68020 PROBE/3

20665 FOURTH STREET * SARATOGA, CA 95070 * (408) 741-5900

TABLE OF CONTENTS

CHAPTER 1. INSTALLING THE 68020 PROBE/3

CHAPTER 2. THE PROBE USER INTERFACE

CHAPTER 3. GENERATING AND USING SYMBOLS

CHAPTER 4. PROBE TUTORIAL EXAMPLE

CHAPTER 5. COMMAND REFERENCE

APPENDICES

FIGURES

TABLES

INTRODUCTION

Thank you for your purchase of the 68020 PROBE/3 ATRON'S 68020 PROBES are a family of debugging tools for the 68020 microprocessor. The PROBES are modular in design and provide a wide range of debugging capabilities.

This version of PROBE is an in-circuit emulator. It consists of the PROBE SOFTWARE which executes on an IBM AT, a POD which plugs into the 68020 socket, and Breakpoint/Trace boards which plug into the AT and connect to the POD.

THE POD

The 68020 POD comes in three versions -16 for 16 mhz systems, -20 for 20 mhz and -25 for 25 mhz systems. The following features are implemented in the POD logic:

1. Hardware and software execution breakpoints (all other hardware breakpoints come from Breakpoint/Trace boards.
2. Guarded memory access
3. 4 channel logic analyzer input
4. Breakpoint detect output trigger
5. Interface to Breakpoint/Trace boards
6. Interface to MAP RAM boards
7. High speed (375k baud) serial link

BREAKPOINT/TRACE BOARDS

There are two Breakpoint/Trace boards. The first board supports processor data buses up to 16 bits wide and address buses up to 24 bits wide. The second Breakpoint/Trace board extends the first by adding 16 additional data bus lines and 8 more address lines. Both boards are needed for the 68020 PROBE. These boards plug into the IBM AT and connect to the POD via ribbon cables.

This manual describes the 68020 PROBE\3 and its upgraded software option 68020 SOURCE PROBE. The standard PROBE software has symbolic debugging capabilities. The SOURCE PROBE option adds source level debugging features to the symbolic debugging capability of PROBE.

MAP RAM

The MAP RAM option lets you add 1/2 megabyte of memory which you can map over the target system.

ORGANIZATION OF THIS MANUAL

CHAPTER 1 contains the PROBE installation procedures.

CHAPTER 2 describes the PROBE user interface and the start up of PROBE.

CHAPTER 3 describes the symbolic and source level debugging information that can be used in PROBE commands.

CHAPTER 4 contains a tutorial example an provides an introduction to PROBE commands.

CHAPTER 5 is the Command Reference section which contains definitions and examples for each PROBE command.

APPENDICES contain additional information about the PROBE. Also included is a useful group of technical reports regarding common issues that arise during debugging.

INDEX indicates where to look for information based on key words or concepts.

CHAPTER 1 INSTALLING THE PROBE

INTRODUCTION

This chapter contains the set up and installation procedures for the 68020 PROBE/3. There are six steps in the installation process.

1. Unpacking the PROBE
2. Installing the PROBE software
3. Installing the Breakpoint/Trace boards
4. Installing the MAP RAM board
5. Reviewing PROBE'S configuration file
6. Running the PROBE diagnostics

UNPACKING THE PROBE

Carefully unpack your PROBE, and inspect all parts for damage. If they are damaged, please contact Atron as well as the carrier used to ship the PROBE. The PROBE/3 package should contain the following components:

1. Master Breakpoint/Trace board - Fig 1-1
2. Slave Breakpoint/Trace board - Fig 1-2
3. 68020 POD Fig 1-3
4. PROBE floppy diskettes
5. Demo board Fig 1-4 with power supply cable
6. In addition, some configurations include a MAP RAM board which may be located inside the POD Fig 1-5.
7. Some systems may include the Logic Probes Fig 1-6.

INSTALLING THE PROBE SOFTWARE

It is recommended that you make a backup copy of your PROBE floppy diskettes or copy them to your hard disk. To copy the diskettes to a hard disk, first make a private directory:

MKDIR \PROBE

Change the directory to PROBE:

CD \PROBE

Now insert each floppy diskette into drive A and copy them to the hard disk:

COPY A:*.*

If you get an error during this process, refer to the DOS manual which came with your PC/AT.

PROBE'S CONFIGURATION FILE

The 68020 PROBE Breakpoint/Trace and MAP RAM boards use 16 kbytes of address space from the AT memory space. The base address of this 16k byte memory block is set via software from a configuration file named PROBE.CNF when the PROBE software is invoked (see Chapter 2 pg 3). The default base address in this file is D0000. This does not need to be changed unless it conflicts with address space used by other AT boards. This space is not reserved and does not conflict with known IBM boards or common peripheral equipment.

The file PROBE.CNF contains simple ASCII text and can be changed with a common text editor. This file also contains configuration parameters which are described in Appendix C.

INSTALLING THE BREAKPOINT/TRACE BOARDS

First plug the Slave Breakpoint/Trace board (Fig 1-2) into the IBM AT or compatible. This board can be plugged into any slot but it is recommended that you use the slot to the left side (when facing the front of the computer). This slot has only one AT motherboard connector and the Slave Breakpoint/Trace board requires only one connector. The Master Breakpoint/Trace board (Fig 1-1) should be plugged adjacent to the Slave into an AT card slot which has 2 motherboard. A 34 pin ribbon cable comes attached to the Slave board. Connect this cable to the adjacent Master board.

The POD (Fig 1-3) is connected to the two Breakpoint/Trace boards through two 6 ft. ribbon cable assemblies. The ends of the Slave Breakpoint/Trace board is color coded red. Match the cable with the red lined connector to this board. The other cable and Master Breakpoint/Trace board are not colored.

In this manual, the plug at the end of the POD into which the 68020 is plugged is called the Buffer Assembly. Plug the Buffer Assembly into the target system. Note that one extra pin grid array socket is included on the Buffer Assembly. If pins break off this socket, replace it with a new socket. Note that this socket is polarized to prevent plugging into the target incorrectly.

INSTALLING THE MAP RAM BOARD

Open the POD chassis by removing the four screws that hold on the rubber feet on the bottom. Install the MAP RAM board by plugging into the two 64 pin connectors as shown in Fig 1-5. Insure that the MAP RAM board is fully seated on these connectors. Next reassemble the POD chassis. Note, if you purchased your MAP RAM board with the PROBE/3, this step has already been done by Atron.

INSTALLING THE LOGIC PROBES

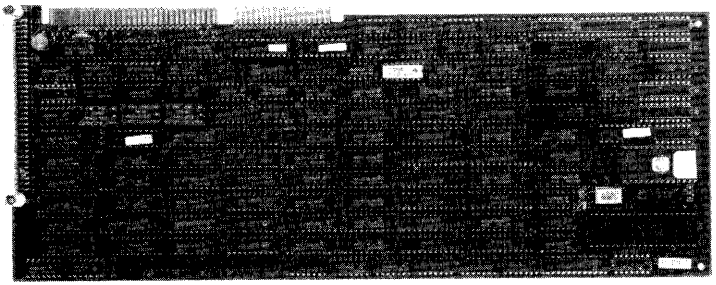
Open the POD chassis by removing the four screws that hold on the rubber feet on the bottom. Plug the Logic Probe cable into the connector labeled PL18 as shown in Fig 1-5. Insert the Logic PROBES such that the Brown wire connects to the pin labeled 1. Route the Logic Probes out the side of the POD between the bottom plate and the top of the chassis. Next, reassemble the POD chassis. See Appendix H for more information on the Logic Probes.

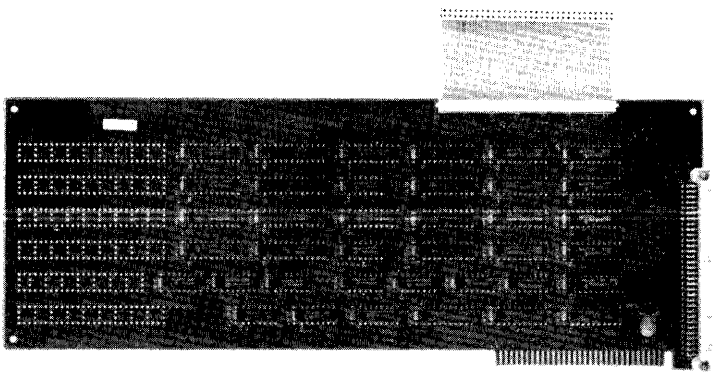
USING THE ATRON DEMO BOARD

The Demo Board contains a simple 68020 design including clock and 2k bytes of static ram memory starting at address 0. This board has two purposes. First it can be used in conjunction with the PROBE Diagnostic Confidence test (CONF020) to test the PROBE logic out through the end of the plug. Second, if you don't have any hardware to start with, this board could provide a simple execution vehicle for starting your software development. If more ram is needed, add the PROBE RAM RAM board.

The Demo Board must be supplied with +5 volts. A power supply cable is provided for this purpose. Connect the red wire to +5v. Connect the black wire to ground. Connect the end with the plug into the Demo Board - note that the polarity should be correct.

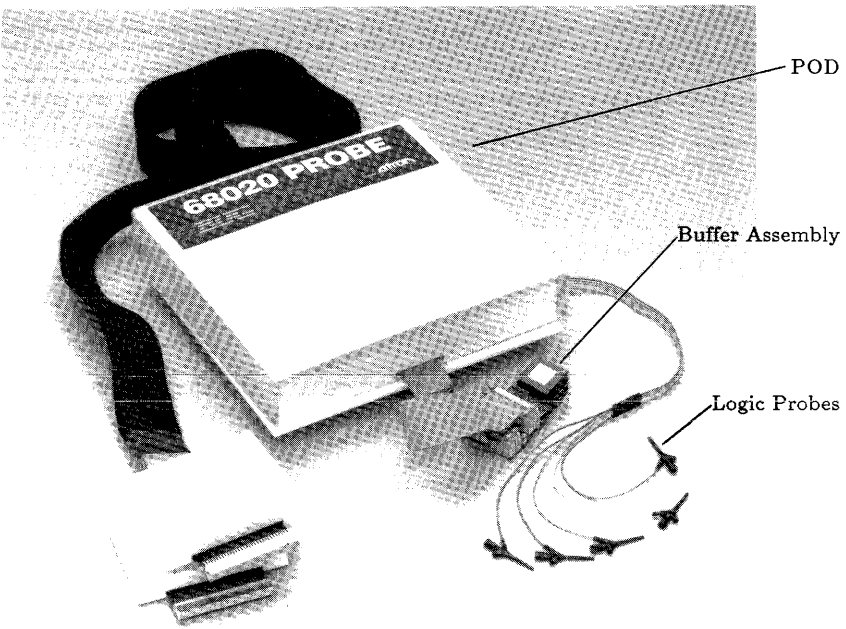
Master Breakpoint/Trace board
Fig 1-1

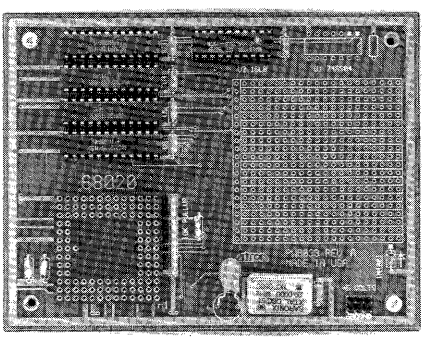




Slave Breakpoint/Trace board
Fig 1-2

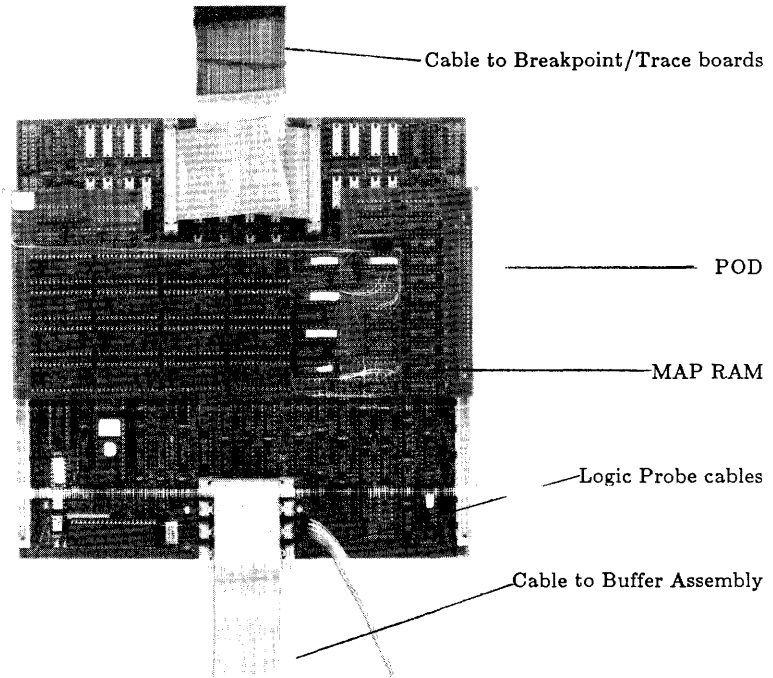
68020 POD
Fig 1-3

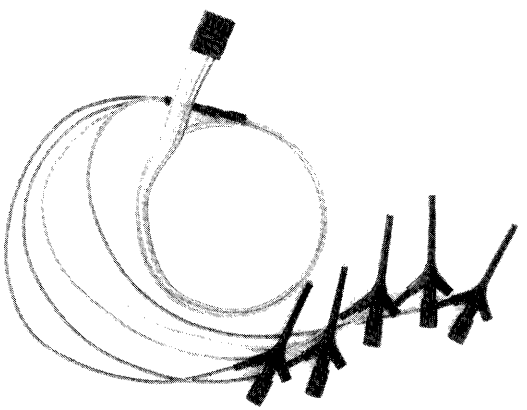




Demo Board
Fig 1-4

MAP RAM BOARD in POD
Fig 1-5





Logic Probes
Fig 1-6

RUNNING 68020 PROBE DIAGNOSTICS

The CONF020.EXE is an extensive set of diagnostics used to perform a confidence test on the 68020 Probe. To start the tests type:

CONF020

You will then receive the following prompts:

Is this a special I/O port addressed set of boards (Y/N)

The answer to this prompt should always be NO unless you have received a special set of Breakpoint/Trace boards from Atron which initialize the boards in the AT's IO space rather than the AT's memory space. See Appendix C for details.

Enter break/trace boards base address (C4 for C4000, CC, [D0], D4, D8, DC):

This prompt requests the address within the AT for accessing the ATRON break/trace boards. D0000 is used as the default address if only <Enter> is typed. This address should match the address in the PROBE.CNF file when running the normal PROBE software. If no PROBE.CNF file is present, then the default base address is used.

Is 68020 buffer plugged into the ATRON DEMO BOARD (Y or [N]):

The confidence test will run in one of two modes:

DEMO BOARD: The 68020 buffer assembly is connected to the ATRON DEMO board. This mode will test target system access as well as performing internal diagnostics. Appendix L describes the details of these accesses. This mode tests all PROBE logic including the Buffer Assembly logic at the end of the PROBE which plugs into the target.

INTERNAL ONLY: The 68020 is connected to something other than the ATRON DEMO board. Only the internal diagnostics are performed; no cycles are run in the user system. This mode does not test the Buffer Assembly logic at the end of the PROBE. If the target system can support the access cycles and conditions as described in Appendix L, then the DEMO BOARD MODE may be used in the target system:

Answering 'N' or <Enter> to this prompt will inform the software to skip all target system access tests as shown in Appendix L. If the 68020 buffer assembly is plugged into the ATRON DEMO BOARD (or the target system supports the addresses and modes discussed earlier,) then 'Y' may be typed to test target system accesses as well as internal pod diagnostics. If 'N' or <Enter> is typed, the internal diagnostics will begin immediately. If 'Y' is typed, one additional prompt is provided:

Execution breakpoint instruction is BKPT #n, n=(0, 1, ..., [7]):

This prompt requests the BKPT instruction that is used by the pod for software and hardware execution breakpoints. For almost all targets, the correct answer is '7' or simply <Enter>. Some targets may have requested a special pod with another breakpoint instruction used. These targets should type the breakpoint number that matches their version of the pod.

CONF020 DIAGNOSTIC MESSAGES

The following tests are run in both DEMO BOARD MODE and INTERNAL ONLY MODE. They test the basic functionality of the hardware on the POD. No target system accesses are ever made. The target system must supply only +5V, GND, and CLK to the 68020 in order for these tests to operate.

Slave 8031 ROM checksum test . . . . . .
Slave 8031 RAM bitwalk/address test. . . . . .
Slave 8031 status check of 68020 test. . . . . .
Slave 8031 <--> 68020 mailbox test . . . . . .
68020 ROM checksum test. . . . . .
68020 data RAM bitwalk/address test. . . . . .
68020 execution BP RAM bitwalk/address test. . . . . .
68020 guard RAM bitwalk/address test . . . . . .
68020 map RAM bitwalk/address test . . . . . .

