Debug-56K

Debugger for the Motorola Digital Signal Processors

A Product of Domain Technologies, Inc.

Debug-56K User's Guide, Version 3.00
January 13, 1998

DSPs supported by this software:
DSP560xx
DSP561xx
DSP563xx
DSP566xx
DSP568xx
MC68356 (DSP side)

Domain Technologies, Inc.
1700 Alma Dr., Suite 495
Plano, Texas 75075
Tel.: (972) 578-1121
Fax: (972) 578-1086
E-mail: support@domaintec.com
Web page: http://www.domaintec.com

TABLE OF CONTENTS

CONTENTS

CONTENTS

CONTENTS

CHAPTER 1 - GENERAL INFORMATION

1.1 - Introduction

Debug-56K is a debugger for the MotorolaTM 16 and 24 bit Digital Signal Processors. This product and this manual can be used with any of Motorola's 16-bit and 24-bit DSPs.

1.2 - Introduction to this Manual

This manual covers the functionality of Debug-56K with any of Motorola's 16 and 24 bit DSP, the examples and the commands provided are specific to the 24-bit DSP56002 DSP. A tutorial chapter walks the user through the majority of the features provided by the debugger.

Debug-56K is a hardware independent debugger designed to run on more than one hardware platform. A separate manual describing the specific hardware platform is provided along with this manual.

1.3 - Debuggers' Features

Source level C, assembly, and mixed debugging
On-screen editing
Built-in assembler/disassembler
Multiple memory display modes
Graphic display of memory
Up to 10 data windows
Up to 128 software breakpoints
True real-time hardware breakpoint
Tool-bar for speedy debugging
User definable buttons
Dedicated window for symbols
Adjustable font sizes

1.4 - Installation of the Debugger Software

To install the software do the following:

1. Insert diskette in the floppy drive and execute INSTALL.EXE.
2. Select the destination directory if different from the default directory.
3. Read the README.TXT file if you are installing the debugger for the first time. The README.TXT file provides information not available in this manual.

1.5 - Command Line Parameters

You may need to set one or more command line parameters before executing the debugger to define the debugger's mode of operation. The command line parameters are set in the program properties window. The following is a list of the debugger's command line parameters.

-H	Help, list the command line parameters
-Q	Disable sorting of the symbol tables
-D	Set the debugger in demonstration mode
-I	Initialize to the default screen settings
-G	Specify a screen configuration file name
-F	Do not strip leading 'F' from the symbols
-Sx	Set the stack memory space
-Pn	Set the base port address of the PC card.
-Kn	Set the multiplier of the DSP's Phase Locked Loops (PLL)
-Cn	Set the RS-232 port number (C1, C2, C3, ...)
-Vn	Specify the target DSP

A detailed description of the command line parameters is provided in Chapter 3.

CHAPTER 2 - TUTORIAL

This chapter will guide you through most of Debug-56K. The tutorial uses the DSP demonstration programs supplied with this software.

2.1 - Start Debug-56K

Invoke Debug-56K with the -I command line options, -I stands for initial screen settings.
After invoking Debug-56K the screen becomes divided into four windows: the command window, the unassemble window, the data, and the registers window.
The pull-down menus and the tool-bar appear at the top of the screen, the status line is the last line of the screen. The command window is now the window selected, this means that key strokes will be placed inside the command window.

2.2 - Observe the Data Window

The data window is used to display DSP data. The upper border of the data window has 3 elements:

1. The window name (Data)
2. The radix of the window (HEX for hexadecimal)
3. The label or the address of the first element inside the window. This field is currently empty because no label is available.

Inside the window, the left most column is the address space (X, Y, P, or L), followed by the address (in HEX). The body of the window is the data.

2.3 - Observe the Unassemble Window

The unassemble window is used to display the DSP program. The upper border of the unassemble window has 3 elements:

1. The window name
2. The program display mode (ASM for reverse assembly)
3. The name of the source file. This applies when the unassemble window is placed in the source mode or the mixed mode.

2.4 - Observe the Registers Window

The registers window displays the DSP's internal registers. The radix is displayed on the upper border of the window. The DALU (Data Arithmetic-Logic Unit) registers are displayed either as a whole registers or as separate sub-registers. All the registers are displayed in hexadecimal. Registers A, B, X, and Y can be displayed hex, dec, fra.

2.5 - Observe the Command Window

The command window is used to enter commands. The radix [HEX] means that numbers entered with commands are treated as hexadecimal numbers.

2.6 - Move Around the Screen

Click inside the data window, this will select the data window, now use the cursor control keys or the mouse to move the cursor inside the data window. Select the unassemble window, select the registers window, select the command window.

2.7 - Change Memory

There are two basic ways to change a value: You can either use the CHANGE command from the command window, or do on-screen editing. To use the CHANGE command, select the command window then enter the following at the prompt:
CHANGE X:0 $123456 - This changes data memory location 0 to 123456 hex.
To do on-screen editing, select the data window, move the editing cursor to memory location 0 and type
654321 - This will change memory location at address 0 to 654321 hexadecimal.

2.8 - Load the Assembly Language Program

If you are using a 16-bit DSP load the program DEMO1_16.CLD. If you are using a 24-BIT DSP then load the program DEMO1_24.CLD. You may load the program by using the File pull down menu or by using the LOAD command.

2.9 - Run the DSP Program

Run the DSP by clicking on the GO button (Green arrow picture). You can also run the DSP by entering the GO command from the command window or by pressing function key F5.
Enable the automatic screen update mode by clicking on the update button (DSP package picture). In this mode the DSP is interrupted regularly to refresh the screen.

2.10 - Stop the DSP

Stop the DSP by clicking on the stop button (Stop sign picture). You can also use the FORCE Break command from the command window or the function key Shift-F5 to stop the DSP.

2.11 - Do Single Step

Make sure the DSP is in stop mode, then perform a single step by clicking on the single step button (magnifier with steps picture). You can also use the STEP command or F8 to perform a single step.

2.12 - Do Continuous Step

No button is available for continuous step, use the
Cont step pull-down menu to put the DSP in the continuous step mode. The continuous step mode performs repeated single steps, with a full screen refresh after every instruction.

2.13 - Do Multiple instructions

Stop the DSP. Execute the command STEP 9. This will cause the DSP to execute 9 instructions without any interruption. After the 9th instructions, the DSP is stopped and the screen is updated.

2.14 - Set a Breakpoint

Set a breakpoint, you can set the breakpoint in one of three ways:

1. Place the cursor at the instruction in the unassamble window and <double-click-left-mouse-button>.
2. Use the BREAK command from the command window.
3. Place the cursor at instruction and press function key F9.

Run the DSP with the breakpoint set. The DSP runs and then stops when the program counter reaches the instruction with the breakpoint.

2.15 - Remove the Breakpoint

Remove the breakpoint that was set previously, you can remove the breakpoint in one of four ways:

1. Place the cursor at the instruction with a breakpoint and <double-click-LMB>
2. Use the BREAK command from the command window.
3. Place the cursor at the instruction with the breakpoint and press function key F9.
4. Use the pull-down menu (Breakpoint, Display, Delete) or (Breakpoint, Clear).

2.16 - Change the Radix of the Data Window

Select the data window and then click on the type button (Capital T picture), this will change the radix of the data window. Change the radix of the data window again by clicking on the type button repeatedly. You may also use F2 to change the radix.

2.17 - Open More Data Windows

Debug-56K allows the user to have up to 10 data windows opened simultaneously. One data window is open at this point, open a new data window by clicking on <View, Data> from the pull-down menu.

2.18 - Graphics

To display a block of memory graphically, first select a data window, then click on the graphics button (sine wave picture). The graphics button is placed on the tool-bar only when the data window is selected.

2.19 - Change a Register Value

You can change a register value in one of two ways:

1. Use the change command
2. On-screen editing

2.20 - Display the Symbols

Stop the DSP, load the DSP program again if necessary. Use the pull-down menu mbol, Display to display the program's symbols. You can also display the symbols by executing the SYMBOL command.

2.21 - Use the Symbols Window

From the symbols window, you may select a symbol and then perform a function on the selected symbol.

Set a breakpoint at LOOP by selecting the symbol LOOP and then clicking on the BREAK button.

Place VALUE_A0 in the watch window by selecting the symbol VALUE_A0 and then clicking on the Watch button.

2.22 - Use the Symbols from the Command Window

Select the command window, perform the following commands:

```
CHANGE       VALUE_A0     $1234

COPY         VALUE_A0     VALUE_A1

WATCH        VALUE_A0
```

2.23 - Try the Evaluator

Select the command window, perform the following commands:

```
?             VALUE_A0 + VALUE_A1 + VALUE_A2

EVALUATE      R0 = $100 + $126
```

2.24 - Set Watches

Select the command window, perform the following commands:

```
WATCH        R0
WATCH       VALUE_A0

WATCH       X:0
```

2.25 - Get Help

Debug-56K under Microsoft Windows has extensive built-in help. Press the function key F1 or use the pull-down menu (Help) to get help.

2.26 - Get Command Sensitive Help

In the command window type COPY followed by a space. This will display help on the COPY command in the help line.

2.27 - Edit Commands in the Command Window

The command window has a built-in history buffer to remember previously entered commands. Use the cursor control keys or the mouse to scroll through past commands.

2.28 - Open More Windows

Open the Flags Window. The flags window displays the status register's flags in a binary format

Open the Watch Window. The Watch Window displays user-defined watch variables.

