . 320
X index register value . . . 136
XREG command . . . 320

Z

Z command . . . 321

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution, P.O. Box 5405, Denver, Colorado 80217, 1-800-441-2447 or 1-303-675-2140. Customer Focus Center, 1-800-521-6274

JAPAN: Motorola Japan Ltd.: SPD, Strategic Planning Office, 141, 4-32-1 Nishi-Gotanda, Shinagawa-ku, Tokyo, Japan, 03-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, New Territories, Hong Kong, 852-26668334

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; http://sps.motorola.com/mfax/; TOUCHTONE, 1-602-244-6609; US and Canada ONLY, 1-800-774-1848

HOME PAGE: http://motorola.com/sps/

Mfax is a trademark of Motorola, Inc.

MOTOROLA

© Motorola, Inc., 1999

M68ICS08GPOM/D

For More Information On This Product,
Go to: www.freescale.com