If any of the previous tests fail, it could be for the following reasons:

1. The cables from the POD to the Breakpoint/Trace boards are not firmly seated. Also make sure that the pins on the Breakpoint/Trace boards do not get bent together when plugging in the cables.
2. Check Vcc in the target, it may be less than +5v.
3. You may be using an AT clone which does not support writes from the cpu in the AT to location FFFF0 in the memory space of the AT.
4. The Base address you selected conflicts with other boards in the system such as extended memory boards.
5. The POD or Breakpoint/Trace boards have failed and must be returned to Atron for repair.

The following tests are run in both DEMO BOARD MODE and INTERNAL ONLY MODE only if a MAP RAM memory board is detected by the POD. They test the basic functionality of the mapped memory hardware on the pod. No target system accesses are ever made. The target system must supply only +5V, GND, and CLK to the 68020 in order for these tests to operate.

68020 wait RAM bitwalk/address test. . . . . .
68020 write enable RAM bitwalk/address test. . . . . .
68020 map memory data array bitwalk/address test . . . .
68020 map memory data array overlap test . . . . . .

If any of the previous tests fail, it could be for the following reasons:

1. The MAP RAM board is not fully seated in the POD. Open the POD and insure a good seating.
2. The MAP RAM board has failed and must be returned to Atron for repair.

The following tests are performed only if 'Y' was answered to the ATRON DEMO BOARD prompt. These tests will perform various accesses to memory in the target system. The target system must meet the requirements specified in Appendix L to support these tests.

Target system access (SAY TARGET WHAT'S HAPPENING) .
Start board, 8031 NMI stop board test. . . . . .
Software execution breakpoints . . . . . .
Hardware execution breakpoints . . . . . .
Hardware breakpoint. . . . . .
Trace data test. . . . . .
Guarded access test. . . . . .
Cache control test . . . . . .

If any of the previous tests fail, it could be for the following reasons:

1. Accesses in the target do not support the requirements of Appendix L.
2. The cable from the POD to the Buffer Assembly.is in close proximity to noisy circuits such as switching power supplies or video controller boards. The PROBE Buffer Assembly has been designed for maximum bandwidth and therefore may respond to noise in the target at frequencies to 33 mhz. Position the cable from the Buffer Assembly to the POD to not drape across noisy circuits in the target.
3. DSACK signals in the target do not respond.
4. The target generated Bus errors during the test.
5. The target system is using the same breakpoint instruction as you selected for use by PROBE in the CONF020 prompts.
6. The POD or Breakpoint/Trace boards have failed and must be returned to Atron for repair.

The following tests are performed only if 'Y' was answered to the ATRON DEMO BOARD prompt and MAP RAM is detected by the POD. These tests will perform various accesses to memory in the target system. The target system must meet the requirements specified in Appendix L to support these tests.

Map array access from target system. . . . . .
Map array overlap in target system . . . . . .
Map array wait state latch . . . . . .

If any of the previous tests fail, it could be for the following reasons:

1. The MAP RAM board is not fully seated in the POD. Open the POD and insure a good seating.
2. The MAP RAM board has failed and must be returned to Atron for repair.

Note: that the 68020 is reset after each of the confidence tests.

CHAPTER 2
THE PROBE USER INTERFACE

INTRODUCTION

This chapter tells you how to start the 68020 PROBE. It also describes the operation of the PROBE user interface. This chapter also includes information about the files on your disks and a summary of the different versions of PROBE software.

HOW TO START PROBE

To start the 68020 PROBE, enter:

PROBE [initfile]

To start the 68020 SOURCE PROBE, enter:

SOURCE [initfile]

If an [initfile] is specified when PROBE is loaded, then this file is loaded automatically. The initfile contains initializations for a debugging session (see Chapter 5 Init Load for more details). It is useful to create this custom file while you are in PROBE then save it for future use. The initfile is an option and need not be specified when you load PROBE or SOURCE.

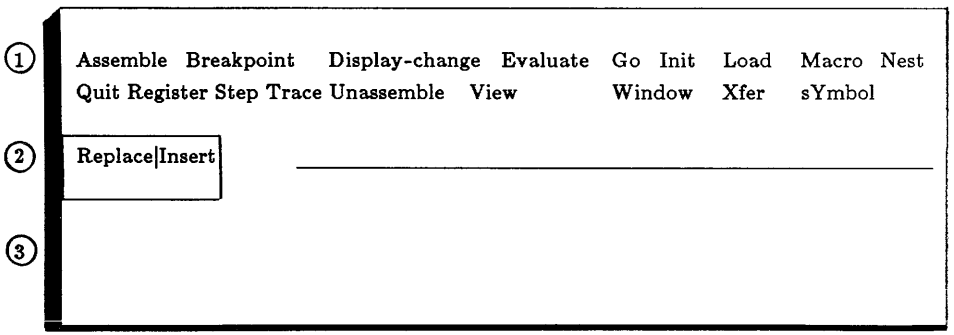
PROBE'S CONFIGURATION FILE

When invoked, PROBE and SOURCE look for a file named PROBE.CNF. If found, configuration parameters for PROBE's hardware base address, screen color, temproary heap files etc. are selected from this file. See Appendix C for a complete description of these configuration parameters. If PROBE.CNF does not exist, then standard defaults described in Appendix C are used. These standard defaults are usable for most systems. The file PROBE.CNF contains simple ASCII text and can be changed with a common text editor.

If you want to walk through an example, go to Chapter 4 at this point. Otherwise, keep reading for more background information.

THE USER INTERFACE

When PROBE is started, the screen looks like this:



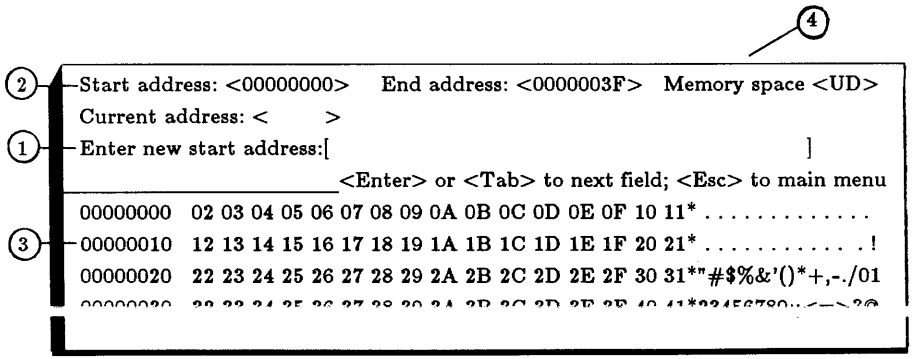
① The top of the screen contains the MENU BAR with an alphabetical list of commands. One of the commands is highlighted. A command can be selected by typing its first capital letter or by moving the highlight with the cursor positioning keys and typing <enter>.

② Immediately under the MENU BAR is a list of the subcommands for the command which is highlighted in the MENU BAR. Once a command from the MENU BAR is selected, a MENU BOX hangs down from the MENU BAR to let you select a subcommand. The subcommand is also selected with a single key. There may be further subcommands in some cases. Thus you progress through the command using single keystrokes without having to remember how the command works or what to do next.

③ This part of the screen is called the DISPLAY WINDOW and it appears immediately below the MENU BAR. Many commands display data in the DISPLAY WINDOW and let you modify the data displayed directly on the screen.

ENTERING COMMANDS AND PARAMETERS

At the end of the command chain, the command either displays data in the DISPLAY WINDOW or lets you enter data into a DIALOG BOX at the top of the screen. Some commands also present a formatted screen in the DISPLAY WINDOW with a set of fields. The fields on these screens are filled in when you enter data into the DIALOG BOX. Here is a sample **D**isplay **B**yte command display:



(1) If parameters must be entered for a command, you are prompted to supply the parameter. In chapter 5 of this manual, the prompts are shown in italics like this:

command prompt: [dialog box]

The command parameters are entered into the *[dialog box]* on the screen. The blinking cursor is in the DIALOG BOX.

(2) You can enter new data into the DIALOG BOX or simply type <enter> to select the default. The default data is shown in *<default data>* on the screen.

In some commands, there is a list of {options} to the right of the DIALOG BOX in curly brackets. The choices are separated by the | character. A default choice from the list is shown within the DIALOG BOX and this choice can be selected by simply typing the <enter> key. The DIALOG BOX would look like this:

command prompt:[choicea] {choicea|choiceb|choicec}

③ If there is a highlight field in the DISPLAY WINDOW while the blinking cursor is in the DIALOG BOX, then the contents of the DIALOG BOX are transferred to the highlight field when you type <enter>. If an expression is typed into the DIALOG BOX, it is evaluated before the value is put into the highlighted field. This lets you construct and edit expressions in the DIALOG BOX and have the results go to the highlight field in the DISPLAY WINDOW. A maximum of 255 characters can be entered into the DIALOG BOX (the characters scroll horizontally within the DIALOG BOX). The expressions inside the DIALOG BOX can be edited with the EDIT KEYS. See EXAMPLES OF USING EDIT KEYS later in this chapter.

Some command prompts you several times with DIALOG BOXES. In this case, after typing <enter> for each DIALOG BOX, the blinking cursor is transferred to the next DIALOG BOX. When all necessary data is entered for the command to operate, the command executes. Execution can mean:

Put data into the DISPLAY WINDOW
Put or get information from the target system
Put or get information from the AT memory or disk files

④ While in a command, you can move to another DIALOG BOX manually by typing the <TAB> key. This lets you make new entries into these fields. Also, in some commands, you are not automatically prompted through all DIALOG BOXES. This is because some of the DIALOG BOXES are so rarely used, it would be annoying to be prompted for them during each command. These fields are, however, displayed in the formatted screen of the command, and you may reach them by typing the <TAB> key until the blinking cursor gets to them. These are sometimes called <TAB> TO fields.

TERMINATING A COMMAND

Once in a command, you stay in the command until the <ESC> key is typed. This lets you make changes in the DIALOG BOX in a command for re-execution of the same command. ESC typed at any point will terminate a command and return control back to the MENU BAR. Any command execution which is in process but which has not already executed is canceled with the <ESC> key. Note however, that you stay in a command and may execute it several times before terminating it with the <ESC>. These changes which have already occurred are not undone with the <ESC>.

EDITING COMMAND PARAMETERS

While entering data into the DIALOG BOXES, you may want to edit for changes and mistakes. If the syntax entered into the DIALOG BOX field is not recognized, an error appears in an MESSAGE BOX describing the problem. In addition, the blinking cursor is placed just after the point in the DIALOG BOX where the syntax could not be recognized by PROBE. The data in the DIALOG BOX may be edited with the editing keys. These keys are described in the section titled WHAT THE KEYS DO.

WHAT THE EDIT KEYS DO

Keys affecting the display window

Cursor Keys (Up, Dn, Left, Right)

Move the highlight field in the DISPLAY WINDOW or the MENU BAR.

PgUp and *PgDn*

For commands which can display additional screens of information in the DISPLAY WINDOW, these keys show the previous or following information associated with the command in process.

Ctrl PgUp and *Ctrl PgDn*

For commands which display files, real time trace data, or source files during single step in the DISPLAY WINDOW, these keys move to the start and end of the file.

<TAB

Move blinking cursor to the next DIALOG BOX for the command in process.

Keys for editing data in the dialog box

<F3

Copy all remaining characters from the last time you typed into this DIALOG BOX the DIALOG BOX. This lets you recall a previous line instead of retyping it. This key functions exactly like the DOS F3 key

RUBOUT

Move blinking cursor left one character and blank this character.

HOME

Move blinking cursor one space toward the beginning of DIALOG BOX but do not delete the characters. The cursor keys cannot be used for moving the blinking cursor in the DIALOG BOX since they are reserved for moving the highlight field in the DISPLAY WINDOW.

END

Move the cursor one space toward the end of the DIALOG BOX. The cursor keys cannot be used for moving the blinking cursor in the DIALOG BOX since they are reserved for moving the highlight field in the DISPLAY WINDOW.

Ctrl HOME

Move blinking cursor to the beginning of DIALOG BOX.

Ctrl END

Move the cursor to the end of the DIALOG BOX.

DEL

Delete the character above the blinking cursor.

Keys terminating commands

ESC

Terminates a command which is in the process of executing.

Ctrl Break

Terminates current command in process which does not stop with the <ESC> key.

EXAMPLES OF EDITING COMMAND PARAMETERS

The EDIT KEYS can be used to make changes in the DIALOG BOX. The definition of these keys was described previously. Here are some examples of using the edit keys.

Start by using the Display Byte command. Type:

DB

The command prompts for startaddress in the DIALOG BOX. Type:

\MAIN$MODULE\PROCXEDURENAME$

Since no symbol matches this, an error message pops up. Type any key to pop down the error message. The blinking cursor is in the DIALOG BOX. Use the END key to move the blinking cursor just under the X. Then type the DEL key. Now, to re-execute this command type <enter>. The reason that the HOME and END keys are used to move the blinking cursor within the DIALOG BOX is because the cursor keys are reserved for moving the highlight field in the DISPLAY WINDOW.

TAB FIELDS

The PROBE commands typically have several options which are not normally changed They may be changed occasionally, however, and therefore must be available when needed. These options are displayed on the screen as:

Label:< >

These fields contain default values in the < >. You may get to these fields to change the defaults by typing <TAB>. The default stays in the < > until you select then change it. The <TAB> fields are described for each command in Chapter 5.

WATCH WINDOWS

You can create custom WATCH WINDOWS which display many different types of data with the Window command. WATCH WINDOWS overlay the DISPLAY WINDOW. They are assigned to any *AltKey* (i.e. hold down Alt while you type any other key). They are popped up and down by typing the *AltKey*. If more than one window is popped up at a time, they "stack" one under the other in the DISPLAY WINDOW. If a command is in process under the WATCH WINDOW, it is temporarily suspended (except for single step and Go commands) until all the WATCH WINDOWS are popped down.

ERROR MESSAGES

When a command or command parameter is not recognized by PROBE, an error message "pops up" on the screen in a MESSAGE BOX to indicate the problem. Appendix A contains a summary of these error messages. The error message is "popped down" and you are returned to the command by typing any key.

COPY AND PASTE

PROBE has a copy and paste features which eliminates typing in the long addresses and symbolnames requried during program debugging. When you are looking at information on one PROBE screen which you want to pick up and deposite into fields on another PROBE screen, you can use PROBE's copy and paste keys.

To copy information from a screen, start by typing the <F10> key. The highlight field in the DISPLAY WINDOW will shrink to the size of one character and will be positioned on the left side of the screen about half way down from the top. As usual, use the cursor keys to move this small highlight field to the start of the information you want to copy. Next, type <F10> again which anchors the highlight field at this location. Use the cursor keys again to spread the highlight field over the characters on the screen you want to copy. Next, you can deposite the highlighted information into any of the following function keys: <F6>, <F7>, <F8>, <F9>. Note the curosr motion will be faster if you have a mouse.

The information stored in these function keys can now be pasted into any other screen. The location on any screen which has the blinking cursor can receive the information stored in the function key simply by typing the function key. The ASCII text stored in the function key is available each time the function key is typed until the information in the key is changed.

VERSIONS OF PROBE SOFTWARE

There are several files on your PROBE distribution diskettes which may or may not be needed depending upon what you are doing. A list of these files, their current version numbers, and a description is given in Appendix E for each type of PROBE software. Only those used for "RUNNING" are required for the actual execution of PROBE software.

This manual describes the standard versions of the PROBE and SOURCE PROBE software. Other versions of these software products are available which are optimized for specific applications. These versions are described by inserts to this manual. These versions are described in Appendix E.

USING PROBE WITH A MOUSE

PROBE supports the use of a mouse to replace keystrokes. The MOUSE must be a Microsoft mouse or have a compatible Microsoft mouse driver. To use PROBE with a mouse you must install the mouse driver in your system. First copy the mouse driver to the top directory on the drive which boots the operating system on your AT computer. Add the following entry in your CONFIG.SYS file:

```
device = mouse.sys
```

When invoked, PROBE detects the presence of the mouse. Moving the mouse is equivalent to using the cursor keys. Clicking the left mouse button is equivalent to typing the <enter> key. Clicking the right mouse button is equivalent to typing the <Esc> key.

CHAPTER 3
GENERATING AND USING SYMBOLS

INTRODUCTION

68020 PROBE allows you to use the symbolic information from your program during debugging instead of absolute numbers. This symbolic information may consist of public variables, public procedures, functions, subroutines, modulenames, and high level language line numbers. Some compilers will also produce symbols for local variables and procedures. This chapter describes the development environments for generating programs and symbols. Then it describes how to reference symbols during symbolic debugging. Other parts of this manual describe important information regarding symbols. This information is not duplicated in this Chapter but here is a summary of where to find this information.