2.29 - Exit Debug-56K

Use the pull-down menu <File, Quit> to exit from Debug-56K. You can also use QUIT command to exit from Debug-56K. The screen configuration is saved automatically.

2.30 - Debug a C program

If you are using a 16-bit DSP load the program DEMO2_16.CLD. If you are using a 24-BIT DSP then load the program DEMO2_24.CLD. Experiment with the following:

- Set a breakpoint on a C line
- Run the C program
- Single step the C program
- Display program in the reverse assembly mode
- Display program in source mode
- Display program in the mixed mode
- Set in automatic screen update mode and run DSP
- Display the symbols
- Examine memory at the sine waves
- Display the sine waves graphically
- Open the watch window
- Place an array in the watch window
- Place a data structure in the watch window
- Zoom the data structure
- Open the local variables window
- Single step and examine the contents of the local variables window

CHAPTER 3 - USING THE DEBUGGER

3.1 - About This Chapter

This chapter provides a description of the debugger user interface. Commands are described in Chapter 4.

3.2 - Command Line Parameters

Command line parameters are used to control the debugger's mode of operation. These parameters are set by using the Microsoft Windows program properties window before the debugger is executed. The following is a description of the command line parameters.

-h	description of all command line parameters
-q	Disable sorting of the symbol table (normally the debugger automatically sorts the symbol table).
-d	Set the debugger in demonstration mode (no target hardware is present). In this mode the DSP memory is simulated on the PC. DSP code may be loaded in the demonstration mode but the code may not be executed.
-l	Initialize screen settings to the default settings. If -l is omitted, the screen settings are read from the file DBG56KW.CFG.
-g	Specify a screen configuration file name (default:DBG56KW.CFG). This feature allows the user to have multiple screen configuration files.
-f	Instructs the debugger not to strip the leading 'F' from the symbols when the symbols are loaded from the symbols table.
-sx	Set the stack memory space to l, x, or y. Default is -sy for 24-bit DSPs. Default is -sx for 24-bit DSPs.
-pn	Set the base port address of the PC card. This parameter is applicable only when the target DSP is on a card residing inside the PC chassis. The default base port address is set to -P240.
-kn	Set the multiplier of the DSP's phase locked loops. This is applicable for the DSP56002 only.
-cn	Set the RS-232 port (c1, c2, c3, ...). If this parameter is omitted, the debugger scans all the RS-232 ports to find the attached hardware. This parameter is applicable only when the PC's serial port is used to control the DSP.

-vn Specify the target DSP. The following are the options available:
-v1 for DSP56001 and DSP56002
-v2 for DSP56004 and DSP56007
-v3 for DSP56003 and DSP56005
-v4 for DSP56156-v5 for DSP56166
-v6 for DSP56301
-v7 for DSP56303
-v8 for DSP56603

No parameter is needed for the DSP568xx processors.

Example

```
C:\DBG56K\DBG56K.EXE -d -I
```
Run Debug-56K in demonstration mode, default screen settings.

3.3 - Description of the pull-down menus

The pull-down menus are used to perform a wide range of commands, functions, and settings. The following is a description of the Debug-56K pull-down menus and options.

3.3.1 - File Menu

Load Load a DSP file into the target DSP memory. The DSP memory can be internal or external to the DSP. The memory space can be X, Y or P. The file formats supported are OMF, COFF, and IEEE-695. Load also loads the symbols for symbolic debugging and source file line information for source level debugging.
To make source level debugging and symbolic debugging possible, the -g switch must be specified when executing the compilation tools. The load files have the following extensions:
.LODOMF files.
.CLDCOFF files.
.ABSIEEE-695 files.
The symbols are sorted automatically by the debugger, unless specified otherwise by the -q command line parameter. The leading “F” is stripped from the symbols unless otherwise specified by the -f command line parameter.
debugger automatically resets the DSP upon load unless otherwise specified by the pull down menu <Config, Reset On Load>.

Open Module	Open a source code module. This opens the selected source code module and places it in the unassemble window. Opening a source code module is applicable in source level debugging only.
View	Place an ASCII file in the view window; this is helpful when the user needs to browse a text file. The file being viewed can not be edited.
Exit	Exit Debug-56K. The screen configuration is saved automatically and The DSP's state is left intact upon exit. Source line information and symbolic information is lost upon exit.
Files List	List of most recently loaded files. A single click on a file name automatically loads the file to the DSP. The list is empty if no files have been previously loaded.

3.3.2 - View Menu

Cache	Open the cache window. The cache window can be opened only when the DSP being used has internal program cache.
Calls	Open the calls history window. The calls window can be opened when C programs are debugged at the source level.
Cmd	Open the command window.
Data	Open a data window. Up to 10 data windows may be opened simultaneously.
DMA	Open the DMA status window. The DSP window can be opened only when the DSP being used has an internal DMA controller.
Flags	Open the flags window.
I/O	Open the peripheral registers window.
Local	Open the C local variables window.
Once	Open the OnCE registers window.
Reg	Open the ALU registers window.

Stack	Open the stack window.
Trace	Open the trace window. The tracw window can be opened only when the DSP being used has an internal trace buffer.
Unasm	Open the unassemble window.
View	Open the ASCII file view window. This window stays empty until an ASCII file is viewed. If a view window is closed and then reopened, the previously viewed ASCII is automatically placed in the view window. A file placed in the view window may not be edited.
Watch	Open the variable watch window. The watch window remains empty until variables are placed in it. If a watch window is closed and then reopened, the previously watched variables are automatically placed in the watch window.

3.3.3 - Run Menu

Run	Run the DSP from program counter value.
Stop	Stop the DSP.
Step	Perform a single step.
Jump	Perform a Jump. Jump is similar to single step, except subroutine calls are treated as one instruction.
Cont Step	Step continuously.
Cont Jump	Jump continuously. In this mode subroutine calls are treated as one instruction.
Update	Toggle the automatic screen update mode. If the DSP is started while the debugger is in the automatic update mode, the debugger's screen is updated continuously.
Reset	Reset the DSP.

3.3.4 - Symbols Menu

Display	Display the symbols in the symbols window. When the symbols are displayed, a symbol can be selected and then a function can be applied to it. The following are the functions that can be performed on a symbol:

Button:	**Description:**
Unassemble	Unassemble memory at symbol
Display	Display memory at symbol
Watch	Watch at symbol
Go To	Run the DSP to instruction
Break	Toggle a breakpoint at symbol
Close	Close the symbol window

Clear	Clear all the symbols. This will put the debugger in the non-symbolic mode of operation.
Load	Load symbols from a program file, all the DSP memory and resources are left intact.
Load On Start	If this flag is set, the symbols are loaded automatically when Debug-56K is started.
Files List	List of most recently loaded files.

3.3.5 - Breakpoint Menu

Display	Display all the breakpoints set. A breakpoint can be selected and then cleared. The close button closes the breakpoint window.
Clear	Clear all breakpoints.
Load	Load breakpoints from a breakpoint file. The breakpoint file is independent of the source and load files. The file name extension for breakpoint files is .BRK.
Save	Save breakpoints into a breakpoint file. The breakpoint file is independent of the source and load files. The file name extension for breakpoint files is .BRK.
Hard Break	Set a hardware breakpoint.

3.3.6 - Config Menu

Case Sens	Case sensitivity on/off. This feature is especially helpful when working with programs with a large set of symbols. If off, case sensitivity is ignored by the debugger. Example: MAIN() will be treated as main().
Colors	Set the windows colors. The colors of individual windows are changed independently or in a group. After being set, the windows colors are saved automatically upon exit from Debug-56K.
Font	Set the screen font.
Comm baud	Set the baud rate. This is applicable when the PC controls the DSP through an RS-232 port.
Command Win	Options and settings for the command window. The following are the Command Window's sub-menus:

Clear History:	Clear the history buffer
History Size:	Set the size of the history buffer
Save Commands:	Save the commands to a file
Save History:	Save the history buffer to a file
Set Prompt:	Set the command line prompt

I/O Port	Set the PC card's I/O base address. This setting is applicable when the debugger is interfacing with the DSP through the DSP's host port.
Quick menu	Configure the tool-bar. The tool-bar may be placed vertically, horizontally, it may be placed in the large or small format. The tool-bar may also be removed from the screen.
Refresh rate	Set the minimum automatic screen refresh rate. This sets the screen refresh rate when the DSP is running in the automatic screen refresh mode. The refresh rate also depends on other factors including the number of data windows opened, the size of the data windows, the speed of the host computer, and the speed of the serial link.
Reset on load	Reset the DSP before a load. In the default mode, the DSP is not reset before a load.
Reset on start	In the normal mode, when Debug-56K is invoked, the DSP is reset automatically. If this switch is turned off, the DSP is kept in it's current state after invoking Debug-56K.

Reverse ASCII	When on, bytes are swapped before they are displayed in an ASCII data window.
Tab size	Set the tab size. The tab size value is used to translate a tab into the specified number of spaces. The tab size affects the view window only.
Verify Memory	If this flag is set, all memory writes by the debugger to the DSP are verified.

3.3.7 - Window Menu

Cascade	Place the resource windows in cascade.
Tile	Tile the resource windows.
Horizontal	Tile the resource windows horizontally.
Vertical	Tile the resource windows vertically.
Arrange icons	Arrange the resource windows icons.
Close all	Close all the opened resource windows.
Windows List	List of all the opened resource windows.