Description	Location
[illegible]	[illegible]
Compiler controls to generate symbols	Appendix F
Single step code by C linenumbers	Source Step command Chapter 5
Loading symbols into PROBE's symboltable	Load command Chapter 5
Defining symbols on line	sYmbol Define command Chapter 5
Deleting symbols	sYmbol Remove command Chapter 5
Ignoring case in symbols	sYmbol Case command Chapter 5
Defining a default modulename prefix	sYmbol Module default Chapter 5
Display symbol which matches address	Window command Chapter 5
Using a symbol in an indirect reference	Dereferenced memory Chapter 5

DEVELOPMENT ENVIRONMENTS

The code for your target can be generated on a number of different computers with a variety of compilers. The compiler controls needed to generate this symbolic debugging information is a function of the manufacturer of the compiler and the type of compiler. Appendix F describes *some* compilers and controls which can be used with PROBE. Refer to your compiler and linker manuals for details on how to produce a PROBE compatible OBJECT MODULE FORMAT (OMF).

OBJECT MODULE FORMATS

To do symbolic debugging, the symbol table for the program must be loaded into PROBE. The symbols may be in the executable object file or in a separate file. The symbols in the file must match one of the OMF's compatible with PROBE. These are described in Appendix G. If the code has not been compiled on the AT, it must be down loaded to the AT and stored in a DOS compatible file on the AT (unless the AT is on a network described later). If the image of executable code and symbol table matches an OMF supported by PROBE, you should load an exact image of the OMF into the DOS file.

If the compiler does not generate a PROBE compatible OMF, you may be able to write a conversion utility which converts the format produced by your compiler to one supported by PROBE.

The executable code can be loaded into the target system (or MAP RAM) and the symbol table loaded into the PROBE symbol table space with the Load commands (see Chapter 5 for details). The Load command has many options and code, data, and symbols can be loaded separately or together. Code and symbols can even loaded with offsets if the code is relocatable.

NETWORKS

If the AT host for the PROBE is operating on a network, PROBE can load the code directly from the network to the target. PROBE software is PC DOS compatible and goes entirely through DOS to load files. If DOS is operating with a network, the PROBE load command calls the operating system with a file to load or save. The network drivers then passes the file between the network and PROBE.

SCOPE OF SYMBOLS

A symbol is a 32 bit value which is normally used as an address in PROBE commands. A symbol can be used in an expression at any place a value is expected. Symbols can have a scope if it is defined in the OMF. PROBE interprets the scope of symbols in the same way as the the C language. Some definitions are shown here for reference:

Symbol scope	**Description**
EXTERNAL or PUBLIC	Visible to all modules. This symbol has an absolute address when the program is loaded
LOCAL	Visible only to the declaring procedure. This symbol is stack based.
STATIC	Visible to module after point of declaration. This symbol has an absolute address when the program is loaded.

If the symbol is stored in PROBE's symbol table as a PUBLIC or EXTERNAL symbol it is referenced as:

\symbolname

If the symbol is LOCAL to a module or STATIC it is referenced as:

\modulename\symbolname

If the symbol is LOCAL to a module in the current procedure and known only to a block it is referenced as:

\modulename\blockname\symbolname

If the symbol is a linenumber for a highlevel language executable statement, then it is referenced as:

\modulename#linenumber

Procedure names and function names are treated the same as symbol names.

REFERENCING SYMBOLS IN COMMANDS

When a symbol is used in a PROBE command, it is treated as a 32 bit value or address. In this manual, most examples will use symbols instead of absolute numbers. Here are some notes regarding symbols used in commands:

1. A default modulename prefix string is assumed by PROBE for each symbol. A detailed description of how a symbol is parsed with this default prefix is described in the sYmbol-Module-name-default command in Chapter 5.
2. A symbol can be used as a pointer to another address by using it in and indirect reference (see Chapter 5 Dereferencing Memory).

EXAMPLES OF USING SYMBOLS

In Chapter 5, the PROBE commands will use symbols in many examples. Here are a few examples of how symbols are used:

This expression references the PUBLIC symbol MAIN

```
MAIN
```

This expression references the procedure named IO assuming it is in the module named IOROUTINES.

```
IOROUTINES\IO
```

This expression references the byte pointed to by the 16 bit pointer MEDIUMPOINTER. See Chapter 5 Dereferenced Memory for details.

```
[MEDIUMPOINTER].W
```

CHAPTER 4
PROBE TUTORIAL EXAMPLE

INTRODUCTION

The commands are listed alphabetically in Chapter 5, COMMAND REFERENCE, and no attempt is made to duplicate the complete explanation of each command as it is being used in these examples. If the short explanation of the command is not sufficient in the example, please turn to Chapter 5, COMMAND REFERENCE, for more information.

The program to be debugged is a C program. The source and object files for the program are included on your disk, so you can actually try the example in real time. The example is taken from the C Programming Manual by Kernighan and Ritchie but has been broken into three modules to demonstrate debugging a multiple module program. If you are using Assembler, Pascal, or some other language in your application, you will still find this tutorial useful from a procedural point of view.

The program calculates a Fahrenheit to centigrade table and stores the table in memory. It loads at location 400 and does not use any external devices, therefore, you may be able to down load it into your target system or to MAP RAM in the PROBE or to the DEMO BOARD to try out the example in real time. Since the example makes use of symbolic debugging, here are some of the symbol names which are pertinent to this debugging session:

MODULENAME	SYMBOLNAME	DESCRIPTION
\	CELSIUS	Variable used to store celsius temperature
\	COMPUTE	Procedurename which does computation
\	FAHR	Variable used to store fahrenheit temperature
\	LOWER	Lower limit for the table
\	UPPER	Upper limit for the table
\	STEP	Increment between temperature values

HOW TO INITIALIZING PROBE

In this example, the input you provide from the keyboard is shown in **bold print.** Prompts in DIALOG BOXES provided by PROBE are shown in *italics.* In simple examples, the bold print is shown alone without the screen which is presented by the command. In more complex examples, the bold print is shown within the screen.

First, you can load PROBE. If you are running 68020 PROBE type:

probe <enter>

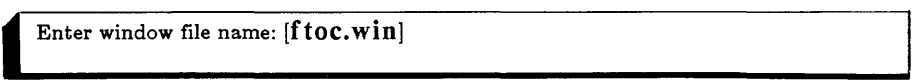
Or, if you are running the source level debugging version, 68020 SOURCE PROBE, type:

source<enter>

As described in Chapter 3, PROBE displays the MENU BAR of commands at the top of the screen. Each command is selected with a single key stroke. PROBE has Watch Windows which let you display information in the target with a single keystroke. A set of windows have been previously created for this demo and stored in a file. To load this file of window definitions, use the Window Load command

wl

PROBE prompts you for the name of the file - type in the filename FTOC.WIN.



HOW TO INITIALIZE THE MAP RAM BOARD

For this example lets assume you have a MAP RAM in your PROBE configuration. Since a program is going to be loaded into low memory, you must map one of the blocks of the PROBE MAP RAM to this area. MAP RAM provides up to 4 128k blocks which can be mapped on any 64k boundary to the target memory space. Start the Display-change Map memory command.

dm

For this example, the lowest 128k of the target address space will be mapped to memory array 0 of the PROBE MAP RAM. The default state of the memory map is shown in the screen below.

```
Array 0: 128K          Array 1: 128K     Array 2: 128K     Array 3: 128K
Don't care bits:<.... .... .... .... .... .... XXXX XXXX XXXX XXXX
Enter new start address of block:[ ]
                                   Arrows to move ■; <Esc> to main menu
Start     End       Bus    Guarded  Map To  Map Write  Map wait for   Map wt
Address   Address   Size   Access   Array#  Protected  target ready   states
00000000  FFFFFFFF  Long   No
```

The default state of the map shows all memory mapped to Long Bus Size (32 bit), Not Guarded access (it is free to be accessed), and mapped to the target system. PROBE maps the memory space of the 68020 into blocks. Each block can have a different size - from 64k to the entire memory space. To map the lower 128k to the MAP RAM board array #0 , move the highlight to the End Address field and enter 1FFFF Then move the highlight to the MAP to Array # field and enter 0. No other fields need to be set to map the memory, therefore, the DISPLAY WINDOW will look like this:

```
Array 0: 128K          Array 1: 128K     Array 2: 128K     Array 3: 128K
Don't care bits:<.... .... .... .... .... .... XXXX XXXX XXXX XXXX
Enter new start address of block:[ ]
                                   Arrows to move ■; <Esc> to main menu
Start     End       Bus    Guarded  Map To  Map Write  Map wait for   Map wt
Address   Address   Size   Access   Array#  Protected  target ready   states
00000000  0003FFFF  Long   No       0       No         N              1
004FFFF   FFFFFFFF  Long   No       none
```

PROBE splits the initial single block. One with the characteristics you specified, and the other with the default characteristics. If you do not have a MAP RAM board, the default state of the MAP

assumes that memory is in the target system and that the bus size for the target is Long. If the Bus Size in your target is not 32 bits for the block into which the program will load, you must change the Bus Size to fit your system or the PROBE breakpoints in this tutorial will not work properly. Type <Esc> to return to the MENU BAR.

LOADING THE PROGRAM

A demo program is included on the PROBE distribution diskette. This demo program loads into memory in the target system at location 400 and is about 1k bytes in length. For this demo program to work, the target must have functioning memory at these locations or the PROBE must include a MAP RAM board. No other target system resources are required for the demo. Use the Load command to load the program into the target memory (or MAP RAM) type:

lp

When PROBE prompts you for the filename, type FTOC.COF.

*Enter program file name: [***ftoc.cof***]*

SYMBOLIC DEBUGGING INFORMATION

This demo program was produced with the Microtec Research C compiler. This compiler produces an Object Module Format which contains symbolic debugging information which PROBE can load into the PROBE internal symbol table. You can now look at symbols in the symbol table with the sYmbol Display command.

yd

This program consists of three modules, (i.e. three separate compilations) linked together. The symbol table, therefore, contains variables and linenumbers for three modules. When referencing these symbols or linenumbers the modulename must be included when specifying the symbol (or it must come automatically from the default prefix). There are also public symbols which do not need a modulename prefix. The symbolnames and linenumbers for the modulename are shown in the DISPLAY WINDOW. Since no modulename has yet been specified, the default modulename assigned

by PROBE is \. Change the modulename to FtoCM and display the symbols in this module. This is done by typing <Tab> to get to the Module:< > field:

```
Module:<\>
Enter symbol name:[<Tab>]
                  <Enter> only picks  : PgUp/Dn, arrows move []; <Esc> main menu
Address    Symbol name
00000826   CELSIUS
00000536   COMPUTE
00000832   FAHR
000004fa   INIT
00000420   LDIVT
00000424   LMODT
00000826   LOWER
00000444   MAIN
0000083C   OUTPUTBUFFER
0000088E   OUTPUTBUFFERINDEX
```

Modulenames are displayed in the DISPLAY WINDOW. The module FtoCM can be selected by moving the highlight field to this modulename and by simply typing <enter>:

```
Module:<\>
Enter module name:[<DownArrow><enter>]
                  <Enter> only picks  : PgUp/Dn, arrows move []; <Esc> main menu
ModuleName
FtoCIO
FtoCM
FtoCStrt
```

The symbols for the module FtoCM are put into the DISPLAY WINDOW. Now you can type the PgDn key to page through the symbols for this module. You could also define a new symbol called START and assign it the value of the current program counter as shown on the screen.

```
Module:<FtoCM\>
Enter symbol name:[start]
Enter new symbol value:[pc]

Address    Symbol name
00000400A          start
```

To get out of the sYmbol commands and back to the MENU BAR, type the <ESC> key.

PROBE can set initial conditions automatically with the PROBE Initialize Load command. This loads the program, symbol table, sets the MAP, loads Watch Windows and macros. For the SOURCE version of PROBE, all initializations for source level debugging are performed. The data for the initial conditions was derived by saving these same items from a previous debug session. If you want to try the Initialize command, Quit and reload PROBE. Then load the initialization conditions from the file FTOC.INI by typing:

ilftoc.ini

An alternative is to include the initialization file when PROBE is first invoked. To do this type:

probe ftoc.ini

This demo is using SOURCE rather than PROBE. Therefore, some of the displays include source level information. If you are using PROBE, the displays will be identical but will not include source code.

SINGLE STEP THROUGH PROGRAM/POP UP WATCH WINDOWS

The program counter is set to the start of program execution when the program is loaded. You can now start single stepping the program in assembly language steps with the Single step Assembly language command, type:

sa

PROBE prompts you for the start of the program execution. Since the program counter is already set to the start of program execution simply type <enter>.

*Enter new start address: [**<enter>**]*

An instruction is executed each time <enter> is typed. Try single stepping several instructions.

```
Steps to take for each <Enter>: <001>
After <Enter>, step while: <False>
"B" to run to  ; "J" to run to instr after []: <Enter> to step from 00000400
        <Tab> to next field above: PgUp/Dn Arrows to move []; <Esc> to main menu
D0=0000002 D4=00000000 A0=0000072A A4=00000000 PC=00000400 CACR=0000000
D1=00000001 D5=00000000 A1=0000077C A5=00000000 USP=00000000 CAAR=00000
D2=00000000 D6=00000000 A2=00000722 A6=00000700 ISP=000006DC VBR=0000000
D3=00000000 D7=00000000 A3=00000000 A7=00006DC MSP=0000000 SFC=0 DFC=0
SR=2704=T0 S1 M0 I7 X0 N0 Z1 V0 C0

\START:
00000400  LEA.l     (00000708),A7
 ^ ^ ^ ^    Op 1 value=00000708, address=00000708=\FtoCStrt\STACKTOP
00000406 MOVE.l     A7,A6
00000408 JSR        (00000444)
0000040E BRA        00000400
\FtoCStrt\DOSIGNEDDIVIDE:
00000410 TST.l      D1
00000412 BEQ        0000041A
```

While single stepping, the blinking cursor is at the instruction which is about to be executed. It is not executed until <enter> is typed. Note during each step that the symbols for address and operands are included in the display. Also note that each operand is evaluated before it is single stepped to show its current contents. This eliminates bailing out of the single step command to see the contents of operands. The highlight field can be moved with the PgUp and PgDn keys to let you scroll through and inspect your code. The

blinking cursor remains at the instruction which will be executed when <enter> is typed. Use the PgDn and cursor keys to locate the highlight field to the instruction shown here:

```
34. Init (&Lower, &Upper, &Step)
00000452  PEA       (0000071E)
```

You can start program execution and set a breakpoint at this instruction on the screen which is highlighted by typing:

b

You can pop up a Watch Window to display program variables while single stepping. The definition of these windows came in when the file of window definitions was loaded. Pop up two Watch Windows by typing:

<AltV>
<AltL>

The first Watch Window displays the variables Fahr and Celsius. The second Watch Window displays the values of Lower, Upper, and Step. The screen looks like this:

```
Steps to take for each <Enter>: <001
After <Enter>, step while: <False>
"B" to run to  ; "J" to run to instr after []: <Enter> to step from 00000400
       <Tab> to next field above: PgUp/Dn Arrows to move []; <Esc> to main menu
D0=00000002 D4=00000000 A0=0000071E A4=00000000 PC=00000452 CACR=000000
D1=55555555 D5=00000000 A1=0000071A A5=00000000 USP=00000000 CAAR=00000
D2=00000000 D6=00000000 A2=00000722 A6=00000700 ISP=000006FC VBR=000000
D3=00000000 D7=00000000 A3=00000000 A7=00006FC MSP=0000000 SFC=0 DFC=0
SR=2700=T0 S1 M0 I7 X0 N0 Z1 V0 C0

    <ALTV>
FAHR = +0
CELSIUS = +0
    <ALTL>
Lower = +0          Upper=+100     Step=+10

34.         Init (&Lower, &Upper, &Step);
00000454  PEA.l     (0000082E)
^ ^ ^ ^     Op 1 value = 0000082E=\STEP
0000045A            PEA.l  (0000082A)
00000460            PEA.1          (00000826)
00000466  JSR       (0000094,PC)
```

The Watch Windows are updated after each single step. Pop the Watch Window down again by typing the same *AltKeys*.

<AltV>
<AltL>

It is common during single stepping to want to branch around subroutines. Actually, you want to run through the subroutine in real time and stop at the instruction after the one in the highlight field. For example, single step the code until you get to this instruction:

```
00000466  JSR       (00000094,PC)
```

To run through this subroutine until you land on the instruction following this one, type:

j

Even if you made a mistake and typed <enter> too many times and stepped into the subroutine, this feature still works. Simple position the highlight field to the instruction above and type J. PROBE automatically sets a breakpoint at the instruction after the one in the highlight field and then runs real time until the breakpoint is detected.

Lets suppose you want to step 5 instructions at a time. You can do this by changing the field in the menu for the step command. First type **<Tab>** to get the following prompt - then type 5.

*Enter number of steps to take for each <Enter>:[**5**]*

Each enter will now take 5 steps. Next lets assume you want to step until the variable FAHR is equal to 10 decimal. This can be done by typing **<Tab>** until you get the following prompt.

*Enter condition to test for end of stepping: [**fahr != 10t<enter>**]*

Now you can continue the single stepping until this condition is met by typing:

<enter>

If the condition is never met and you want to bail out of all this single stepping, type **<esc><esc>**.

DISPLAYING MEMORY AND REGISTERS

Display and change of target memory is done easily with PROBE's **D**isplay command. Display bytes in memory with the **D**isplay **B**yte command

db

PROBE prompts you for the start address and end addresses for the data to be displayed. Type in the values shown in this screen:

```
Start address: <00000832>   End address: <0000097F>   Memory space [SD]
Current address: <00000722>
Enter new start address:[fahr]
Enter new end address: [<enter>]
                          <Enter> or <Tab> to next field; <Esc> to main menu
00000830  00 00 00 00 00 0A FF FF FF EA 30 20 20 2D 31 00* .......... .0. -17 *
00000840  12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21* ..............  .!*
00000850  22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31*"#$%&'()*+,-./01 *
00000860  32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41*23456789:;<=>?@*
```

You can use the cursor keys to move the highlight around the screen and to page through memory. You can also change the data in memory by simply entering a new value or expression. To get out of this command type:

:esc:

Sometimes it is desirable to only display and change a single address in memory without reading any other memory locations - especially in the case of peripheral devices. PROBE lets you do this with the Display Single address command. In this case PROBE also prompts you for the length of the memory location and if you want to read or write to the location. Let PROBE prompt you through the command to read the single address of a Long word at address FAHR. The following keys are typed to do this.

dslrfahr

While in the Display Single Address command, each time you type the <enter> key, this address is read and appears in the DISPLAY WINDOW.

```
Address: <00000832>   Memory space [SD]

Enter new address:[fahr]
                          Enter> or <Tab> to next field; <Esc> to main menu
 00000832 = 00000000
```

Displaying registers is even simpler - type R and the registers are displayed in the DISPLAY WINDOW:

```
Enter new value:[        ]
                                        Arrows to move ; <Esc> to main menu
D0=0000000 D4=000000 A0=00000000 A4=0000000 PC=00000000 CARC=000000
D1=0000000 D5=000000 A1=00000000 A5=0000000 USP=00000000 CAAR=000000
D2=0000000 D6=000000 A2=00000000 A6=0000000 ISP=00000000 VBR=000000
D3=0000000 D7=000000 A3=0000000 A7=000000 MSP=00000000 SFC=0 DFC=0
SR=0000 = T0 S0 M0 I0 X0 N0 Z0 V0 C0
```

You can move the highlight field in the DISPLAY WINDOW and change the registers on the screen.

START PROGRAM EXECUTION AND SET BREAKPOINTS

Program execution can start with the Go command. PROBE has two kinds of breakpoints. Non-sticky breakpoints can be set in the Go command. Sticky-breakpoints can set with the Breakpoint command and are automatically included in future Go commands if they are active. Start program execution from the current program counter, type:

g<enter>

PROBE asks you if you want to start program execution or set a non-sticky breakpoint. Set a non sticky breakpoint at the address represented by the symbol COMPUTE, type:

*Start program execution now: [***n***] {Yes|No}*
*Enter breakpoint address:[***compute***]*
*Enter new end address:[***<Tab>***]*
*Start program execution now: [***<enter>***] {Yes|No}*

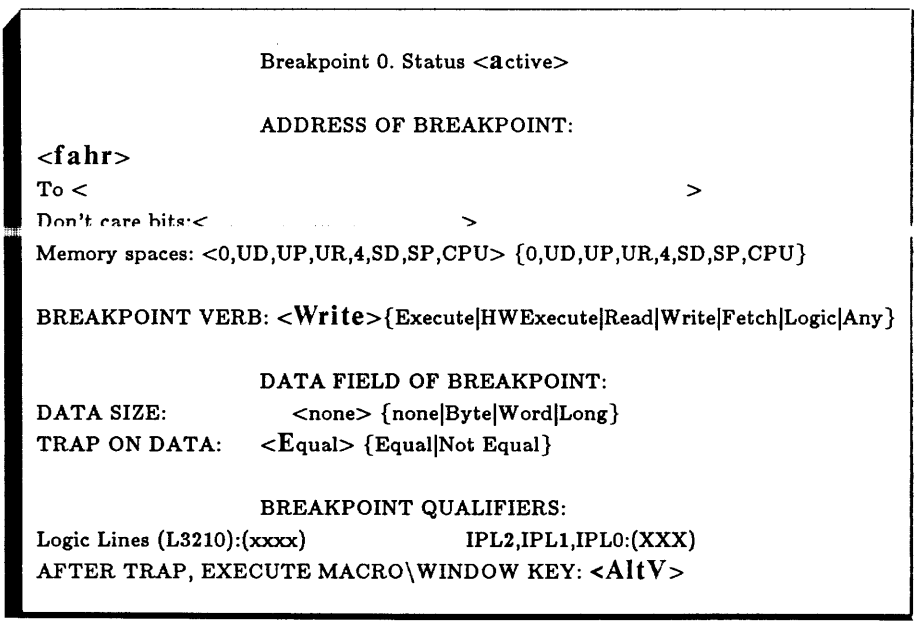
When this breakpoint occurs, PROBE prints the following message telling you the breakpoint occurred and showing you the current program location.

Non-sticky Breakpoint detected: PC=00000536=\COMPUTE

PROBE has 10 sticky breakpoints labeled 0 through 9 which can be defined with the Breakpoint command. Define breakpoint 0 to trap a write to the variable FAHR and tell PROBE to pop up the Watch Window assigned to the *ALTV* key when it occurs. First type:

bd0

The sticky breakpoint screen pops up. Fill in the screen as shown.



Once the breakpoint is defined, type the <ESC> key to go back to the MENU BAR. Then, use the Go command to start program execution, type the following keys:

g<enter><enter>

When the breakpoint occurs, the Watch Window pops up. Pop down the Watch Window by typing any key. Now modify the breakpoint to trap a write to this variable only when the data written to it is a

10 decimal. Note that FAHR is a long word variable. The breakpoint screen looks like this:

```
                    Breakpoint 0. Status <active>

                    ADDRESS OF BREAKPOINT:
<fahr>
To <                                                    >
Don't care bits:<.... .... .... .... .... .... .... ....>
Memory spaces: <0,UD,UP,UR,4,SD,SP,CPU> {0,UD,UP,UR,4,SD,SP,CPU}

BREAKPOINT VERB: <Write>{Execute|HWExecute|Read|Write|Fetch|Logic|Any}

                    DATA FIELD OF BREAKPOINT:
DATA SIZE:          <Long> {none|Byte|Word|Long}
DATA VALUE:         <10t>
Don't care bits:<.... .... .... ...>
TRAP ON DATA:   <Equal> {Equal|Not Equal}

                    BREAKPOINT QUALIFIERS:
Logic Lines (L3210):(xxxx)              IPL2,IPL1,IPL0:(XXX)
AFTER TRAP, EXECUTE MACRO\WINDOW KEY: <AltV>
```

Now start the program again with the Go command. This time an error message pops up.

BP0 -- Long data must start on long word boundary (A1A0=00)

When the 68020 does a long word write to a variable which is not aligned on a long word boundary, such as FAHR, it does two sequential writes. To breakpoint on this condition you can trap on one of the writes or set a sequential breakpoint to trap both writes as described later. In this case, since the data you are trying to trap fits within a word value, go back and trap a word write to FAHR+2. The screen looks like this:

```
                    Breakpoint 0. Status <active>

                    ADDRESS OF BREAKPOINT:
<fahr+2>
To <                                                    >
Don't care bits:<.... .... .... .... .... .... .... ....>
Memory spaces: <0,UD,UP,UR,4,SD,SP,CPU> {0,UD,UP,UR,4,SD,SP,CPU}

BREAKPOINT VERB: <Write>{Execute|HWExecute|Read|Write|Fetch|Logic|Any}

                    DATA FIELD OF BREAKPOINT:
DATA SIZE:          <Word> {none|Byte|Word|Long}
DATA VALUE:         <10t>
Don't care bits:<.... .... .... ...>
TRAP ON DATA:    <Equal> {Equal|Not Equal}

                    BREAKPOINT QUALIFIERS:
Logic Lines (L3210):(xxxx)              IPL2,IPL1,IPL0:(XXX)
AFTER TRAP, EXECUTE MACRO\WINDOW KEY: <AltV>
```

Now do the Go command and note that the breakpoint occurs.

SETTING SEQUENTIAL TRIGGER CONDITIONS

PROBE also has a real time sequential trap capability to trap a series of events. To define a sequential breakpoint trap, you must first define the breakpoints which will be used in the sequence. Breakpoint 0 has already been defined from the previous example. Use the **B**reakpoint command again to set sticky breakpoint number 1 and 2 as shown in the DISPLAY WINDOWS below.

```
                    Breakpoint 1. Status <active>

                    ADDRESS OF BREAKPOINT:
<celsius+2>
To <                                                    >
Don't care bits:<.... .... .... .... .... .... .... ....>
Memory spaces: <0,UD,UP,UR,4,SD,SP,CPU> {0,UD,UP,UR,4,SD,SP,CPU}

BREAKPOINT VERB: <Write>{Execute|HWExecute|Read|Write|Fetch|Logic|Any}

                    DATA FIELD OF BREAKPOINT:
DATA SIZE:          <Word> {none|Byte|Word|Long}
DATA VALUE:         <-17t>
Don't care bits:<.... .... .... ...>
TRAP ON DATA:   <Equal> {Equal|Not Equal}
```

```
                    Breakpoint 2. Status <active>

                    ADDRESS OF BREAKPOINT:
<fahr+2>
To <                                                    >
Don't care bits:<.... .... .... .... .... .... .... ....>
Memory spaces: <0,UD,UP,UR,4,SD,SP,CPU> {0,UD,UP,UR,4,SD,SP,CPU}

BREAKPOINT VERB: <Write>{Execute|HWExecute|Read|Write|Fetch|Logic|Any}

                    DATA FIELD OF BREAKPOINT:
DATA SIZE:          <Word> {none|Byte|Word|Long}
DATA VALUE:         <20t>
TRAP ON DATA:   <Equal> {Equal|Not Equal}
```

Now you can define a sequential breakpoint to stop when these three (0,1,2) have been encountered in a sequence. The sequential breakpoint definition is started by typing:

bds

The sequence to be detected for this example is breakpoint 0 followed by 1 followed by 2. In addition, tell PROBE not to stop until this sequence has happened FF times(255t). PROBE prompts you through the screen to set up this sequence. When you are done, the DISPLAY WINDOW should look like this.

```
Enter sequential conditional number: [3] {1|2|3|4|5|6}
_____<Enter> to next field; <Tab to next breakpoint; <Esc> to main menu.
                    Sequential Breakpoint.  Status: <active>
Sequential condition: <3>                 Breakpoint assignment to BP's
1. A or B or C or D                       A is assigned to BP <0>
2. A arms B, reset by C                   B is assigned to BP <1>
3. A arms B arms C, reset by D            C is assigned to BP <2>
4. A arms (B or C), reset by D            D is assigned to BP <none>
5. (A or B) arms C, reset by D
                                          Pass count before trap:<FF>
                                          Continue trace after breakpoint:<No>

BP # Status      Breakpoint-addr [To-range]   Verb     Size   Data   Match
BP 0 active      fahr+2                       Write    Word   10t    Equal
BP 1 active      celsius+2                    Write    Word   -17t   Equal
BP 2 active      fahr+2                       Write    Word   20t    Equal
BP 3 inactive                                 Execute
BP 4 inactive                                 Execute
BP 5 inactive                                 Execute
BP 6 inactive                                 Execute
BP 7 inactive                                 Execute
BP 8 inactive                                 Execute
BP 9 inactive                                 Execute
```

Note that while you are defining the sequential breakpoint, an abbreviated summary of the sticky breakpoints is shown for easy reference. Sticky breakpoints which are used in a sequential breakpoint, only cause a breakpoint in the sequence and not separately. Other active breakpoints not used in the sequence, however, are detected separately.

g<enter><enter>

Once the sequential conditions have been defined, type <ESC> to get to the MENU BAR. Then use the Go command to start program execution. Once the 255 breakpoint sequences have occurred, PROBE stops program execution. It also displays which breakpoint occurred in the DISPLAY WINDOW since you may have many breakpoints set. Another type of sequential breakpoint can detect excessive amounts of time occurring between two breakpoints. To try this, go back to the sequential breakpoint screen and set a breakpoint when BP2 follows BP0 by more than 50 microseconds. The screen looks like this:

```
Enter sequential conditional number: [6] {1|2|3|4|5|6}
          <Enter> to next field; <Tab to next breakpoint; <Esc> to main menu.
                        Sequential Breakpoint.  Status: <active>
Sequential condition: <6>                  Breakpoint assignment to BP's
1. A or B or C or D                        A is assigned to BP <0>
2. A arms B, reset by C                    B is assigned to BP <2>
3. A arms B arms C, reset by D             C is assigned to BP <none>
4. A arms (B or C), reset by D             D is assigned to BP <none>
5. (A or B) arms C, reset by D
6. A to B time greater than: <0:00.000,050>
```

Get back to the MENU BAR by typing <ESC>. Since sticky breakpoint 1 is still active, it must be inactivated or it may cause a breakpoint. Type:

bi1<esc>

If the time between BP0 and BP2 is greater than 50 microseconds in your system, then a breakpoint will occur.

DISPLAY THE PROCEDURE CALLING SEQUENCE

PROBE can display the procedure calling sequence by analyzing the stack frames. In modular program design, you may have many levels of procedure nesting and PROBE can show you the calling sequence of these procedures. Use the Nest command to produce this screen:

```
Stack Chaining Register:<A6> Stack  Memory space:<SD>  Code Memory space<SD>
Enter new chaining register:[<enter>
                                        <Tab> to next field; <Esc> to main menu
PC is      000005E4=\ftocio#47+0000000E
Called from 000006D4=\ftocio#102+0000002C
Called from 000006A4=\ftocio#97+0000002A
Called from 0000070E=\ftocio#133+00000004
Called from 000004A0=\ftocm#55+00000006
Called from 00000408=\Start+00000008
```

The DISPLAY WINDOW shows the procedure calling sequences. Note that in some systems, PROBE may track the stack frames into non-existent memory and a ready time out may occur. If it does the following message will appear:

Bus time out exception caused by access at address xxxxxxxx

If this occurs, simply type any key to clear the error message.

REAL TIME TRACE DATA

PROBE gathers the execution of the program in real time into its high speed trace memory. This information can be displayed in several different forms. To demonstrate the different trace displays, go back to the breakpoint screen and inactivate all sticky breakpoints except 0. Then run the program again. Type:

bi12s

G<enter><enter>

Use the Trace Activity command to view the real time trace data. Note that this command prints a message describing the limitation of the trace information in this format.

ta

```
Data cycles were not matched with prefetched instructions for this display.
There are 3 major side effects of this:
1) Data cycles will probably not appear with the instruction that generated them.
2) Instructions following those that can cause a transfer of control may not have
actually been executed.
3) The trace display software may print the target of a jump instruction as the
wrong word (low vs.high) of a 4-byte instruction fetch.
Press any key to begin display
```

```
 0000082E :SD    READ    -    0000   \ftocm\Step
 00000830 :SD    READ    -    000A   \ftocm\Step+00000002
 000004D6 :SP  MOVE.l   (FFFFFFFC,A6),D0
 0000080C :SD    WRITE   - 0000000A   \ftocio\PutDecimal+0000012A
 000004DA :SP  ADD.l    D0,(A0)
 00000808 :SD    READ    - 00000832   \ftocio\PutDecimal+00000126
 000004DC :SP  MOVE.l   (FFFFFFF8,A6),A0
 0000080C :SD    READ    - 0000000A   \ftocio\PutDecimal+0000012A
 00000832 :SD    READ    -    0000   \ftocm\Fahr
 00000834 :SD    READ    -    0000   \ftocm\Fahr+00000002
 000004E0 :SP  MOVE.l   (A0),(FFFFFFFC,A6)
 00000832 :SD    WRITE   -    0000   \ftocm\Fahr
B 00000834 :SD    WRITE   -    000A   \ftocm\Fahr+00000002
 00000808 :SD    READ    - 00000832   \ftocio\PutDecimal+00000126
 000004E4 :SP  MOVE.l   (FFFFFFFC,A6),D0
 00000832 :SD    READ    -    0000   \ftocm\Fahr
 00000834 :SD    READ    -    000A   \ftocm\Fahr+00000002
 0000080C :SD    WRITE   - 0000000A   \ftocio\PutDecimal+0000012A
 0000080C :SD    READ    - 0000000A   \ftocio\PutDecimal+0000012A
```

In this trace display, instructions which were fetched by the 68020 and the memory reference cycles are shown in exactly the order in time in which they occurred. Under the source code, the assembly language is displayed. Source code and symbols from the symbol table are included in the display so that it is easy to understand. To simplify locating the breakpoint in the trace display, a "B" is shown in the first column for the cycle which caused the breakpoint.