3.3.8 - Help Menu

Index	Get the help index.
Topic search	Search help for a topic.
Using help	How to use the help.
About	Information about Debug-56K.

3.4 - Description of the Tool-bar

The Tool-bar provides a quick and convenient way to enter many of the most often used commands. The following is a detailed description of the Tool-bar buttons.

3.4.1 - Go Button

Run the DSP from the program counter. The GO command and the pull-down menu can also be also used to run the DSP from the program counter.

3.4.2 - Stop Button

Stop the DSP. The FORCE B command and the pull-down menu can also be used to stop the DSP. The debugger's screen is refreshed after the stop is completed.

3.4.3 - Step Button

Single step, this executes a single DSP instruction. The STEP command and the pull-down menu can also be used to perform a single step. The debugger's screen is refreshed after the step.

3.4.4 - Jump Button

Perform a Jump. JUMP is similar to STEP, except that a subroutine call is treated as one instruction. The JUMP command and the pull-down menu can also be used to perform a jump. The debugger's screen is refreshed after the jump.

3.4.5 - Automatic Update Button

Toggle the automatic screen update mode (on/off). In the automatic screen update mode, and when the DSP is running, the DSP is interrupted periodically to update the data windows and the registers window.Other resource windows are not updated. The CONFIG command and the pull-down menu can also be used to toggle the automatic update mode.

3.4.6- Reset DSP Button

Reset the DSP. The FORCE R command and the pull-down menu can also be used to reset the DSP.The debugger's screen is refreshed after the reset.

3.4.7 - Radix Button

Change the radix of the selected window. This button can be used to change the radix of a data window, the registers window, or the command window. The button is not displayed if other windows are selected. The radix of the window is displayed on the upper border of that window. The RADIX command can also be used to change the radix of the command window. Following are the available radices:

Data windows:	Hex, Dec, Fra, Asc, Bin
Graphical Data windows:	Hex, Dec, Fra
Registers window:	Hex, Dec, Fra
Command window:	Hex, Dec, Fra, Bin
Flags window:	Bin
All other windows:	Hex

The radix of the registers window affects the display of registers X, X0, X1, Y, Y0, Y1, A, B only, All other registers in the registers window are always displayed and edited in Hex.

The RADIX command may also be used to change the default radix of the command window.

3.4.8 - Graph Button

The graph button is used to display the selected data window graphically. A graph may be scrolled, scaled, sized, and moved. The Up/Down arrow keys are used to change the vertical graph scaling. A graph can have a hex, dec, or fra radix. One or more data windows can be displayed graphically. The graph button is placed on the screen only when a data window is selected. The DISPLAY command may also be used to toggle the graphics mode (on/off).

3.4.9 - Reverse Assembly Mode Button

Put the unassemble window in the reverse assembly mode. In the unassemble mode, the DSP's memory is reverse assembled and then placed in the unassemble window. The reverse assembly mode button is placed on the screen only if the unassemble window is the window selected. The UNASSEMBLE command may also be used to change the display mode. On-screen editing of the DSP code is permitted in this mode.

3.4.10 - Source Mode Button

Put the unassemble window in the source mode. In this mode the DSP code is read from the source text file and then placed in the unassemble window. The source mode button is placed on the screen only when the unassemble window is selected. If the source line information is not found by the debugger this button is not displayed. The UNASSEMBLE command may also be used to change the display mode. On-screen editing of the DSP program is not permitted in the source mode.

3.4.11 - Mixed Mode Button

3.5 Description of the Resource Windows

Put the unassemble window in the mixed mode. In the mixed mode, reverse assembly and source instructions are interlaced in the unassemble window. The mixed mode button is displayed on the screen only when an unassemble window is selected. If the source line information is not found by the debugger this button is not displayed. The UNASSEMBLE command may also be used to change the displayed mode. On-screen editing of the DSP program is not permitted in the mixed mode.

3.5.1 - The Cache Window

The cache window is used to display the DSP's instruction cache registers. This window is applicable only for DSPs with internal instruction cache.

Tag registers TAG0 to TAG 7 with their respective RLU and LOCK bits are displayed in the cache window. The tag registers are displayed in hexadecimal, the RLU and LOCK bits are displayed in binary. The cache registers window is organized into the following columns:

TAG# 24-bit Tag register RLU bit LOCK bit

3.5.2 - The Calls Window

The calls window displays the calls history for C function. The contents of the calls window are updated automatically when a subroutine is called or when a subroutine is returned from. The calls window is applicable only for C source level debugging.

3.5.3 - The Command Window

The command window is used to enter commands. Commands are entered at the prompt when the command window is selected.

A history buffer is built into the command window, a past command may be executed or edited prior to execution. The user may use the mouse or the keyboard's cursor movement keys to scroll through the history buffer.

An editor is built into the command window. The user may use the mouse or the keyboard's cursor movement keys to edit a command.

The valid radices for the command window are Hex, Dec, Bin, and Fra. The radix of the command window may be changed with the RADIX command or by pressing the T button of the tool-bar.

The pull down menu (config, command) provides a number of functions to program the command window.

Refer to the description of the pull-down menus and to Chapter 4 for further description of the command window.

3.5.4 - The Data Window(s)

Data windows are used to display DSP memory. Up to 10 data windows may be opened simultaneously. X: Y: P: or L: memory may be displayed and edited in a data window. Internal as well as external memory may be displayed in a data window.

The left hand side of a data window is the address column, always in Hex. The upper border of the window has the window name, the radix, and the symbolic location of the first address inside the window (if a symbol exists). The body of the window has the values of the memory locations.

A data window may be displayed and edited in Hex, Dec, Fra, Bin, and ASCII. A data window may be displayed graphically in Hex, Dec, and Fra. The data and the address may be edited on-screen.

3.5.5 - The DMA Window

The DMA window displays registers specific to the DMA controllers. This window is applicable only to DSPs with internal DMA controllers.

3.5.6 - The Flags Window

The flags window is used to display and edit the DSP's status registers in binary format. Although the status register in the registers window provides the same information, the flags window provides a convenient way to read and edit individual flags.

3.5.7 - The I/O Window

The I/O Peripherals window is used to display and edit the DSP's I/O peripheral registers. Non-readable registers are not displayed. Non-writable registers can not be edited. Registers that change the DSP's status when read are not displayed. All values are displayed and edited in Hex.

3.5.8 - The Local Variables Window

The local variables window is used to display the C local variables. The C local variables are placed and removed automatically by the debugger. The local variables window is applicable only in C source level debugging.

3.5.9 - The OnCE Window

The OnCE window is used to display the OnCE port registers. The values are displayed in hexadecimal.

3.5.10 - The Registers Window

The registers window is used to display and edit the DSP's internal registers. Registers X, X0, X1, Y, Y0, Y1, A and B can be displayed and edited in Fra, Dec, or Hex. All other registers are displayed and edited only in Hex.

3.5.11 - The Stack Window

The stack window is used to display and edit the stack. The stack values are displayed and edited in Hex.

3.5.12 - The Trace Window

The trace window is used to display the instruction discontinuity trace buffer. The trace window is applicable only for DSPs with an internal instruction discontinuity trace buffer.

3.5.13 - The Unassemble Window(s)

The Unassemble window is used to display and edit the DSP programs. The unassemble window can be displayed in one of three modes:

Mode:	**Description:**
ASM	Program displayed in reverse assembly mode.
SRC	Program displayed in source mode.
MIX	Program displayed in mixed mode (reverse assembly and source)

The source program must be compiled, assembled, and linked with he -g switch in order for the SRC and the MIX modes to function. On-screen editing is valid in the ASM mode. The unassembe window is organized into the following columns, described from left to right.

Column:	**Description:**
Breakpoint type:	Breakpoint type, or blank if a breakpoint is not set at that instruction.
Address:	Memory space indicator followed by the address. X or P memory may be placed in the unassemble window.
Opcode:	Opcode of the instruction, displayed in Hex.
Second opcode:	Second word of a two word instruction, displayed in Hex.
Mnemonic:	The DSP instruction
Breakpoint #:	Breakpoint number (0 to 128), or blank if a breakpoint is not set at that instruction.

3.5.14 - The View Window

The view window is used to display ASCII text files. Use the pull-down menu <view, view> to open the view window.
Use the pull down menu <file, view> or the VIEW command to place a text file in the view window. Editing of the text file is not allowed inside the view window.

3.5.15 - The Watch Window

The watch window is used to conveniently group variables for examination and editing. A memory location, a register, or a data structure may be placed in the watch window. Watched variables can be displayed in Hex, Dec, Bin, or Fra. A variable may be placed in the watch window by using one of the following three techniques:

1. Use the WATCH command
2. Use the symbols window
3. Place the cursor in the watch window and press the insert key.

A variable may be removed from the watch window by using one of the following two techniques:

1. Use the WATCH command
2. Place the cursor in the watch window on the variable to remove and press the delete key.

Variables placed in the watch window may be edited. To edit a watch variable place the cursor in the watch window, on the variable to edit, then press the space bar or the right mouse button.

Arrays and data structures placed in the watch window may be expanded and compressed. To expand or compress a data structure or an array, place the cursor on the name and double click the left mouse button, or press enter key, right mouse key, or the space key.

Information on the WATCH command is provided in Chapter 4.