Note the ADD.l instruction at address 4DA. The bus cycles which operate on the data for this instruction are several cycles lower at addresses 832 and 834. Two other instruction prefetches also precede these bus cycles. This makes following the execution of the ADD.l difficult. Now Pg/Up the trace data to address 5D4 which contains a BRA instruction. The instructions between this address and the instruction at address 622 were prefetched but not executed. Figuring this out mentally can become tedious. Now for a better way to view the trace data, type <Esc> and invoke the Trace Instructions command.

ti

```
 0000082E :SD    READ    -    0000    \ftocm\Step
 00000830 :SD    READ    -    000A    \ftocm\Step+00000002
 0000080C :SD    WRITE   - 0000000A   \ftocio\PutDecimal+0000012A
 000004D2 :SP  MOVE.l    (FFFFFFF8,A6),A0
 000004D6 :SP  MOVE.l    (FFFFFFFC,A6),D0
 0000080C :SD    READ    - 0000000A   \ftocio\PutDecimal+0000012A
 000004DA :SP  ADD.l     D0,(A0)
 00000832 :SD    READ    -    0000    \ftocm\Fahr
 00000834 :SD    READ    -    0000    \ftocm\Fahr+00000002
 00000832 :SD    WRITE   -    0000    \ftocm\Fahr
B00000834 :SD    WRITE   -    000A    \ftocm\Fahr+00000002
 000004DC :SP  MOVE.l    (FFFFFFF8,A6),A0
 00000808 :SD    READ    - 00000832   \ftocio\PutDecimal+00000126
 000004E0 :SP  MOVE.l    (A0),(FFFFFFFC,A6)
 00000832 :SD    READ    -    0000    \ftocm\Fahr
 00000834 :SD    READ    -    000A    \ftocm\Fahr+00000002
 0000080C :SD    WRITE   - 0000000A   \ftocio\PutDecimal+0000012A
 000004E4 :SP  MOVE.l    (FFFFFFFC,A6),D0
 0000080C :SD    READ    - 0000000A   \ftocio\PutDecimal+000001
```

In this format, PROBE uses a very sophisticated algorithm to simplify the interpretation of the program execution. This algorithm does the following:

1. Instructions which are prefetched but not executed are removed from the display so you do not have to guess which ones executed.
2. The memory reference cycles which are produced by the instructions are shown directly under the instructions which produced them. These cycles actually occurred many cycles later in time because of the 68020 prefetch queue. With the Trace Instructions command, you do not have to reconstruct the program execution yourself mentally.

Note the ADD.l instruction at address 4DA again. The bus cycles which operate on the data for this instruction are now displayed directly under the ADD.l. Now Pg/Up the trace data to address 5D4 which contains a BRA instruction. The prefetched but unexecuted instructions between this address and 622 are eliminated in this display.

During this program execution, the 68020 cache was disabled. This is the default case when PROBE software is started. Since the cache was disabled, the trace display can be "dequeued" with the Trace Instructions command (i.e., the instructions which were fetched but not executed are filtered out of the trace display). Next, try running the program with the cache enabled. The state of the cache is controlled by three conditions:

1. A hardware signal in the target
2. The least significant bit of the CACR register
3. A control field in the PROBE GO command.

Lets assume your hardware has the cache enabled. Next, since this demo program did not enable the cache, you must enable it by changing the least significant bit of the CACR register to 0. To do this, invoke the Register command and position the highlight to the CACR register. When PROBE prompts you for a value, type:

cacr |1

This "or's" the current CACR register and sets the least significant bit. Now start the program again and set the same breakpoint, but this time, enable the cache. This is done by typing the <TAB> character to get to the field in the GO command which lets you enable the cache. Type:

g<enter><tab><tab><tab><tab>y<enter>

When the breakpoint occurs, dump the trace again with the TI command. Note that the message and trace display you got previously with the TA command appears, instead of the TI format. This is because PROBE cannot dequeue the trace data when the cache is enabled. Instead, it shows fetched instructions and bus cycles in the order in which they occurred.

QUALIFIED TRACE DATA

PROBE also lets you qualify the real time trace data. This optimizes the useful trace information and ignores trace data you do not want to gather in real time. A qualified trace region is defined as a range of memory from which instructions are fetched. While in the region, the instructions and all memory reference cycles associated with the instructions are entered into the trace data. While not in the region, no trace data is gathered. PROBE disables the cache while in the qualified trace region even if it is enabled otherwise. Outside of this region, the 68020 cache is controlled by the same three conditions described earlier. This is called "dynamic cache control". It provides the benefits of running the target with the cache on for most of the time while giving PROBE the ability to trace through the qualified trace region. The trace region is set by typing:

tq

PROBE prompts you for the region, choose the responses shown here:

```
Start address: <00000000>   End address: <00000000>   Program space:<Both>
Don't care bits:<.... .... .... .... .... .... .... ....>
Enable trace qualification: [Yes] {Yes|No}
Enter new start address: [\ftocm\#22]
Enter new end address: [\ftocm\#23]
```

This sets a qualified trace region between linenumbers 22 and 23 in the module FTOCM. Now start program execution again with with the sticky breakpoint 0 still set and the cache still enabled.

g<enter><enter>

When the breakpoint is detected, dump the trace data with the TA command. You will see that only instructions between these two line numbers are traced.

```
*********************************************
 000007F0 :SD    WRITE   - 00000836   \ftocio\PutDecimal+0000010E
 0000054C :SP  MOVE.l    #00000020,D0
 0000054E :SP  MOVE.l    D0,(FFFFFFF4,A6)
 000007E4 :SD    READ    - 00000000   \ftocio\PutDecimal+00000102
 000007EC :SD    WRITE   - 00000000   \ftocio\PutDecimal+0000010A
 00000552 :SP  MOVE.l    (FFFFFFF4,A6),D0
 00000556 :SP  SUB.l     D0,(FFFFFFF8,A6)
 000007E8 :SD    WRITE   - 00000020   \ftocio\PutDecimal+00000106
 0000055A :SP  MOVE.l    (FFFFFFFC,A6),A0
 000007E8 :SD    READ    - 00000020   \ftocio\PutDecimal+00000106
 0000055E :SP  MOVE.l    (FFFFFFF8,A6),(A0)
 000007EC :SD    READ    - 00000000   \ftocio\PutDecimal+0000010A
 000007EC :SD    WRITE   - FFFFFFE0   \ftocio\PutDecimal+0000010A
 000007F0 :SD    READ    - 00000836   \ftocio\PutDecimal+0000010E
123.  *CelsiusTemp = *CelsiusTemp * 5;
B 00000562 :SP  MOVE.l    (FFFFFFFC,A6),A0
```

SEARCHING THE TRACE DATA FOR EVENTS

The PROBE/3 real time trace contains so much trace data that displaying it on the screen would take several minutes. In order to quickly find useful information in this trace data, the Trace commands have a built in text editor which will let you search for events in the various trace fields. While you are in any of the Trace commands, the DIALOG BOX looks like this:

```
Search address:<Any>      Space:<Any>      Verb:<Any>      Data:<Any>
<.... .... .... .... .... .... .... ....>
Begin search of trace: [No] {Yes|No}
__PgUp/PgDnArrows move within memory; <Tab> to next field; <Esc> to main menu
Cache disabled during execution.      Trace not qualified
Address          OP CODE      OPERANDS
```

PROBE prompts you to begin the search with:

Begin search of trace: [No] {Yes|No}

If you answer No to the "search" prompt, PROBE will provide a series of prompts to let you specify which fields in the trace data you want to search. If you answer Yes to the "search" prompt, PROBE starts searching from the top line of the current DISPLAY WINDOW until the end of the trace data. If you want to start searching from the start of the trace data, type Ctrl PgUp and the first data entered into the trace is shown in the DISPLAY WINDOW. When PROBE finds a match between the search fields you specified and a line in the trace data. the character "S" designates the line. You can continue the search by typing Yes in response to the "search" prompt again.

VIEWING UNASSEMBLED CODE AND LOGGING TO DISK

PROBE has an online unassembler to let you display memory as 68020 instructions. Let's assume you want to unassemble a block of memory and save it to a disk file. PROBE has a "log-file" capability that lets you direct all PROBE screen output to a disk file or line printer. This could also be used to save a debug session. To open a "log file" for logging data type:

xly

PROBE prompts you for the name of the file to open. Respond with:

log.tmp

Now you can display a page full of memory starting from the procedure COMPUTE by typing:

ucompute<enter><enter>

The following display appears. You can use the PgUp and PgDn keys to scroll through memory. Note the symbols from the symbol table are included with the instructions to help match the displayed code to your program. Since this demo is being done with the SOURCE versions of PROBE, the display also includes C source code.

```
Start address: <00000000>                    Memory space: < >
Display instruction words: <No>    Display operand addresses and values: <No>
Enter new start address: [ ]

118.      Compute (FahrTemp, CelsiusTemp)
000004FC          LINK          A6,#00000000
00000500          MOVEM.l       A2,-(A7)
122.      *CelsiusTemp = FahrTemp - 32;
00000504          MOVE.l        (00000008,A6),D0
00000508          MOVE.l        (0000000C,A6),A2
0000050C          MOVE.l        #00000020,D1
0000050E          SUB.l         D1,D0
00000510          MOVE.l        D0,(A2)
123.      *CelsiusTemp = *CelsiusTemp * 5;
00000512          MOVE.l        (A2),D0
00000514          MOVE.l        D0,D1
00000516          ASL.l         #2,D0
00000518          ADD.l         D1,D0
0000051A          MOVE.l          D0,(A2)
124.      *CelsiusTemp = *CelsiusTemp / ;
0000051C            MOVE.l      #00000009,D1
0000051E          MOVE.l        (A2),D0
00000520          JSR           (00000420)
00000526          MOVE.l        D0,(A2)
```

Now you can close the log file by typing:

xln

VIEWING FILES ON DISK

You can view files on disk with the PROBE view command. To display the file FTOCM.C on type:

vftocm.c

The PgUp and PgDn keys let you scroll through the file. When you leave the View command and then reenter it, file is positioned on the screen in exactly the same place as you left it. Up to 10 files can be viewed with the View command and pointers are maintained to your view of the file.

SAVING INITIALIZATIONS AND BLOCKS OF MEMORY

PROBE lets you save the conditions of a debugging session so that you can easily set up PROBE the next time you use it. This command is started by typing:

is

PROBE prompts you for the name of the file in which to save the initializations.

Enter filename for initialization information:[ ]

You could also save a block of memory from the target or MAP RAM to a disk file which you could later reload. This is done with the Xfer Block-save command.

xb

CHAPTER 5 COMMAND REFERENCE

INTRODUCTION TO COMMON COMMAND DEFINITIONS

This chapter contains a detailed description and examples for each PROBE command. The commands are listed alphabetically. This chapter also defines the common terms that are used in the PROBE command definitions.

VALUE

A value is a 32 bit quantity that can be represented by any of the items shown in table 5-1.

TABLE 5-1
EXAMPLE OF VALUE DEFINITIONS

Value represented as:	Examples
a symbol name (it's address)	MAIN
a 32 bit numeric hex constant	12345678
a 32 bit numeric decimal constant	12345678T
a register name (it's contents)*	D0 (see note below)
an ascii character in quotes	'A'
a dereferenced memory location	(described later)

*Note: A value which matches a register name is interpreted as the register name. Register names are specified in the R command. If you want a hex value instead, precede the hex value with a 0.

BASE

When you enter values into PROBE commands, they are interpreted as hex unless you specify another numeric base. You have the following choices:

Subscript the value with t for a base of ten (i.e. decimal)
Example: 10t means 10 decimal.

Put single or double quotes around the value to interpret it as an ASCII string.
Example: 'This is a string" represents 16 bytes of data.

ADDRESS

An address is simply a value.

EXPRESSION

An expression is a value calculated by combining a series of values with operators.

+ , - , * , / , ~, & , | , % ^

Normal precedence of operators as defined in the C language is assumed: (*, / ,% ,&) are higher than (+, -, |) and evaluation proceeds left to right on operators with equal precedence. Precedence may be overridden by the use of parenthesis. Table 5-2 explains each operator and lists them in order or precedence.

TABLE 5-2
DEFINITION AND PRECEDENCE OF OPERATORS

Operator	Definition
Highest precedence	
-	2's complement
~	bit by bit negation
Next highest	
*	multiplication
/	division
%	modulus (remainder)
&	bit by bit and
Lowest	
+	addition
-	subtraction
|	bit by bit inclusive or
^	bit by bit exclusive or

EXAMPLES: Assuming the following values in memory, here are several expressions and their resulting values.

The symbol main is 20000000
Memory at 20000000 is AABBCCDD
Memory at 20000005 is 11223344
Memory at 11223344 is EEFF7788
Memory at 11223354 is 99005566

Expression	Value
-main	E0000000
~main	DFFFFFFF
main*2	40000000
main/2	10000000
main %10	0
[illegible]	[illegible]
main+5	20000005
main-5	1FFFFFFB
main|FF	2000000FF
main^F0000000	D00000000

BOOLEAN EXPRESSIONS

Boolean expressions use boolean operators and result in one of two boolean values. The PROBE boolean operators result in a value of FFFFFFFF (or non zero) if the result is TRUE and a value of 00000000 if the result is FALSE. The boolean operators may be joined with the '&' and '|' operators for the boolean AND and OR functions.

TABLE 5-3
DEFINITION AND PRECEDENCE OF BOOLEAN OPERATORS

Operator	Definition
<	less than (unsigned)
<s	less than (signed)
<=	less than or equal (unsigned)
<=s	less than or equal (signed)
=	is equal to
==	is equal to
<>	is not equal to
!=	is not equal to
>=	greater than or equal (unsigned)
>=s	greater than or equal (signed)
>	greater than (unsigned)
>s	greater than (signed)

EXAMPLES: Here are some boolean expressions and a description of how the operators work.

Boolean Expression	Description
D0>D1	If the contents of data register 0 is greater than data register 1, then the result is true.
[123]=[456]	If the contents of memory at address 123 are equal to the contents of memory at address 456, then the result is true.
Procedurea <PC	If the address represented by the symbol Procedurea is less than the program counter, then the result is true.
[Celsius] <10t	If the contents of the variable Celsius is less than decimal 10, then the result is true.

DEREFERENCED MEMORY

The contents of a memory location may be used as a value in an expression. This is commonly referred to as dereferencing memory. The value is pointed to by an address expression which is defined as follows:

[expression].size

The .size may be omitted or may be:

SIZE	DEFINITION
B	use the byte at the specified address -** Default **
W	use the word (16 bits) at the specified address.
L	use the long (32 bits) at the specified address.

EXAMPLES: Assuming the following values in memory, here are some address expressions and their resulting values as interpreted by PROBE.

The symbol main is 20000000
The bytes of memory at 20000000 are AA, BB, CC, DD
The bytes of memory at 20000005 are 11, 22, 33, 44
The bytes of memory at 11223344 are EE, FF, 77, 88
The bytes of memory at 11223354 are 99, 00, 55, 66

AddressExpression	Value (in hex)
main+5	20000005
[main+5].b	00000011
[main+5].w	00001122
[main+5].l	11223344
[[main+5].l]	000000EE
[[main+5].l].w	0000EEFF
[[main+5].l].l	EEFF7788
[[main+5].l+10]	00000099
[[main+5].l+10].w	00009900
[[main+5].l+10].l	99005566

MEMORY SPACES

The 68020 has 8 memory spaces. Commands which display or change memory can do so in any of the 68020 memory spaces. If the hardware design of the target system has separately decoded the memory spaces of the 68020, then you must pay attention to the memory space you want to display or change. *If your hardware does not do this decoding, then you can ignore the memory space and use the defaults as supplied by PROBE.* Since the memory spaces of the 68020 are not often used or changed, the portion of the commands which select different memory spaces is a field which you must <TAB> to in order to change. Here are the memory spaces of the 68020.

Memory space	Description
0	Decoded as 0
UD or 1	User data
UP or 2	User program
UR or 3	User reserved
4 or 4	Decoded as 4
SD or 5	Supervisor data
SP or 6	Supervisor program
CPU or 7	CPU data

USING MEMORYSPACE WHEN DEREFERENCING MEMORY

If you are entering an address expression into a field which is dereferencing memory, you must specify a memory space for the address expression if it is to be different than the default used for the command. This adds to the definition of a dereferenced memory location as follows:

[expression].size:space

EXAMPLES: This address expression points to the Long word located at the address represented by the symbol main in the user data space.
[main].l:ud

This address expression points to the byte at the address contained in register A0 in the User Program space. Note that no .b need be specified for byte since this is the default.

A0:up

FORMAT FOR DESCRIBING PROBE COMMANDS

In this chapter, the MENU BOX, DISPLAY WINDOW, and DIALOG BOXES produced by the commands are shown. Along the perimeter of the screens you will find numbers in circles. In the text you will find the corresponding numbers in circles along with a description of the screen information.

The prompts PROBE provides you in DIALOG BOXES are shown in *italics* in this chapter.

In the text, a keystroke is specified as:

<keyname>

For example, <Enter> means type the key labeled Enter on your keyboard.

Some <keynames> require two keys. For example <*AltKey*> means hold down Alt and type the keyname.

USING WILDCARD CHARACTERS WHEN ACCESSING FILENAMES

PROBE interprets file specifications in the same manner as DOS on the AT. A filespec is defined as:

[drive] [path] [name of file]

If not specified, the default drive/path is used. PROBE lets you use wildcard characters in any command which prompts you for a filename. If you cannot remember the name of the file you want to specify or you can only remember part of the filename, then use the wildcard character *. If you are familiar with DOS, the * works in exactly the same way as it does in the DIR command.

EXAMPLES: To tell the PROBE command to display all filenames in the current directory, type:
.

To specify all files in the directory \MAIN\DEMOS type:
\MAIN\DEMOS*.*

To specify all files in the current directory with a .HEX extension type:
*.hex

To specify all files in the current directory which start with the letter A type:
A*.*

SUMMARY OF 68020 PROBE COMMANDS

REAL TIME ANALYSIS AND CONTROL COMMANDS

1st key	*Sub key*	*Description*
G		Start program execution using current or new address
		Set non sticky breakpoints from breakpoint screen
B	D	Define/display sticky breakpoints from breakpoint screen
	I	Inactivate sticky breakpoints
	A	Activate sticky breakpoints
T	I	Display real time trace data dequeued
	A	Display trace data with prefetch not dequeued
	Q	Set qualified trace regions
	R	Display raw trace data
	S	Save trace data to disk as ascii text.
	U	Save trace data to disk in a binary format

INTERROGATION AND MODIFICATION

1st key	*Sub key*	*Description*
U		Display a block of memory in assembly language
A	R	Replace memory with assembly language
	I	Insert assembly language instructions
D	B	Display and change bytes memory
	W	Display and change words in memory
	L	Display and change long words in memory
	S	Display and change single address
	F	Display and change floating point data
	M	Display and change memory mapping
	N	Display and change noverify memory condition
R		Change or display 68020, 68881, MMU registers and flags
X	B	Block save memory to disk file
	C	Compare two blocks of memory
	F	Find string in memory
	M	Move a block of memory
	S	Set block of memory to new values
	L	Log PROBE output to disk
N		Display high level language procedure nesting

UTILITY COMMANDS

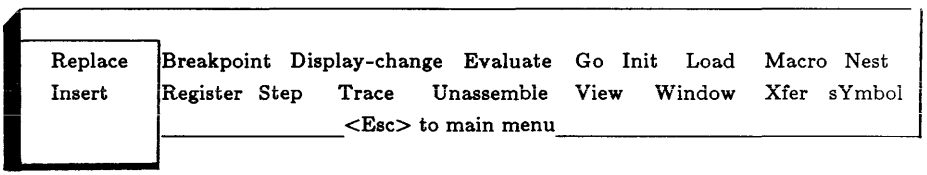
1st key	*Sub key*	*Description*
L	P	Load target system program and symbol table
	O	Set Load options
I	L	Load initial set up conditions for PROBE
	S	Save initial set up conditions for PROBE
S	A	Single step program via assembly language
	S	Single step program via source code statements
Y	D	Display/change symbol name or value
	R	Remove symbol or block of symbols
	C	Ignore case for symbols
	M	Define default modulename prefix
	L	Selectively load symbols for specified modules
	S	Selectively source step through modules
	A	Assign modulenames to source files
E		Evaluate an expression in several bases
Q		Return to operating system
W	D	Define/edit a window on the screen
	L	Load window definitions from disk
	S	Save window definition to disk file
	R	Remove or delete the definition for a window
V		View and scroll through disk files
M	D	Define a macro
	E	Edit a currently defined macro
	C	Define a macro for conditional execution
	L	Load previously defined macros from a file
	S	Save currently defined macros to a file
	R	Remove a macro
H	C	Display target processor clock speed
	R	Reset the target system hardware
	I	Disable/Enable interrupt requests from the target
	D	Disable/Enable DMA requests from the target
	H	Disable/Enable Halt signal from the target
	B	Disable/Enable stoppage of execution after breakpoint
	W	Disable/Enable watch-dog-timeout of target system
	L	Looping read-write test of target system

ASSEMBLE COMMAND

PROBE has an on-line symbolic assembler that lets you put 68020 assembly language directly directly into memory. This command is invoked from the MENU BAR by typing:

A for Assemble

The subcommands for the Assemble command now appear in a MENU BOX and the screen looks like this.

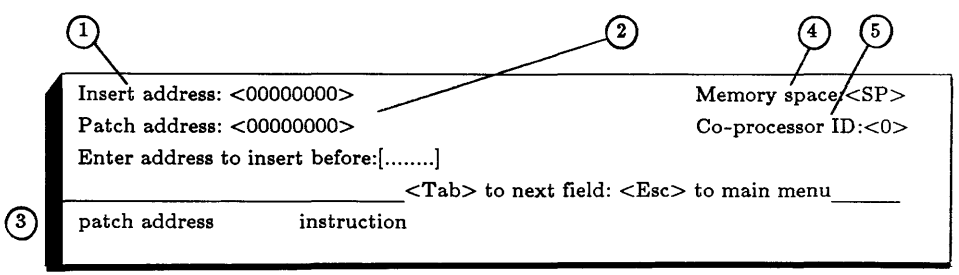


The subcommands for Assemble are:

Command	Sub command	Operation
Assemble	Insert	Insert code before instruction
	Replace	Replaces code with new code

INSERTING CODE

After invoking the Insert subcommand, the following screen appears:



① The first DIALOG BOX prompts you for the insert address:

Enter address to insert before: [ ]

The insert address can be any type of expression followed by <enter>. A JMP instruction to the patch address is placed by the Assemble Insert command at the *insert before*.address. The instructions which were previously at the *insert before* address are moved to the end of the patch area. NOP's are inserted if the moved instructions are a different size than the JMP.

② The start of the patch area is specified in the next prompt:

Enter address for patch: [ ]

The new instructions will be assembled into memory starting at this address.

③ Next, you are prompted to start assembling instructions.

Instructions [ ]:

The instructions which are initially typed into the DIALOG BOX are transferred to memory and then unassembled into the DISPLAY WINDOW after each <enter>. The PROBE edit keys let you make corrections to the instructions in the DIALOG BOX.

④ Typing <TAB> while in the Insert subcommand will bring up the following DIALOG BOX:

Memory space: < > {0|UD|UP|UR|4|SD|SD|CPU}

The memory space into which the instructions are assembled can be selected with this prompt. The default memory space is UP (user program) or SP (supervisor program) depending on the initial state of the flags in the 68020. Typing additional <TAB>s will recall the other DIALOG BOXES for this command.

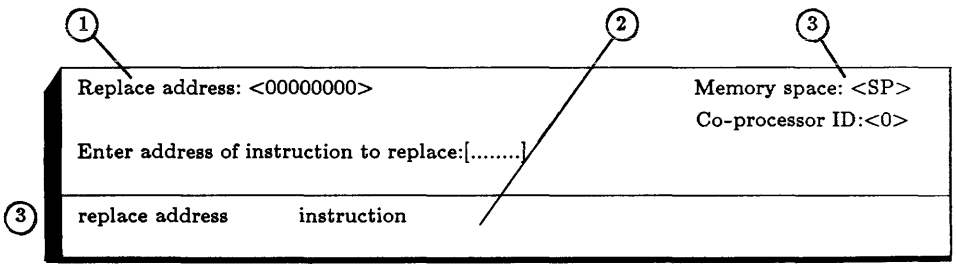
⑤ The assembler will work for Co-processors as well as the 68020 cpu. If there is more than one Co-processor of the same type in the system, the ID of Co-processor must be identified to PROBE. Type <Tab> to get the following prompt.

Enter new co-processor ID:[ ]

Co-processor ID's can be from 0 to 7. For the case of a single Floating point Co-processor, 68881 , memory management unit,68851, or one of each, PROBE can automatically determine the ID. You may ignore the ID in these cases. If you have two floating point Co-processors, however, a floating point instruction must include the ID to determine which processor to use.

REPLACING CODE

After invoking the Replace subcommand, the following screen appears.



(1) The first DIALOG BOX prompts you for the replace address

Enter address of instruction to replace: [ ]

The replace address can be any type of expressing followed by <enter>. Instructions at this address are replaced.

(2) Next, you are prompted to start assembling instructions.

Instructions:[ ]

The instructions which are initially typed into the DIALOG BOX are transferred to the DISPLAY WINDOW after each <enter>. The PROBE edit keys let you make corrections to the instructions in the DIALOG BOX.

(3) Typing <TAB> while in the Replace subcommand will bring up the prompts for Memory Space and Co-processor ID as described in the insert subcommand.

NOTES ON THE 68020 PROBE LINE ASSEMBLER

The following notes apply to the assembly language which which is understood by the 68020 PROBE standard line assembler.

1. Standard 68020 assembly language mnemonics are used.
2. The assembler will automatically assemble short jumps and calls depending on the displacement of the destination address.
3. When a byte, word, or long size cannot be determined by the operand, the data type of the operand must be specified by .b, .w or .l.

EXAMPLES OF THE ASSEMBLE COMMAND

Insert instructions in memory starting at location MAIN in the current default module with new code starting at location IOPROC in module DOIO. The key sequence for this is:

ARMAIN<enter>\DOIO\IOPROC<enter>
68020 instructions<Esc>

Replace instructions in memory starting at location FOO in the current default module with new code:

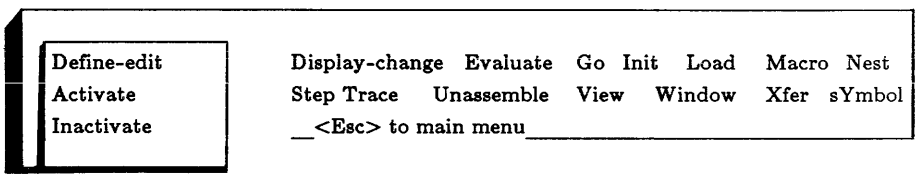
ARMAIN
68020 instructions<Esc>

BREAKPOINT COMMAND

The Breakpoint command lets you define, delete and activate sticky breakpoints. If sticky breakpoints are active, they are automatically inserted when the Go command is executed. The Breakpoint command is invoked from the MENU BAR by typing:

B for **B**reakpoint

The subcommands for the Breakpoint command appear in a MENU BOX and the screen looks like this:

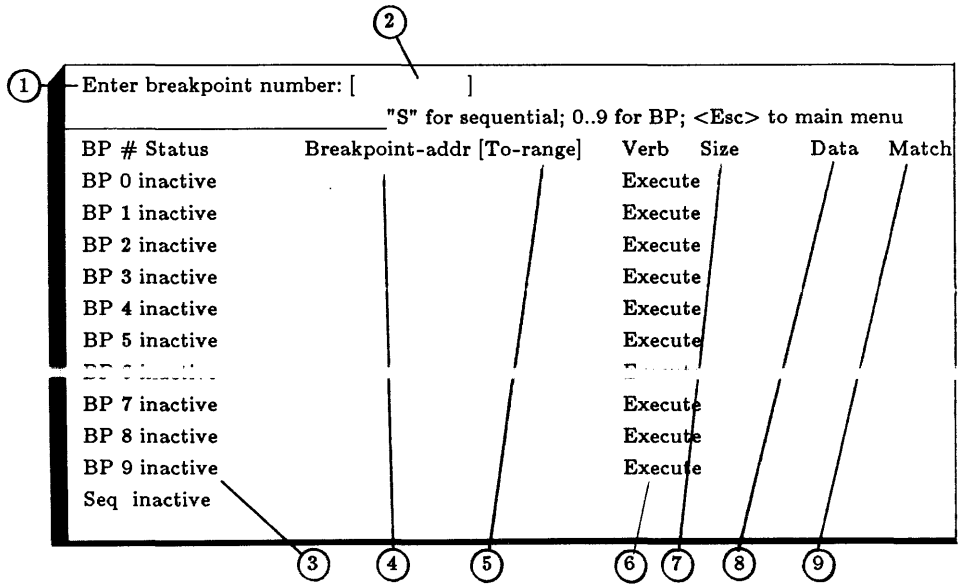


The ubcommands for Breakpoint are:

Command	Sub command	Operation
Breakpoint	**D**efine	Define/change breakpoint
	Activate	Enable breakpoints during Go
	Inactivate	Do not enable breakpoint during Go

DEFINING AND EDITING BREAKPOINTS

When the Define subcommand is selected, the DISPLAY WINDOW shows a summary of current condition of the sticky breakpoints.. This is sometimes referred to in this manual as the Abbreviated Breakpoint Summary.



(1) The first DIALOG BOX prompts you for the breakpoint number:

Enter breakpoint number: [ ]

PROBE lets you define up to 10 sticky breakpoints with breakpoint numbers from 0 to 9. In addition, a breakpoint number may be S (which stands for sequential) is described later. To make it easy to remember which breakpoints are already defined, an abbreviated summary of the status of all 10 sticky breakpoints is shown in the DISPLAY WINDOW. A short explanation of these fields is given here. A more thorough explanation is given later.

(2) This field is the Breakpoint number.

(3) If a breakpoint has not previously been defined, its default status is shown as inactive.

④ This is the address of the currently defined sticky breakpoints.

⑤ This is the end address of the currently defined sticky breakpoint if it is a range breakpoint.

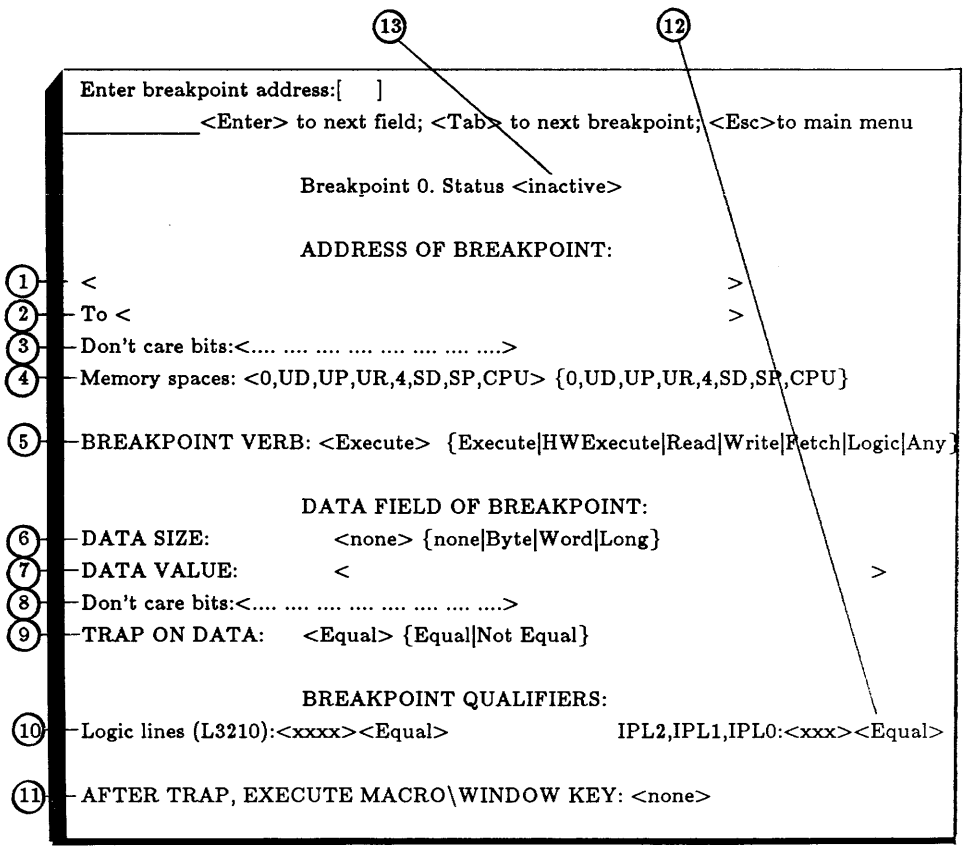
⑥ This is the verb of the currently defined sticky breakpoints. The default is Execute.

⑦ This is the Size of the data field if the data bus is included in the breakpoint.

⑧ This is the Data field if the data bus is included in the breakpoint.

⑨ This is the Match or No match condition if the data bus is included in the breakpoint.

Once the the breakpoint number has been selected, the following DISPLAY WINDOW appears.



① The first DIALOG BOX prompts you for the address for the breakpoint

Enter breakpoint address: [ ]

You can use an expression or absolute number for the address.

② The next prompt lets you enter the TO address if the breakpoint is for a range of addresses:

Enter new end address:[ ]

You may enter any type of address expression. The end address may also be of the form:

+ number

In this case the end address becomes start address+number. If this field has previously been set to other values and you want to clear the field, type <space><enter>.