3.6 - Status/Help Line

The last line of the display is the Status/Help line. The status/help line is used to display the following:

1. The DSP's program counter value
2. Command sensitive help when a command is being entered
3. List of all available commands when the command window is selected and the space bar is pressed.

3.7 - Function Keys

The keyboard's function keys provide the user with a convenient way to perform some of the most frequently used commands. Function key usage is optional since the tool bar and the pull-down menus provide the same functionality. The following is a description of the function keys.

3.7.1- F1 - Help

F1 displays help.

3.7.2 - F2 - Toggle Radix or Display Mode

For the data windows, the registers window, and the command window, function key F2 toggles the radix. For the unassemble window, function key F2 toggles the program display mode. F2 has no affect on all other window. The specific window must be selected before using F2.

3.7.3 - F3 - Display Data Graphically

F3 toggles the graphic display of a data window (on/off). F3 has no affect on all other windows. A data window must be selected before pressing F3.

3.7.4 - F5 - Go

F5 starts the DSP from the current program counter.

3.7.5 - Shift-F5 - Stop

Shift-F5 stops the DSP.

3.7.6 - F6 - Select Next Window

F6 selects the next window. The current window is deselected and the next window is selected. The opened windows are numbered in a logical order.

3.7.7 - Shift-F6 - Automatic Screen Update

Shift-F6 toggles the automatic screen update mode (on/off). In the automatic screen update mode, when the DSP is started, the DSP is interrupted periodically to refresh the data and the registers windows.

3.7.8 - F7 - Go to Cursor

F7 causes the execution of the DSP program until the program counter reaches the instruction at the cursor. The unassemble window must be selected before pressing F7.

3.7.9 - F8 - Single Step

F8 causes a single DSP instruction to be executed.

3.7.10 - Shift-F8 - Run in Continuous Step

Shift-F8 toggles the continuous step mode (on/off). This runs the DSP with a complete screen refresh after every DSP instruction. The speed of execution depends on the number of windows opened, the speed of the PC, the speed of the link between the PC and the DSP, and the speed of the target hardware.

3.7.11 - F9 - Toggle a Breakpoint

When the unassemble window is selected, F9 sets/clears a breakpoint. To set a breakpoint with F9 move the cursor to the DSP instruction of choice and press F9. To remove the breakpoint with F9 move the cursor to the instruction with the breakpoint and press F9.

3.7.12 - F10 - Jump

F10 instructs the DSP to execute one instruction while treating a subroutine as one instruction. This is similar to single-step with the exception that when a subroutine call is reached the whole subroutine is executed.

3.7.13 - Shift-F10 - Run in Continuous jump

Shift-F10 toggles the continuous jump mode (on/off). This runs the DSP but it causes a complete screen refresh after every DSP instruction, while treating a subroutine call as one instruction. The speed of execution depends on the number of windows opened, the speed of the PC, the speed of the link between the PC and the DSP, and the speed of the target hardware.

3.8 - Symbols and Symbolic Debugging

Symbolic debugging is available for C and assembly language programs. To enable symbolic debugging, the source files must be prepared with the -g switch when executing the compilation tools. Symbolic debugging is supported by COFF and IEEE-695 files.

In symbolic debugging, the user can refer to a variable by the variable's name instead of the variable's numerical address. The variables are named by the user in the source files.

The following are a few examples of symbol usage:

Example 1	`BREAK START` Set a breakpoint at START
Example 2	`DISPLAY VALUE_A0` Display memory at VALUE_A0
Example 3	`GO main` Change PC to main then start the DSP

Symbolic debugging is possible only when the debugger finds the symbols. A menu specific for symbols is available in Debug-56K. The symbol menu can be used to display the symbols, clear the symbols, and load symbols from a file. A flag may be set to instruct the debugger to load symbols upon starting the debugger.

Symbols are displayed in the symbols window. In the symbol window the user can select a symbol and apply a command to that selected symbol. The following are the commands that may be applied to a selected symbol:

Button:	**Description:**
Unassemble	Unassemble at symbol
Display	Display memory at symbol
Watch	Place variable in the watch window
Go To	Execute from program counter until instruction at the symbol.
Break	Set a breakpoint at symbol

3.9 - Source Level Debugging

Source level debugging allows the user to see the DSP programs in the unassemble window exactly as they appears in the source files. Source level debugging is available for DSP programs written in C or assembly language.

In source level debugging, Debug-56K automatically reads the object files, the source code files, and the information that links the object files to the source flies.

To enable source level debugging, the source files must be prepared with the -g command line switch when executing the compilation tools, Debug-56K must also be able to find the source files.

Assembly and C source level debugging is supported by COFF and IEEE-695 files.

3.10 - Debugging C Software

To debug C programs with Debug-56K, the source files must be prepared with the -g switch when executing the compilation tools, Debug-56K must also be able to find the source files. The commands and operations available for assembly language debugging are also available for C debugging.

The local variables window is a C specific window, it is used to display the C local variables. The local variables are placed and removed automatically by the debugger. Arrays and structures inside the local variables window may be expanded and compressed, variables may be edited.

The calls window is a C specific window, it is used to display the C calls history.

C source level debugging is supported by COFF files and IEEE-695 files

3.11 - Macro Commands

Debug-56K allows the user to group a number of commands in a macro command file for batch processing. Refer to Chapter 4 for further information on commands and macro command files.

3.12 - Editing inside a Resource Window

To edit inside a resource window, place the cursor at the variable to be edited then type the new value over the old value. Press the Enter key to accept the new value, press the Esc key to restore the old value.

CHAPTER 4 - COMMANDS

4.1 - Command Entry

Commands are entered in the command window at the command line prompt.

4.2 - Command Help

The last line of the screen displays command sensitive help. Pressing the space bar displays all the commands available.

4.3 - Command Entry Rules

- Only one, two, or three letters of a command need to be typed when entering a command. This feature is designed to reduce the number to keystrokes needed to enter a command.

Example:	C PC 0
is equivalent to:	CHANGE PC 0

-The radix of the command window determines the default input radix. This affects constants as well as addresses being entered. The default radix may be overwritten by preceding a number by: ‘ for decimal, $ for hexadecimal, and % for binary.

-A block of memory may be specified in two ways:

Format 1: The memory block is defined by the starting address and the ending address of the block:

Syntax: start_address..end_address

Example: CHANGE X:0..$FF $1234
Will change memory locations 0 to $FF to $1234

Format 2: The memory block is defined by the starting address of the block and the number of words in the block.

Syntax: start_address#number_of_words

Example: CHANGE X:0#100 $1234
Will change 100 memory locations starting at 0 to 100 to $1234)

- A register block defines a block of DSP registers The block of registers is defined by the first register and the last register of the block.

Syntax: start_register..end_register
Register blocks possible:
X0..X1,
Y0..Y1,
A0..A2,
B0..B2,
R0..R7,
N0..N7,
M0..M7,
Example: CHANGE R0..R7 $1234
Will change registers R0, R1, ..R7 to $1234.

4.4 - Command Help Rules

The last line displays command specific help when a command is being entered. The following is the command help convention:

- A list of the commands is made available by pressing the space bar repeatedly.

- The highlighted part of a command name indicates the least letters that need to be types when entering a command.

- Parameters between brackets [] are for optional parameters.

Example: GO [address]

- Parameters between parentheses () are for descriptive purposes only.

Example: COPY (from) address_block (to) address

- A bar | is used to separate a list of possible command parameters.

Example: FORCER | B | RU

4.5 - Command Editor

To speed-up commands entry, the command window has a built-in command editor with a command history buffer. The mouse and the following keys may be used when a command is being entered or edited.

Key:	**Description**:
↑ ↓	Scroll the command history
_ _	Move cursor left and right
Ins	Toggle insert mode on/off
Del	Delete a character
Home	Move to beginning of command line
End	Move to end of command line
Esc	Delete changes
Enter	Accept and execute command

The vertical scroll-bar can also be used to scroll through the command history buffer.
The pull down menu (config, command) can be used to program the command window.

4.6 - Macro Commands

A number of commands may be grouped in a command file and then executed automatically. The Macro command opens a file of commands MacroName(.CMD) and process the commands in that file. The macro command file can have any combination of valid commands. The commands are read, processed, and executed one at a time. Commands are delimited by a carriage return or a semicolon. Text after a semicolon is ignored and can be used as a command.

A macro command file can have variable parameters. Variable parameters are identified by a circumflex sign followed by a number (^1, ^2, ...). Variable parameters are replaced with their numeric values when the macro is invoked.

Example

```
Radix       hex         ;change radix to hexadecimal
wait         break       ;wait until DSP enters debug mode
change      x:80#10 0    ;clear 16 memory locations
go                      ;run the DSP
force       b           ;stop the DSP
wait        break       ;wait until DSP enters debug mode
step                    ;single step the DSP
wait        break       ;wait until DSP enters debug mode
break       p:7C        ;set a breakpoint at PC=7C
go                      ;run the DSP
wait        break       ;wait until DSP gets in debug
                        ;mode
```

Macro commands are especially helpful when performing automated system tests.