Rather than go through the remainder of the Breakpoint definition prompts, you may be finished at this point. If this breakpoint has not been previously defined, the remainder of the defaults in this breakpoint screen set it to a simple Execute breakpoint. You may want to move on to define another breakpoint. You may want to continue the breakpoint definition. The following is a summary of the keys you could type at this point in the definition (or at any point).

Key	What you get
<ESC>	Accept current screen, and go to MENU BAR
<TAB>	Accept current screen and start defining another breakpoint
<enter>	Move to next prompt for this breakpoint. Do this if you want to set more breakpoint fields but this is not a range breakpoint
address	Define this breakpoint as a range breakpoint and move on to next prompt

③ This field lets you mask out bits in the address field of the breakpoint.

Don't care bits:<.... ....>

This is useful if the target does not decode the entire address space and ignores some address lines. It is also useful instead of the range breakpoint for some applications. You enter an X for each address bit position you want to mask and enter a period (.) for each address bit position you want to trigger on. The initial default is all periods. The data you enter for this prompt is left justified since it is most common to mask out the upper address bits.

④ The next prompt lets you set the breakpoint on any combination of the 8 different 68020 memory spaces. If the breakpoint is to cover several memory spaces, separate them with a comma as shown with the default. If VERB is Execute or HWExecute, then only UP and SP are used for this field and all others are ignored. If your hardware does not decode memory spaces, then take the default which is all memory spaces by simply typing <enter>.

Enter memory space list: [ ] {0,UD,UP,UR,4,SD,SP,CPU}

⑤ The next prompt sets the breakpoint verb:

Enter breakpoint verb: [ ]
{Execute|HWExecute|Read|Write|Fetch|Logic|Any}

The breakpoint verbs are defined as follows:

VERB	VERB DEFINITION
Execute	Instruction execution breakpoint via software interrupt instruction. May only be set in ram memory. This type of breakpoint replaces the target code with the 68020 BKPT #7instruction. See Appendix C to change the BKPT 7 to another vector.
HWExecute	Instruction execution breakpoint via POD. May be set in ram or prom memory.
Read	Breakpoint on Read of memory address or range of memory
Write	Breakpoint on Write to memory address or range of memory
Fetch	Breakpoint on read from UD or SD memory address or range of memory.
Any	Breakpoint on Any access (i.e.Read, Write, or FETCH) of memory address or range of memory addresses.

⑥ The breakpoint can be further qualified with the DATA FIELD. This is done by first selecting the size of the data field with this prompt:

Enter type of data for breakpoint: [ ]{none|Byte|Word|Long}

The size of the data field can be 1,2 or 4 bytes by choosing Byte, Word or Long. If you do not want to include the DATA FIELD in the breakpoint, then type <enter> to select the default which is <none> and the next two prompts will be ignored.

⑦ If you select one of the types for the DATA FIELD, then you must specify the DATA VALUE which will cause the breakpoint with this prompt. The DATA VALUE can be an expression.

Enter data value for breakpoint: [ ]

⑧ This field lets you mask out bits in the DATA FIELD of the breakpoint.

Don't care bits:<.... ....>

This is useful if you are looking for a bit field such as a flag or an ASCII character in the breakpoint. You enter an X for each data bit position you want to mask and enter a period (.) for each data bit position you want to trigger on. The initial default is all periods. The data you enter for this prompt is right justified.

⑨ The breakpoint can be selected to trap on the data being equal or not equal to the DATA VALUE with this prompt.

Breakpoint on datavalue:[ ] {Equal|Not Equal}

This is useful when you are looking for a change of state of a variable or a bit in memory.

⑩ This field lets you qualify the breakpoint further with the external Logic PROBES. The match condition can be set to 1, 0, or Don't care for each line. See Appendix H for more information on these Logic PROBES. When a match occurs on this field, the other breakpoint fields on this screen are enabled. The prompt for this field is:

Enter qualifier bits ("0"|"1"|"x"): [ ]

Once a value has been put into the logic line file, a second prompt appears:

Breakpoint on value:[Equal] {Equal|Not Equal}

This lets you look for an equal or not equal condition on the Logic lines.

⑪ An additional option lets you invoke a PROBE macro or window when the breakpoint has occurred. This is done by specifying an *AltKey* (i.e. hold down Alt and type any key) in response to this prompt.

Enter Macro/Window <AltKeyname>: [ ]

If a macro is defined for the specified *AltKey*, it will execute when this breakpoint is executed. If a Watch Window is defined for the specified *AltKey*, then it will pop up after the breakpoint is detected.

⑫ This field lets you qualify the breakpoint further with the IPL (interrupt priority level) lines on the 68020. The match condition can be set to 1, 0, or Don't care for each line. When a match occurs on this field, the other breakpoint fields on this screen are enabled. The prompt for this field is:

Enter qualifier bits ("0"|"1"|"x"): [ ]

Once a value has been put into this field, a second prompt appears:

Breakpoint on value:[Equal] {Equal|Not Equal}

This lets you look for an equal or not equal condition on the IPL lines.

⑬ This field lets you activate or inactivate a breakpoint:

Enter status of this breakpoint: [ ] {active|inactive}

A <TAB> typed during the breakpoint definition will transfer you to the prompt which lets you activate the breakpoint or back to the first screen in the Breakpoint command so you can define another breakpoint. You can also use the cursor keys to move you directly to the field in this screen which you want to change. If the field has a value in it and you want to make it blank, simply type <space><enter> to clear the field.

BREAKPOINT DETECT OUTPUT

For Breakpoint verbs; Read, Write, Fetch, and Any, a breakpoint output detect pulse is available to trigger an external device such as a Logic Analyzer or Scope. This Breakpoint Detect output signal is produced in the POD is is avalialbe through the Logic PROBES. See Appendix H for more information.

BREAKPOINT RESTRICTIONS

The following restrictions apply to the breakpoints whether set by the Breakpoint command or the Go command.

1. Since PROBE uses the POD, Breakpoint/Trace boards, and software breakpoints to set breakpoints, you can set up to 24 breakpoints. Since the BP screen lets you define only 10 sticky breakpoints, the remainder of the 24 available breakpoints must be set when you use the Go command. The maximums for each type of breakpoint is shown in this table.

Type	Maximum
Execute	16
HWExecute	4
Read|Write|Fetch|Any|Logic	4

2. Execute and HWExecute breakpoints may only be set in UP or SP memory spaces. All other spaces are ignored for this type of breakpoint even if they are specified.
3. Breakpoints which specify a Data Size have the following restrictions:

   Data Size cannot be selected for Execute or HWExecute verbs. The Data Size may not be larger than the Bus Size. The Bus Size is set for a block of memory with the **D**isplay-change **M**ap command and matches the physical size of the target bus over a range of addresses. In addition, if Data Size is specified, the breakpoint address must start on the address boundary and end 1 byte before the address boundary shown in the table below:

TABLE 5-4
Data Size vs Address Boundary

Address Boundary A1A0	DATA SIZE Byte	Word	Long
0 0	ok	ok	ok
0 1	ok	ok(if 32bit bus)	not allowed
1 0	ok	ok	not allowed
1 1	ok	not allowed	not allowed

For example, if DATA SIZE is word, then the breakpoint must be set on a word boundary. That is, A0 = 0. If DATA SIZE is long, then the breakpoint must be set on a long boundary. That is, A0 = 0, A1 = 0.

4. Range breakpoints have the following restrictions:

   a. They may not cross a 1 megabyte boundary.

   b. Verbs are limited to Read, Write, Any, or Fetch.

   c. Range breakpoints must start on the address boundary which is the same size (or larger) as the Bus Size. The following table illustrates this:

TABLE 5-5
Bus Size vs Address Boundary for Range Breakpoints

Address Boundary A1A0	BUS SIZE Byte	Word	Long
0 0	ok	ok	ok
0 1	ok	not allowed	not allowed
1 0	ok	ok	not allowed
1 1	ok	not allowed	not allowed

Range breakpoints which also include Data Size must meet the same requirements as a normal range breakpoint. In addition, for

range breakpoints which include Data Size, the Data Size must exactly match the Bus Size.

BREAKPOINT COMMAND EXAMPLES

Define and activate breakpoint number 0 which detects a write to the range of addresses starting at 1000 and ending at 14f6 when the value written is 1234. Execute macro *Alt*J after the breakpoint. Assume the BUS SIZE as set by the Display Map command is Long. The key sequence to enter this breakpoint is shown followed by the Abbreviated Breakpoint Summary.

BD01000<enter>+4f6<enter><enter><enter>WW1234<enter><enter><enter><enter><enter><AltJ>A<ESC>

BP # Status	Breakpoint-addr	[To-range]	Verb	Size	Data	Match
BP 0 active	1000	To +4f6	Write	Long	1234	Equal

Define and activate breakpoint number 1 which traps executing an instruction at location PROMPT in module MAIN. Assume the instruction is in ram memory:

BD1MAIN\PROMPT<Up arrow>A<Esc>

BP # Status	Breakpoint-addr	[To-range]	Verb	Size	Data	Match
BP 1 active	main\prompt		Execute			

Define and activate breakpoint numbers 2,3,4, and 5 for executing instructions at line numbers 20, 40 108 and 212 in the current module. Assume that the breakpoints are in eprom memory.

BD2#20<enter><enter><enter><enter>H<Tab>Y
3#40<enter><enter><enter><enter>H<Tab>Y
4#108<enter><enter><enter><enter>H<Tab>Y
5#212<enter><enter><enter><enter>H<Tab>Y<Esc>

BP # Status	Breakpoint-addr [To-range]	Verb	Size	Data	Match
BP 2 active	#20	HWExecute			
BP 3 active	#40	HWExecute			
BP 4 active	#108	HWExecute			
BP 5 active	#212	HWExecute			

Define and activate sticky breakpoint 6 to detect a FETCH from the lower 4k bytes of memory. Execute the macro assigned to the key AltF when this breakpoint occurs.

BD60<enter>+3FF<enter><enter><enter>F<enter><enter><enter><enter>*Alt*FA<Esc>

BP # Status	Breakpoint-addr [To-range]	Verb	Size	Data	Match
BP 6 active	0 To +3ff	Fetch			Equal

Define and activate sticky breakpoint 7 to trap if the data pattern AAAA is written to location 1000 and bit 0 of the PROBE Logic lines is equal to 1.

BD71000<enter><enter><enter><enter>W<enter>0<enter><enter><enter>A<Esc>

Define and activate sticky breakpoint 8 to trap Any type of access between locations FFFFF000 and FFFFFFFF. Set the trap for memory spaces Supervisor program and User program only.

BD8FFFFF000<enter>+FFF<enter><enter>UP,SP<enter>A<enter><enter><enter><enter>A<Esc>

BP # Status	Breakpoint-addr [To-range]	Verb	Size	Data	Match
BP 8 active	fffff000 To +fff	A			Equal

Define and activate a sticky breakpoint 9 to trap Any access to the range of memory between 0 and 000FFFFF with PROBE Logic Line 0 equal to a 1.

BD90<enter>0FFFFF<enter><enter><enter>A<enter>1<enter><TAB>Y<Esc>

Go back and redefine breakpoint 9 to make it trap executing an instruction at location MAIN.

BD9MAIN<enter><space><enter><enter><enter>E<enter>X<enter><Esc>

Assume the following code is in memory and the breakpoint defined as shown below. The breakpoints show some of the restrictions as described earlier in this command.

```
400    move.l #1234,(1000)
408    move.l #5678,(1002)
410    bra    400
```

BP # Status	Breakpoint-addr [To-range]	Verb	Size	Data	Match
BP 0 active	1000	W	L	1234	Equal
BP 1 active	1002	W	L	5678	Equal
BP 2 active	1004	W	W	5678	Equal
BP 3 active	1000 To 100F	W	W	5678	Equal
BP 4 active	1000 To 100F	W	L	5678	Equal

BP 0 will work ok. BP 1 will give the error "BP1 Long data must start on Long word boundary (A1A0=00)" when the Go command is typed. Breakpoint 2 will work ok. Breakpoint 3 will give the error "Range breakpoint Data Size must match Bus Size". Breakpoint 4 will not give an error but will not trap a write of 5678 to location 1002 because it is set to trigger on a Long not Word write.

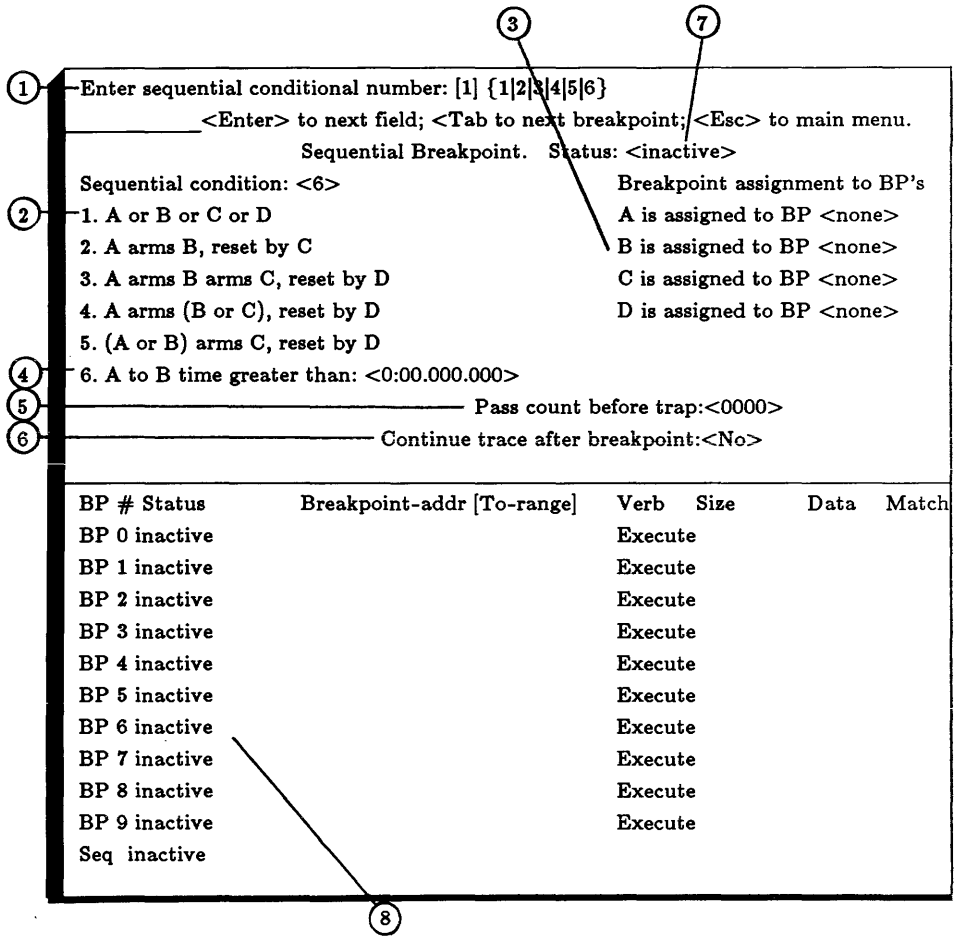
SEQUENTIAL, PASS COUNT AND TIMEOUT BREAKPOINTS

In addition to the sticky breakpoints described previously, a special kind of sticky breakpoint can be defined by selecting "S" as the breakpoint number in the Breakpoint Define command. On this breakpoint screen you can define

sequential breakpoints
timeout breakpoints
real time breakpoint pass counter
continue real time trace after breakpoint

A sequential breakpoint is the result of a sequence of operations between other sticky breakpoints. A timeout breakpoint occurs when the time between two other breakpoints exceeds a prespecified timer value. The real time breakpoint pass counter lets you count the number of breakpoints in real time before program execution is stopped. The trace after trigger condition lets you continue the real time trace after the breakpoint has occurred.

When a sequential breakpoint is being defined, this DISPLAY WINDOW appears.



(1) The first DIALOG BOX prompts you to select the type of breakpoint sequence you want to cause a trap :

Enter sequential condition number: [ ] {1|2|3|4|5|6}

There are 6 different types of sequential breakpoint conditions. As you can see from the sequential conditions, four PROBE hardware breakpoints can be involved in the breakpoint sequence. These are labeled A, B, C, and D.

② The sequential condition definitions are described in this area. Arms in the breakpoint description means that the first breakpoint enables detection of the second breakpoint. Reset in the breakpoint description means that the previous Arms condition is cleared. Thus if the reset breakpoint is detected after the arming breakpoint but before the armed breakpoint, then all bets are off and the sequence must start all over. If you do not want the reset breakpoint, leave D blank.

③ Next you must assign A, B, C, and D to PROBE breakpoints. This is done with the following series of prompts.