4.7 - List of Commands

Command:	**Brief Description:**
`**AL**IAS`	Define custom command string
`**A**SSEMBLE`	Define custom command string
`**B**REAK`	Breakpoint operations
`**CF**ORCE`	Reset the emulator
`**C**HANGE`	Change register or memory value(s)
`**CON**FIG`	Set system configuration options
`**CO**PY`	Copy a memory block
`**DI**SASSEMBLE`	Disassemble memory
`**D**ISPLAY`	Display memory
`**E**MI`	Define EMI parameters
`**E**VALUATE`	Evaluate expression
`**F**ORCE`	Reset or stop DSP
`**G**O`	Execute DSP program
`**H**ELP`	Display help
`**I**NPUT`	Assign input file
`**J**UMP`	Jump over a subroutine call
`**L**OAD`	Load DSP program
`**LO**G`	Log information
`**O**UTPUT`	Assign output file
`**P**ATH`	Define directory path
`**Q**UIT`	Quit Debug-56K
`**R**ADIX`	Change radix of command window
`**REF**RESH`	Toggle screen refresh (on/off)
`**RE**TURN`	Execute until RETURN instruction

SAVE	Save DSP memory to file
STEP	Single step DSP program
SYMBOL	Display symbols
TRACE	Trace through DSP program
TIME	Display execution time
UNALIAS	Remove alias
UNASSEMBLE	Unassemble memory
USE	Use different directory to search for source files
VARIABLE	Define a new variable
VERSION	Display Debug-56K software version and mode
VIEW	Open an ASCII text file for viewing
WAIT	Wait specified time
WATCH	Place variable in the watch window
?	Evaluate expression

ALIAS Define Custom Command String

Syntax

ALIAS [name | key [,"string"]]

Description

The ALIAS command is used to assign a name to a set of keystrokes, it is usually used to rename commands or to pack multiple commands into one command.

The ALIAS command generates an alias button on the tool-bar if name is preceded by the ! character. A large button can hold up to three characters, a small button can hold one character.

name — A name representing the alias. The name is optional. The ALIAS command generates a button for mouse usage if the name is preceded by !

key — A function key representing the alias The following is a list of the function keys supported:

F1..F12	F1 to F12
SF1..SF12	Shift-F1 to Shift-F12
AF1..AF12	Alt-F1 to Alt-F12
CF1..CF12	Ctrl-F1 to Ctrl-F12

"string" — String function to be executed when the alias is called. Multiple commands can be defined within one alias definition. Multiple commands are separated by a semicolon. Command parameters are identified by a circumflex sign followed by a number (^1, ^2, ...).
If "command string" is omitted, the alias definition for that name or key is displayed. If name, key, and command string are omitted, all defined aliases are displayed.

Example 1

```
ALIAS prgload "FORCE R; LOAD ^1; GO"
```
"prgload" resets the DSP, loads the specified file, and then runs the DSP

Example 2

```
ALIAS AF2
```
Display the alias definition of Alt-F2

Example 3

```
ALIAS
```
Display all alias definitions

Example 4

```
ALIAS !1st "change PC 0"
```
Generate a tool-bar button for the command "change PC 0"

4

ASSEMBLE On-Screen Assembler

Syntax	ASSEMBLE [address] [instruction]
Description	Assembles one DSP instruction.
addr	Address of the memory location to assemble. The address may be in the X: Y: or P: space. The address is optional. If the address is omitted, the current program counter value is used.
Instr	Instruction to be assembled. The instruction is optional, if the instruction is entered that instruction is assembled and the DSP's memory is updated. If the instruction is omitted, the unassemble window is selected and the cursor is placed at the specified address.

If the address and the instruction are omitted, the unassemble window is selected and the cursor is placed at the instruction equal to the program counter.

Example 1
```
A P:$100 ADD A,B
```
Assemble the instruction "ADD A,B" and place opcode in memory location P:$100.

Example 2
```
A
```
Assemble at the address equal to the program counter

Example 3
```
A START_PROGRAM
```
Assemble at address labeled START_PROGRAM

BREAK Breakpoint Operations

Syntax	BREAK [breakpoint_number] [OFF] [break_access] [break_type] [break_address] [break_block] [break_count] [expression] [S]
Description	The BREAK command sets, removes, and displays breakpoints. Many types of breakpoints are supported:

- Software breakpoints (unconditional)
- Software breakpoints (conditional)
- Software breakpoints (with expressions)
- Hardware breakpoint (with a count)
- Hardware breakpoint (without count)
- Hardware breakpoint (with a range)
- Hardware breakpoint (without a range)
- Hardware breakpoint (on read)
- Hardware breakpoint (on write)
- Hardware breakpoint (on read or write)

The simplest and most commonly used breakpoint is the unconditional software breakpoint, with the following simplified syntax.

BREAK	[breakpoint_number] [OFF] [break_address]
BREAK	with no arguments displays all breakpoints set.
	Parameters for software breakpoints:
brk#	breakpoint_number is a number between 1 and 128 for software breakpoints and 0 for the hardware breakpoint. This number is incremented automatically for software breakpoints, or it can be specified by the user.
OFF	Off removes the breakpoint at the breakpoint number. If a breakpoint number is not entered, all hardware and software breakpoints are removed.
S	Start the DSP automatically after the breakpoint is reached.
brkAddr	break_address is the address of the breakpoint. The address may be X:, Y:, P:, or a symbol.

Example 1	`BREAK P:$100`
	`Set a software breakpoint at program memory 100 hex`

Example 2	`BREAK START_PROGRAM` Set a software breakpoint on program memory instruction labeled START_PROGRAM
Example 3	`BREAK` Display all currently set breakpoints (hardware as well as software).
Example 4	`BREAK OFF` Remove all breakpoints
Example 5	`BREAK OFF 2` Remove breakpoint number 2

Parameters for hardware breakpoints:

[brk_acc] break_access is used to set a hardware breakpoint on memory read(r), write(w), or access(rw). Break access can not be used with software breakpoints. If a break access is specified, the breakpoint is automatically is treated as a hardware breakpoint. break_access applies only to break _types xa, ya, and pa.

Break_access:	**Meaning:**
R	break on memory read
w	break on memory write
rw	break on memory read or write

Brk_typ Break type determines the breakpoint type. There are 8 types of hardware breakpoints and 16 types of software breakpoints. The following are the hardware breakpoint types, they apply to the 24 bit DSPs only, please refer to the help screen for other DSPs.

Break_type:	**Meaning:**
pcf	Break on program memory fetch.
pcm	Break on program memory move.
pcfm	Break on program memory read/write.
pce	Break on executed fetches only.
pa	Break on P memory access
xa	Break on X memory access
ya	Break on Y memory access

The following are the 16 types of conditional software breakpoints. The software breakpoint types are based on the value of the Condition Code Register (CCR). To set a conditional software breakpoint, Debug-56K replaces your DSP instruction with a DEBUGcc instruction, thus the DSP program may be contaminated. break_type applies only to break_access r, w, and rw.

Type:	Meaning:	Condition code:
CC	carry clear	C=0
CS	carry set	C=1
EC	extension clear	E=0
EQ	equal	Z=1
ES	extension set	E=1
GE	greater or equal	N xor V = 0
GT	greater than	Z or (N xor V)=0
LC	limit clear	L=0
LE	Less or equal	Z or (N xor V)=1
LS	Lower or same	Z or C=1
LT	less than	N and V = 1
MI	minus	N = 1
NE	not equal	Z=0
NR	normalized	Z or (not N & not E)=1
PL	plus	N=0
NN	not normalized	Z or (not N and not E)=0

[brkCnt] Break count specifies the number of times the condition must be satisfied before a hardware breakpoint is reached. The break_count does not apply for software breakpoints. If the count is not specified, it defaults to one.

For 24-bit DSPs break_count may be greater than 0 and less than $1000000. For the DSP561xx break_count may be greater than 0 and less than $10000. For the DSP568xx break_count may be greater than 0 and less than $100.

[addrBl] address_block specifies an address range for hardware breakpoints. break_block can not be used with software breakpoints. The address range may be X: Y: P: or a symbol.

T[expr] Break expression allows the break at a breakpoint only if a condition is satisfied. The DSP is stopped, the expression is evaluated while the DSP is stopped. If the expression is not satisfied, the DSP is started again. If the expression is satisfied, the DSP stays halted. The format of T[expression] is as follows: T(val1?val2)

Val1 and Val2 can be one of following:

Immediate value (number preceded by “**#**”)
Register
Memory location

Tests supported (?): , =,   ==, !=

Spaces within test syntax are not permitted.

Other Breakpoint Rules:

1. The default breakpoint is a software breakpoint.
2. Only one hardware breakpoint may be set at a time.
3. Up to 128 software breakpoints may be set simultaneously.
4. You may use the function key F9 to set and clear unconditional software breakpoints.
5. You may use the mouse to set and clear unconditional software breakpoints.
6. You may use the pull-down menu to display, clear, save, or load breakpoints.

Example 6 `BREAK pcf PLAYABLE#$20`
Break on program memory fetch and execution of the instruction at range of instructions between PLAYABLE and PLAYABLE+32 (hardware breakpoint)

Example 7 `BREAK pcm P:LABEL_1..P:LABEL_2 7`
Break when program memory location between P:LABEL_1 and P:LABEL_2 is written to 7 times (hardware breakpoint).

Example 8 `BREAK pcm P:LABEL_1 7`
Break when program memory location P:LABEL_1 is accessed 7 times (hardware breakpoint).

Example 9 `BREAK rw xa X:100 200`
Break when data memory location 100 is accessed 200 times (hardware breakpoint).

Example 10 `BREAK P:$228 T(AM_VALUE#$1000)`
Set a breakpoint at address P:228 and enter debug mode if the value of AM_VALUE is greater than $1000 (software breakpoint).

Example 11 `BREAK LOOP T(X0!=Y:$1200)`
Set a breakpoint at program address labeled “LOOP” and enter debug mode if the value of register X0 is not equal to memory location Y:$1200 (software breakpoint).

CFORCE Reset the Emulator

Syntax <u>**CF**</u>ORCE R

Description Reset the emulator hardware unit. The CFORCE command is applicable only when Debug-56K is used with an external emulation unit.

Example 1

```
CFORCE R
```

Reset the emulator

CHANGE Change Register or Memory

Syntax CHANGE [register] | [register_block] | [address] | [address_block] (to) value

Description Change a register, a register block, a memory, or a memory block to value.

reg	A register
reg_blk	A block of registers
addr	A memory location
addr_blk	A block of memory
value	The change value

Example 1
```
CHANGE X:0 10
```
Change data memory location X:0 to 10

Example 2
```
CHANGE LABEL_1..LABEL_2 10
```
Change memory locations from LABEL_1 to LABEL_2 to 10

Example 3
```
CHANGE X:0#100 1234
```
Change 100 data memory locations starting at address X:0 to 1234

Example 4
```
CHANGE A2 10
```
Change register A2 to 10

Example 5
```
CHANGE R0..R3 10
```
Change register R0, R1, R2, and R3 to 10

CONFIG Set System Configuration Options

Syntax	CONFIG Refresh n | Tab n | Reset {On|Off} | Sens {On|Off} | Verify {On|Off} | Update {On|Off}
Description	The Config command is used to set the system options. The CONFIG command is especially helpful when used with macro command files.
Refresh:	Sets the minimum time delay (in ms) between screen refreshes in the continuous update mode.
Tab:	Sets the number of spaces for a tab character. This applies when displaying ASCII text files in the view window.
Reset:	When set, the DSP is automatically reset before a program load. The default is off.
Sens:	Enables or disables case sensitivity for symbols entry. The default is on.
Verify:	Enables or disables memory verify after memory writes. Default is off.
Update:	Enables or disables the data and registers windows updates when the DSP is running. Default is OFF.

Example 1

```
CONFIG ref ‘2000
```

Sets refresh interval time to 2000 ms (2 s)

Example 2

```
CONFIG update on
```

Turn automatic update ON.

COPY — Copy a Memory Block

Syntax	COPY (from) address_block (to) address
Description	Copy a block of memory.
addrBlk	The block of memory to be copied.
addr	The starting address of the second block.

Example 1

```
COPY X:0..100 P:0
Copy data memory block X:0 to X:100 into program
memory starting at p:0
```

Example 2

```
COPY LABEL_1..LABEL_2 LABEL_3
```

Copy memory block LABEL_1 to LABEL_2 into memory block starting at LABEL_3.

Example 3

```
COPY X:0#100 X:200
```

Copy 100 memory locations starting at data memory location 0 into data memory location starting at 200.

DISSASEMBLE Disassemble Memory

Syntax <u>DI</u>SASSEMBLE [address] [-mode]

Description Disassembles memory (program or data) in the unassemble window.

addr — The starting address. If address is omitted, the window is scrolled down by one page.

-mode — mode specifies the program display mode. Three modes are available are:

Mode:	**Description:**
Asm	Reverse assembly mode
Src	Source mode
Mix	Mixed mode (ASM and SRC interlaced)

If the mode is omitted, the current mode is used.

Example 1
```
DISASSEMBLE LABEL_1 -mix
```
Disassemble from memory location LABEL_1 in mixed mode

Example 2
```
DISASSEMBLE X:100
```
Disassemble from memory location X:100

Example 3
```
DI P:100
```
Disassemble from memory location P:100

Example 4
```
DI
```
Scroll the unassemble window down by one page

DISPLAY Display Memory

Syntax DISPLAY [address] [-radix] [-mode]

Description The DISPLAY command displays program or data memory in a data window. If more than one data window is open, the data is displayed in the window that was last selected.

addr — The starting address of the window. If [address] is not specified, the window is scrolled down by one page.

-radix — Specifies the data display radix. The valid radices are:

Radix:	**Description:**
-d	Decimal
-h	Hexadecimal
-b	Binary
-f	Fractional
-a	ASCII

If the radix is omitted, the current radix is used.

[-mode] — Specifies the data display mode. The valid modes are:

Mode:	**Description:**
-t	Text
-g	Graph

If the mode is omitted, the current mode is used.

Example 1
```
DISPLAY LABEL_1
```
Display from memory location LABEL_1

Example 2
```
DISPLAY L:100 -b
```
Display from memory location L:100 in binary format

Example 3
```
DISPLAY P:100 -d -g
```
Display from memory location P:100 in graphics with a decimal axis.

EMI Set EMI Parameters

Syntax	EMI [-Wx] [-Lx]
Description	The EMI command displays or sets the External Memory Interface access parameters. This command is applicable for DSPs with dynamic memory support (DSP56004, DSP56007, etc.)
-Wx	Set the EMI data bus width. Two selections are available: 4 bit data bus: -w0 8 bit data bus: -w1
-Lx	Set the EMI data word length. Four options are available: 8 bit data word: -L0 16 bit data word: -L1 24 bit data word: -L2 16 bit data word, 24 bit address: -L3

EVALUATE Evaluate Expression

Syntax	EVALUATE ([lvalue =] rvalue)
Description	The EVALUATE command is used to evaluate expressions or to set a variable, a memory location, or a register. This command may be used with C and assembly language variables and constants.
lvalue =	Left Value. The left value and the equal sign are optional. They are used when a variable or a memory location are set.
rvalue	Right Value. The right value is the expression to evaluate.

The EVALUATE command is similar to the ? command, except the ? command displays the result in the command window.

Example 1
```
EVALUATE R0 = $100
```
Set register R0 to 100 hex

Example 2
```
EVALUATE R2 = (R0+R1) * 3
```
Evaluate the expression, set R2 to the result

Example 3
```
EVALUATE value_1 = value_2/value_3
```
Evaluate the expression, set value_1 to the result

Example 4
```
EVALUATE FILTER[100] = 0
```
Clear the 101's element of the array FILTER

FORCE Reset or Break DSP

Syntax	<u>F</u>ORCE R | B | RU
Description	Force the DSP
R	Reset DSP and enter debug mode
B	Stop DSP and enter debug mode
RU	Toggle the DSP’s reset line

Example 1 `FORCE R`
Reset DSP and enter debug mode

Example 2 `FORCE B`
Stop DSP and enter debug mode

Example 3 `FORCE through`
Toggle the DSP’s reset line

GO Execute DSP Program

Syntax	<u>G</u>O [address]
Description	The GO command runs the DSP.
addr	Starting address. If [address] is omitted, the program executes from the current program counter. If the address is specified, the program counter is changed to the address by the debugger before the DSP is started.
Example 1	`GO` Start DSP.
Example 2	`GO LABEL_1` Start DSP from program memory location LABEL_1.
Example 3	`GO P:$100` Start DSP from program memory location 100 hex.

HELP Display Help Screen

Syntax	**H**ELP [command]
Description	Start Help.
comm	if [command] specified, help is displayed for the specific command.
Example 1	`HELP` Display the help screen.
Example 2	`HELP BREAK` Display the help for the BREAK command.

4

INPUT Assign Input File

Syntax	INPUT [#(file_number)] [address] OFF | TERM | filename [-dec | -fra | -hex]
Description	The input command is used to open a text file and use the file to pass data to the target DSP chip. The data is passed to the target DSP when the user's program reaches a software breakpoint. The text file lists the data sequentially.

The INPUT command with no parameters displays all open input files.

file#	file_number is the number of the file opened (multiple files can be opened simultaneously). The file number is optional, the range is 1 to 99.
addr	address is the address where the DSP breakpoint is halted in order to perform the file input (always in the P: space). Assigning the address of the DEBUG instruction in INPUT command defines direction of the data transfer when user program reaches DEBUG opcode. The address is optional if one of the direction bits in register R1 is set.
OFF	closes an opened input file. If a file is not specified, all opened files are closed.
TERM	uses the terminal/keyboard (instead of a text file) to input data to the target DSP.
fname	is the name of the file used for data input.
-dec	specifies decimal data representation
-fra	specifies fractional data representation
-hex	specifies hexadecimal data representation. This is the default.

Parameters specified by the user's DSP program.

Since multiple files can be opened, your DSP program must specify the file number, this is accomplished by placing the file number in the most significant 8 bits of register X0.

The user program must specify the number of words to transfer, this is accomplished by placing that number in the low 16 bits of register X0. The low 8 bits of register X0 are used for the DSP56100.

The user program must also specify the address where the data will be placed. This is accomplished by placing that address in register R0.

The user program must specify the address space (P:, X: or Y:) where the data will be placed. This is accomplished by placing a value in the register R1. A 0 is used for P:, 1 for X: and 2 for Y: memory space.

The direction of the data transfer may be specified (optional) by placing a flag in register R1. If the most significant bit (8000 hex) is 1, the direction is into the DSP. If the 14th bit (4000 hex) of R1 is 1, direction is out of the DSP. If neither bits are set, the address of the DEBUG instruction is used.

The breakpoint must reached by a user induced DEBUG instruction.

Details on the text file

The text file is always ASCII, with the data listed sequentially. Debug-56K provides ways to simplify the file editing (for example if one word must be send to the DSP repeatedly, you can have a repeat code instead of typing an infinitely long file.

The following are the codes that may be used in the ASCII file

#	Specifies the number of times a data word is repeated.
()	Parenthesis are used to group words. If parenthesis are preceded by **#** the whole group is repeated. If **#** doest not precede the parenthesis, the group of words is repeated indefinitely.

Example 1
```
0.1  0.2  0.3  0.4
```
Send data words 0.1, 0.2, 0.3, 0.4.

Example 2
```
0.1#10
```
Send the data words 0.1 10 times.

Example 3
```
0.1 0.2)#10
```
Send 0.1, 0.2, 0.1, 0.2 ....

Example 4
```
7fff)a
```
Send 7fff indefinitely

Example 5
```
1 2 3 5#2 (10 20)#3 (10)
```
send 1, 2, 3, 5, 5, 10, 20, 10, 20, 10, 20, 10, 10, 10, 10, ...

Examples on entering the input command

Example 1
```
INPUT
```
Display all currently open input files.

Example 2
```
INPUT #1 P:100 DATA.IO -dec
```
Open “DATA.IN” file, label it file number 1, and send the data to the DSP when the breakpoint at address P:100 is reached. Data in “DATA.IO” is in decimal.

Example 3
```
INPUT P:100 DATA.IN2
```
Open “DATA.IN2" file, label the file automatically, and send the data to the DSP when the breakpoint at address P:100 is reached. Data in ”DATA.IN" is in hexadecimal.

Example 4
```
INPUT P:100 TERM
```
When the breakpoint at P:100 is reached, read data from the terminal (keyboard) and send it to the DSP.

Examples of DSP code to support the INPUT command

Example 1

```
MOVE        $010010,X0  ; 16 WORDS FROM FILE NUMBER 1
MOVE        $0000,R0    ; PLACE DATA AT ADDRESS 0
MOVE        $0001,R1    ; PLACE THE DATA IN SPACE X:
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

Example 2

```
MOVE        $020012,X0  ; 18 WORDS FROM FILE NUMBER 2
MOVE        $0020,R0    ; PLACE DATA AT ADDRESS 20
MOVE        $0000,R1    ; PLACE THE DATA IN SPACE P:
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

Example 3

```
MOVE        $030080,X0  ; 128 WORDS FROM FILE NUMBER 3
MOVE        $0200,R0    ; PLACE DATA AT ADDRESS 200 HEX
MOVE        $8002,R1    ; PLACE THE DATA IN SPACE Y:
                        ; (INPUT DIRECTION)
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

JUMP — Jump Over Subroutine Calls

Syntax <u>J</u>UMP

Description JUMP instructs the DSP to execute one instruction while treating a subroutine as one instruction. This is similar to single-step with the exception that when a subroutine call is reached the whole subroutine is executed.

If the DSP reaches a breakpoint inside the subroutine, it will enter the debug mode before completing the subroutine call.

Example 1

```
JUMP
```

Execute jump.

LOAD — Load DSP Program

Syntax	<u>L</u>OAD [filename] [.CLD] [.LOD] [-S]
Description	The LOAD command loads a program file generated by the Motorola assembler/linker into the DSP's memory. The DSP memory can be internal or external. The memory space can be X: Y: P: or L: The file is read from the default directory (see the PATH and USE commands).
	To enable symbolic debugging and source level debugging, the switch -g must be entered when executing Motorola's compilation tools. If the symbols are not found, the debugger enters the non-symbolic mode of operation. If the source line information is not found, the debugger enters the non-soure level mode of operation.

.CLD	COFF filename extension.
.LOD	OMF filename extension.
.ABS	IEEE-695 filename extension.
-S	Load the symbols only.

Example 1

```
LOAD DEMO1.LOD
```

Load the file DEMO1.LOD (OMF format) from the default path.

Example 2

```
LOAD DEMO3.CLD
```

Load the file DEMO3.CLD (COFF format) from the default path.

Example 3

```
LOAD C:\PROGRAMS\DEMO3.CLD -s
```

Load the symbols from C:\PROGRAMS\DEMO3.CLD.

LOG Log Information

Syntax	LOG ON [filename[.LOG]] | OFF | memAddress_block | UNA_address_block | REG [reg_name] ONCE
Description	The LOG command is used to log information to the log file.
ON fname	Opens a log file. If the file already exists, information is appended to the file. The current date and time are written to the log file after it is opened. Default file extension is .LOG
OFF	Closes the log file. The current date and time is written to the file before it is closed.
memAddrBlk	Memory block to save to the file.
REG[reg_name]	REG saves all registers to the file. If a register name is specified, only that register is saved.
UNA_addrBlk	Block of instructions to save to the file.
ONCE	Save the OnCE registers to the file.

Example 1

```
LOG ON session5
```

Start logging to file SESSION5.LOG

Example 2

```
LOG MEM X:$40#$20
```

Save memory locations X:40 to x:60 to log file

Example 3

```
LOG REG
```

Save registers to log file

Example 4

```
LOG REG R0
```

Save register R0 to log file to log file

Example 5

```
LOG OFF
```

Close the log file

OUTPUT Assign Output File

Syntax **O**UTPUT [**#**(file number)] [address] OFF | TERM | filename [-dec | -fra | -hex] [-Ovr] [-col_cnt]

Description The output command is used to open a text file and use the file to save data passed from the target DSP chip. The data is passed to the file when the user's DSP program reaches a software breakpoint.

The OUTPUT command with no parameters displays all open output files.

file**#**	is the number of a file opened (multiple files can be opened simultaneously). The file number is optional, the range is 1 to 99. The ASCII text file lists the data sequentially.
addr	is the address of the debug instruction (always in the P: space). The address is optional if one of the direction bits in register R1 is set.
OFF	closes an opened output file. If a file is not specified, all opened files are closed.
TERM	uses the terminal/monitor (instead of a text file), and display the data.
fname	is the name of the file used for data output.
-dec	specifies decimal data representation
-fra	specifies fractional data representation
-hex	specifies hexadecimal data representation. This is the default.
-Ovr	Over-write file if the file exists.
-col_cnt	Column count

Parameters specified by the user's DSP program.

Since multiple files can be opened, your DSP program must specify the file number, this is accomplished by placing the file number in the most significant 8 bits of register X0.

The user program must specify the number of words to transfer, this is accomplished by placing that number in the low 16 bits of register X0. 8 bits are used for the DSP56100.

The user program must also specify the address where the data will be read from. This is accomplished by placing that address in register R0. The user program must specify the address space (P:, X: or Y:) where the data will be placed. This is accomplished by placing a value in the register R1. A 0 is used for P:, 1 for X: and 2 for Y: memory space.

The direction of the data transfer may be specified (optional) by placing a flag in register R1. If the most significant bit (8000 hex) is 1, the direction is into the DSP. If the 14th bit (4000 hex) of R1 is 1, direction is out of the DSP. If neither bits are set, the address of the DEBUG instruction is used.

The breakpoint must reached by a user induced DEBUG instruction.

Details on the text file

The text file is always ASCII, with the data listed sequentially, in the format specified but the OUTPUT command.

Examples on entering the input command

Example 1 `OUTPUT`

Display all currently open output files.

Example 2 `OUTPUT #1 P:100 DATA.OUT -rd`

Open “DATA.OUT” file, label it file number 1, and use the file to save data when the breakpoint at address P:100 is reached. Data is saved in decimal.

Example 3 `OUTPUT P:100 DATA.OUT`

Open “DATA.OUT” file, label the file automatically, and send the data to the file when the breakpoint at address P:100 is reached. Data in “DATA.OUT” is in hexadecimal.

Example 4 `OUTPUT P:100 TERM`

When the breakpoint at P:100 is reached, read data from the DSP and send it to the terminal (screen).

Examples DSP code to support the OUTPUT command

Example 1

```
MOVE        $010010,X0  ; 16 WORDS to FILE NUMBER 1
MOVE        $0000,R0    ; READ DATA FROM ADDRESS 0
MOVE        $0001,R1    ; READ DATA FROM SPACE X:
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

Example 2

```
MOVE        $020012,X0  ; 18 WORDS to FILE NUMBER 2
MOVE        $0020,R0    ; READ DATA AT ADDRESS $20
MOVE        $0000,R1    ; READ DATA FROM SPACE P:
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

Example 3

```
MOVE        $030080,X0  ; 128 WORDS to FILE NUMBER 3
MOVE        $0200,R0    ; READ DATA AT ADDRESS $200
MOVE        $4002,R1    ; READ DATA FROM SPACE Y:
                        ; (OUTPUT DIRECTION)
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

PATH — Define directory path

Syntax	PATH [pathname]
Description	The PATH command sets the default directory for file read and file write operations.
pathname	String defining the default directory. If pathname is omitted, the current path is displayed (if it was previously set). See also the USE command.
Example 1	`PATH D:\56002\PROGRAMS\` SET the default directory to D:\56002\PROGRAMS\
Example 2	`PATH` Display the default directory path

QUIT — Quit Debug-56K

Syntax	**Q**UIT
Description	The QUIT command terminates Debug-56K. When QUIT is executed, the DSP is left in it's current state and the screen configuration is automatically saved in the screen configuration file. Upon reentry to Debug-56K, the screen configuration is read automatically from the screen configuration file.
Example 1	`QUIT` Quit Debug-56K

RADIX — Change Radix of Command Window

Syntax RADIX B(binary) | D(decimal) | F(fractional) | H(hexadecimal)

Description The RADIX command changes the default radix of the command window. The following are the available options:

Radix:	**Description:**
b	Binary
d	Decimal
f	fractional
h	Hexadecimal

All constants (data and addresses) entered in radix may be overwritten if one of the following characters precedes a number:

$	for hexadecimal numbers (example $7fffff)
‘	for decimal numbers (example ‘5000)
%	for binary numbers (example %10101011)

The RADIX command can not be used to change the radix of a data or a register window.

Example 1

```
RADIX H
```

Set command window’s default radix to hexadecimal

REFRESH Toggle Screen Refresh On/Off

Syntax	**REF**RESH [ON | OFF]
Description	In normal operation, the debugger's screen is refreshed when the DSP is halted. Screen refreshes can be disabled with the refresh command. Disabling screen refreshes may be helpful with slow systems with high screen refresh times.
ON	Enable screen refreshes
OFF	Disable screen refreshes

The refresh command without On or Off refreshes the screen.

Upon a refresh, the DSP's resources are uploaded by the debugger and displayed on the screen.

Example 1

```
REFRESH OFF
```

Disable screen refreshes

Example 2

```
REFRESH ON
```

Enable screen refreshes

Example 3

```
REFRESH
```

Refresh screen

RETURN — Execute Until RETURN Instruction

Syntax **RE**TURN

Description Run the DSP until a RETURN instruction is reached. The DSP is stopped when a RET instruction is reached.

Example 1

```
RETURN
```

Execute DSP, stop when a RET instruction is reached.

SAVE — Save DSP Memory to File

Syntax	**S**AVE addr_block [addr_block] [addr_block] ...filename[.LOD] [-hex | -dec | -fra | -bin] [-App | -Ovr] [-col_cnt]

Description

The SAVE command saves one or more blocks of memory into a text file. The file format is compatible with files generated by the Motorola assembler/linker. Program as well as data memory can be saved. Memory can be internal or external to the DSP.

If the file directory is not specified, the file is read from the default directory (see the PATH and USE commands). If the file extension is not specified, it is automatically assumed to be .LOD

Symbols are not saved with the SAVE command.

addrBlck	Address block
addrBlk	Additional address block(s)
fname	File name, default extension is .LOD
-App|-Ovr	Append to file, over-write if file exists
-col_cnt	Column count

Example 1

```
SAVE X:0..90 DATA.001
```

Save memory locations X:0 to X:90 into file “DATA.001"

Example 2

```
SAVE X:0..90 P:0..100 DATA1
```

Save memory locations X:0 to x:90 and memory locations P:0 to P:100 into file “DATA1.OUT”

STEP — Step DSP Program

Synatx	**ST**EP [count]
Description	The STEP command instructs the DSP to execute a number of instructions and then enter the debug mode. The instruction(s) execute from the current value of the program counter.
count	Number of DSP instructions to execute. If [count] is not specified, one instruction is executed.
Example 1	`STEP` Execute one DSP instruction and enter the debug mode.
Example 2	`STEP 5` Execute 5 DSP instructions and enter the debug mode.

SYMBOL Display Symbol Table

Syntax	**<u>SYM</u>**BOL [symbol or part of symbol name]
Description	The SYMBOL command displays the symbols. The symbols are displayed in alphabetic order. Symbols are case sensitive unless the case sensitivity is turned off.
symbol	If omitted, all the symbols are displayed. If specified, only the symbols starting with [symbol or part of symbol name] are displayed. It will match all the symbols starting with the specified string
Example 1	`SYMBOL` Display all symbols
Example 2	`SYMBOL La` Display all symbols starting with La

TIME Display Execution Time

Syntax **TI**ME

Description

The TIME command can be used to measure the execution time of a DSP program. A timer is automatically cleared and started when the GO command is executed, the same timer is automatically stopped when the DSP is stopped. The TIME command displays the value of that counter. The resolution of the timer is 1 µs. For proper timer operation, the program should be run for more than 50 µs.

The TIME command is available when the debugger is used with Domain Technologies' LINK-56K emulator. The TIME command may not be supported under other hardware platforms.

Example 1

```
TIME
```

Display the timer value

TRACE — Trace Through DSP Program

Syntax	TRACE [count]
Description	The TRACE command executes a single or multiple DSP instructions with a full screen refresh after every instruction.
count	Number of instructions to execute. If [count] is omitted, one instruction is executed.
Example 1	`TRACE` Execute one DSP instruction and update screen
Example 2	`TRACE 10` Execute 10 DSP instructions with a screen update after every instruction

UNALIAS Remove Custom Command String

Syntax	<u>**UN**</u>ALIAS name | Key
Description	The UNALIAS command removes a previously defined alias. Aliases are defined with he ALIAS command.
name	Name of the alias to remove.
key	Function key of the alias to remove.
	Either name or key must be defined.
Example 1	`UNALIAS prgload`
	Remove “PRGLOAD” alias
Example 1	`UNALIAS F1`
	Remove alias at function key F1

UNASSEMBLE Unassemble Memory

Syntax UNSASSEMBLE [address] [-mode]

Description Unassemble memory in the unassemble window.

addr — Address of the first memory location to unassemble. The address may be in the X, Y, or P space. The address is optional. If the address is omitted, the unassemble window is scrolled down by one page.

-mode — mode specifies the program display mode. Three modes available are:

Mode:	**Description:**
ASM	Reverse assembly mode
SRC	Source mode
MIX	Mixed mode (ASM and SRC interlaced)

If the mode is omitted, the current mode is used.

Example 1

```
UNASSEMBLE LABEL_1
```

Disassemble from memory location LABEL_1

Example 2

```
UNASSEMBLE
```

Scroll the Unassemble window down by one page

USE Use Different Directory

Syntax	<u>**US**</u>E [path string [-r]]
Description	The USE command specifies an additional directory to be searched to locate the source files. Multiple directories can be specified. If no parameters are specified, all defined directories will be listed.
-r	will remove the directory from the search list.
Example 1	`USE d:\dsp\source\coeff` Use additional directory while locating source file

VARIABLE Define a New Variable

Syntax	VARIABLE name
Description	The variable command defines a variable at the debugger level. The variable defined may then be used for temporary storage.
name	The name of the variable defined

Example 1

```
VARIABLE PC_STORAGE   ; define a new variable
? PC_STORAGE = PC     ; save PC value
GO                    ; start DSP
FORCE R               ; stop DSP
? PC = PC_STORAGE     ; restore PC value
```

VERSION Display Debug-56K's version

Syntax

<u>**VE**</u>RSION

Display the Debug-56K version number.

Example 1

```
VER
```

Will display the Debug-56K version number.

VIEW Open an ASCII Text File

Syntax	VIEW OFF | filename [line **#**]
Description	Display an ASCII file in the view window.
OFF	Remove the viewed file
fname	Name of the file to display. If a directory is not specified within the file name, the default directory is used (see the PATH and USE commands).
line **#**	Display from line number. If not specified, the file is displayed from the first line.

Example 1

```
VIEW DEMO1.ASM 10
```

Display the file DEMO1.ASM in the view window starting at line 10.

WAIT — Wait Specified Time

Syntax	**WA**IT [time | BREAK]
Description	Wait a number of seconds. This command is helpful when executing macro command files. The wait can be aborted by pressing any key.
time	Wait time in seconds. If time is omitted, the wait is indefinite.
BREAK	Wait until the DSP gets in the debug mode of operation.

Example 1

```
WAIT 12
```

Wait for 12 seconds.

Example 2

```
WAIT break
```

Wait until the DSP enters debug mode.

WATCH Place Variable in Watch Window

Syntax	WATCH expression, [label], [-mode] [-r]
Description	The WATCH command places a variable in the watch window.
expr	Expression defining the locations to place in the watch window.
label	Label of the variable to watch
mode	The mode specifies the radix used for the display. Available formats are: hex, dec, fra, bin. The default is hex.
-r	Is used to remove the variable from the watch window. If [address] is not specified, all the watched variables are removed.

Example 1

```
WATCH X:$0000
```

Place memory location at X:0000 in the watch window.

Example 2

```
WATCH AM_VALUE -fra
```

Place memory location at AM_VALUE in the watch window, display in fractional.

Example 3

```
WATCH P:$0000 -r
```

Remove P:$0000 from the watch window.

Example 3

```
WATCH -r
```

Remove all watched variables.

? Evaluate Expression

Syntax	? ([lvalue =] rvalue)
Description	The ? command is used to evaluate expressions. The result of the expression evaluation is displayed in the command window. The ? command can also be used to change the value of a variable, a memory location, or a register. The ? command may be used with C and assembly language variables and constants.
lvalue =	Left Value. The left value and the equal sign are optional, they are used when a variable or a memory location are modified.
rvalue	Right Value. The right value is the expression to evaluate.
	The ? command is similar to the EVALUATE command, except the EVALUATE command does not display the evaluated value.

Example 1

```
? R0 = $100
```

Set register R0 to 100 hex, display new R0

Example 2

```
? A
```

Display the value of register A

Example 3

```
? R2 = (R0+R1) * 3
```

Evaluate the expression, set R2 to the result, display new R2

Example 4

```
? value_1 = value_2/value_3
```

Evaluate the expression, set value_1 to the result, display new value_1

Example 5

```
? FILTER[100] = 0
```

Clear the 101's element of the array FILTER, display new 101's element of array FILTER.

Index