Assign breakpoint A to BP [ ] {0|1|2|3|4|5|6|7|8|9}
Assign breakpoint B to BP [ ] {0|1|2|3|4|5|6|7|8|9}
Assign breakpoint C to BP [ ] {0|1|2|3|4|5|6|7|8|9}
Assign breakpoint D to BP [ ] {0|1|2|3|4|5|6|7|8|9}

The A, B, C, and D items are assigned sticky breakpoint numbers (0 to 9). When a sticky breakpoint is assigned to A, B, C, D then the sticky breakpoint is not also used separately as a sticky breakpoint and is only used in the sequence. If A, B, C, or D are not assigned to a breakpoint (shown as none), then they are null terms in the breakpoint sequence. When the breakpoints 0 thru 9 are assigned to a sequence, they must all fall within one of the groups below:

All must have a verb of Execute or HWExecute.
All must have a verb of Read, Write, Fetch, Any, Logic.

In the first group, execution stops at each breakpoint and the sequential condition is checked with software. In the second group, sequential detection is done in real time. The error checking to determine if the all breakpoints in a sequence are in one of these two groups is not done until program execution actually starts with the Go command. This lets you define which sequence you want to use in advance of defining the breakpoints in the sequence.

④ Sequential breakpoint type 6 in this display is a special case. This is a timeout breakpoint. After this sequence is selected, the following prompt appears:

Enter new time-out time as <min:sec.msec,uses>:[ ]

A timer value is entered into this field. The timer value can be specified as follows:

Min:Sec
Min:Sec.Millisecond
Min:Sec.Millisecond,Microsecond
Sec.Millisecond
Sec.Millisecond,Microsecond
Millisecond,Microsecond
Microsecond

Data is interpreted as follows when entered into this field.
Numbers to the left of a colon are interpreted as minutes.
Numbers to the right of a comma are interpreted as microseconds.
Numbers to the left of a period are interpreted as seconds.
Numbers to the right of a period are interpreted as milliseconds.

The timeout breakpoint occurs when the time between breakpoints assigned to A and B exceeds the timer value while the program is executing. A timer is started when the breakpoint assigned to A is encountered. If the timer reaches the timer value before the breakpoint assigned to B is encountered, then program execution is stopped. If the breakpoint assigned to A occurs a second time before the timer reaches the timer value, the timer is reset and starts over again. The breakpoints assigned to A and B cannot have Execute or HWExecute verbs (use Fetch instead). For timeout breakpoints, PROBE forces the pass count to 1 and the trace after trigger to no.

⑤ In addition to setting a trap on a sequence of breakpoints, the PROBE has a real time pass counter. You can trap on the Nth occurrence of a breakpoint sequence. By not assigning B, C, or D, an individual breakpoint can have a pass count.

Enter new pass count: [ ]

If you type <enter> without a pass count, the pass count will not be changed. Pass count can have values from 1 to FF.

⑥ Sometimes it is desirable to cause the real time trace to continue after the breakpoint has occurred. The trace will continue for 128 cycles after the breakpoint is detected. This option is selected with this prompt:

Continue trace after breakpoint:[No]{Yes|No}

⑦ You can activate or inactivate the S breakpoint with the following prompt

Enter status of this breakpoint: [ ] {active|inactive}

⑧ The process of making the breakpoint assignments to A, B, C, and D is made easier since an abbreviated summary of the current sticky breakpoint definitions is also shown in this screen for reference.

SEQUENTIAL BREAKPOINT COMMAND EXAMPLES

Define and activate the S breakpoint to detect the occurrence of sticky breakpoint 5 followed by 2. If, however, breakpoint 3 occurs between 5 and 2, restart the sequence of looking for breakpoint 5 again. Do not stop program execution until this sequence happens 255 times. Continue the real time trace after the breakpoint occurs.

BDS252<enter>3FF<enter>YA<ESC>

```
Enter sequential conditional number: [1] {1|2|3|4|5|6}
          <Enter> to next field; <Tab to next breakpoint; <Esc> to main menu.
                    Sequential Breakpoint.  Status: <active>
Sequential condition: <2>                    Breakpoint assignment to BP's
1. A or B or C or D                          A is assigned to BP <5>
2. A arms B, reset by C                      B is assigned to BP <2>
3. A arms B arms C, reset by D               C is assigned to BP <none>
4. A arms (B or C), reset by D               D is assigned to BP <3>
5. (A or B) arms C, reset by D
6. A to B time greater than: <0:00.000.000>
                                  Pass count before trap:<ff>
                              Continue trace after breakpoint:<yes>
```

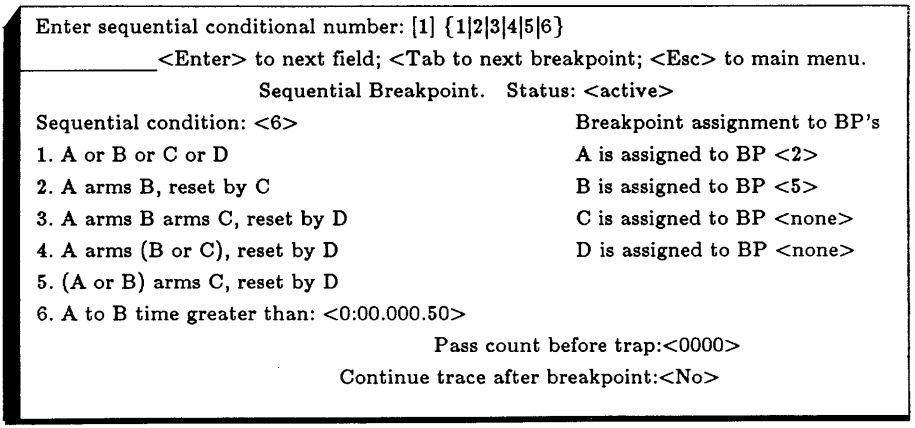
Set a breakpoint which occurs if the time between sticky breakpoints 2 and 5 exceeds 50 microseconds.

BDS625<enter><enter>50<enter>A<ESC>

```
Enter sequential conditional number: [1] {1|2|3|4|5|6}
_____<Enter> to next field; <Tab to next breakpoint; <Esc> to main menu.
                    Sequential Breakpoint.  Status: <active>
Sequential condition: <6>                   Breakpoint assignment to BP's
1. A or B or C or D                         A is assigned to BP <2>
2. A arms B, reset by C                     B is assigned to BP <5>
3. A arms B arms C, reset by D              C is assigned to BP <none>
4. A arms (B or C), reset by D              D is assigned to BP <none>
5. (A or B) arms C, reset by D
6. A to B time greater than: <0:00.000.50>
                                  Pass count before trap:<0000>
                              Continue trace after breakpoint:<No>
```

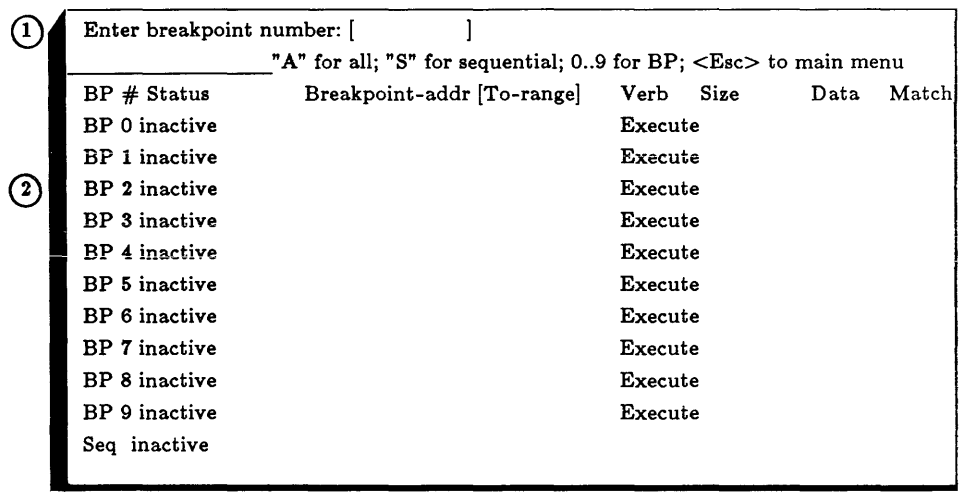
Set a breakpoint on overwriting the variable FAHR which is at address 722 when its data value is 90 decimal. Assume that BUS SIZE is Long and that FAHR is addressed as a Long variable. Since FAHR is spread across two memory locations, PROBE can only trap on a write to the upper or lower word. To trap a write to both words, use a sequential breakpoint A>B. A is a write to the most significant word and B is a write to the least significant word. Here are the screens for A, B, and S.

BP # Status	Breakpoint-addr [To-range]	Verb	Size	Data	Match
BP 0 active	FAHR	W	W	0	Equal
BP 1 active	FAHR+2	W	W	90T	Equal


Enter sequential conditional number: [1] {1|2|3|4|5|6}
<Enter> to next field; <Tab to next breakpoint; <Esc> to main menu.
Sequential Breakpoint. Status: <active>
Sequential condition: <6>
1. A or B or C or D
2. A arms B, reset by C
3. A arms B arms C, reset by D
4. A arms (B or C), reset by D
5. (A or B) arms C, reset by D
6. A to B time greater than: <0:00.000.50>
Breakpoint assignment to BP's
A is assigned to BP <2>
B is assigned to BP <5>
C is assigned to BP <none>
D is assigned to BP <none>
Pass count before trap:<0000>
Continue trace after breakpoint:<No>

ACTIVATE AND INACTIVATE BREAKPOINTS

Two other subcommand choices for the Breakpoint command are Activate and Inactivate. When either subcommand is selected, the following screen appears:



① The first DIALOG BOX prompts you for the sticky breakpoint number to activate or inactivate:

Enter breakpoint number: [ ]

Select a breakpoint number or type A for all breakpoints to be activated or inactivated.

② An abbreviated breakpoint summary is shown to make it easy to see the current status of sticky breakpoints.

EXAMPLES ACTIVATE/INACTIVATE BREAKPOINTS

Inactivate all sticky breakpoints.

BUA<Esc>

Activate sticky breakpoints 1,2, and 5.

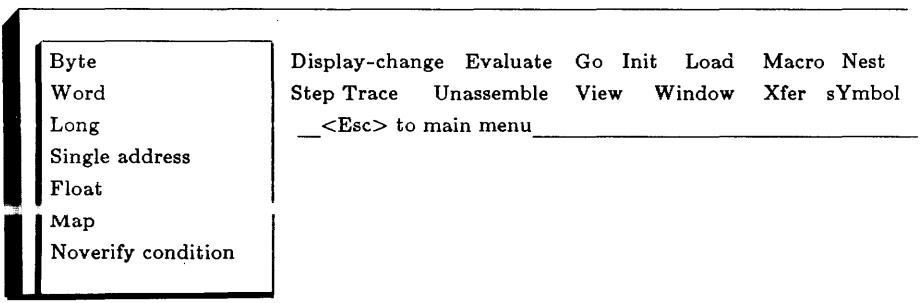
BA125<Esc>

DISPLAY AND CHANGE MEMORY

This command lets you change and display memory, and variables. It lets you display and change MAP RAM attributes. It also lets you determine if PROBE should do a read after write check whenever it changes memory. The display command is invoked from the MENU BAR by typing:

D for **D**isplay

The subcommands for Display appear in a MENU BOX like this:



The subcommands for the Display command are:

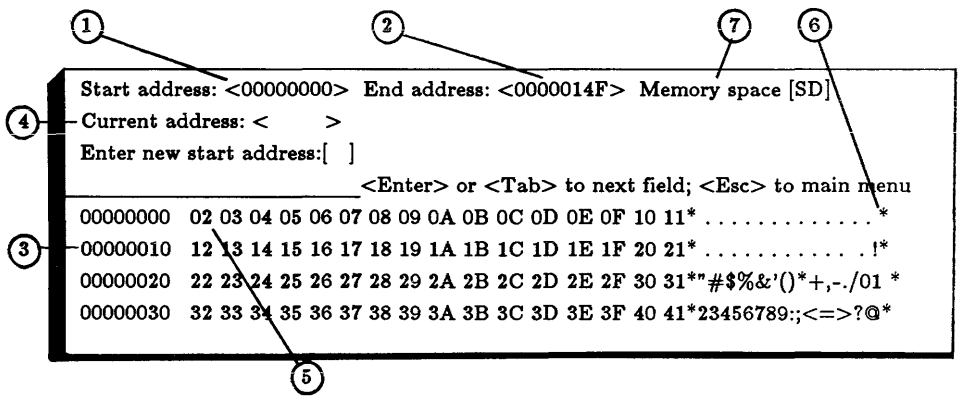
Subcommand	Operation
Byte	Display/change bytes in memory
Word	Display/change words in memory
Long	Display/change long words in memory
Single address	Display/change peripheral devices or single address.
Float	Display/change floating point data in memory
Map	Display/change memory mapping for Bus Size, Guarded access and MAP RAM
NoVerify condition	Display/change read after write condition

BYTES, WORDS, LONG

Memory can be displayed and changed in memory by selecting the first character of the following data types:

Byte
Word
Long word

After selecting the data type, the DISPLAY WINDOW looks like this:



(1) The first DIALOG BOX prompts you for the start address of the memory to be displayed:

Enter new start address: [ ]

You may enter any type of address expression. If you type <enter>, you will get the default start address.

(2) Next you are prompted for the end address

Enter new end address: [ ]

You may enter any type of address expression. The end address may also be of the form:

+ number

In this case the end address becomes start address+number. If you simply type <enter> without entering a new end address, you get the default end address shown on the screen.

③ This is the address field of the displayed memory in the DISPLAY WINDOW.

④ This is the current address of the memory location which is highlighted in the DISPLAY WINDOW. If the current address matches or is close to a symbol in the symbol table, then the symbol is also shown in this field along with the address.

⑤ A location in memory is highlighted in the DISPLAY WINDOW. You are prompted to enter a new value into memory at this location. The value can be a symbolic expression, a number, or a string of "characters". If the value is a string, each character is written to the next unit (i.e. byte, word, or long). If the value is an expression, the standard PROBE editing keys can be used to make changes to the expression in the DIALOG BOX.

Enter new value: [ ]

After <enter> is typed, the value is deposited into memory and the highlight moves to the next address. The value deposited in memory is also displayed in the cell in the DISPLAY WINDOW unless you have the Noverify condition set so that a read after write does not occur. You can move the highlight field in the DISPLAY WINDOW with the cursor and PgUp/PgDn keys. This lets your scroll through memory and make changes visually on the screen.

⑥ A duplicate of the data is shown as ASCII in this area for the Display Byte command. Not for the Word or Long.

⑦ Typing <TAB> while in the Display command will bring up the following prompt:

Memory space: < > {0|UD|UP|UR|4|SD|SD|CPU}

The memory space into which the data are accessed can be selected with this prompt. The default memory space is SD (supervisor data). Typing additional <Tab> keys will recall the initial prompt sequences for this command. Type ESC to terminate this command.

Note that PROBE will display multiple locations in memory when the cursor and paging keys are used. This may be a problem if the displayed area is actually a device. In these cases, use the Display Single address command so you will not chance reading a peripheral device which you do not want to read.

EXAMPLES FOR DISPLAY BYTES, WORDS, LONG

This example demonstrates using the Display command to show a screenful of words starting at location BUFFER in the current module. The key sequence is:

DWBUFFER<enter><enter>

To look at the next screenful of words at this point type:

<PgDn>

Change the value of 3 bytes of memory starting at TEMP to 0. Assume symbol TEMP is in the module MAIN which is not the current default module.

DBMAIN\TEMP<enter><enter>0<enter>0<enter>0<enter><ESC>

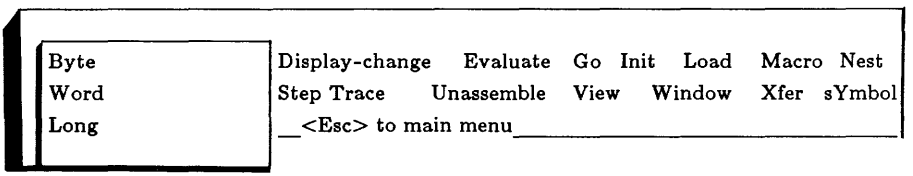
The following examples show only the memory location and its contents for data of different lengths. Assume memory contents:

ADDRESS	CONTENTS
00000000	01234567
00000001	23
000001234	AA
01234567	BB

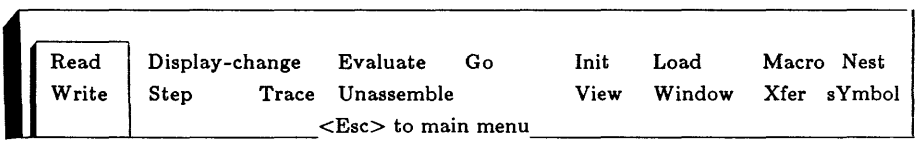
COMMAND	RESULT ADDRESS	DATA
DB0<enter>+1<enter>	00000000	00
DB[0].B<enter>+1<enter>	00000001	23
DB[0].W<enter>+1<enter>	00001234	AA
DB[0].L<enter>+1<enter>	01234567	BB

SINGLE ADDRESS FOR PERIPHERAL DEVICES

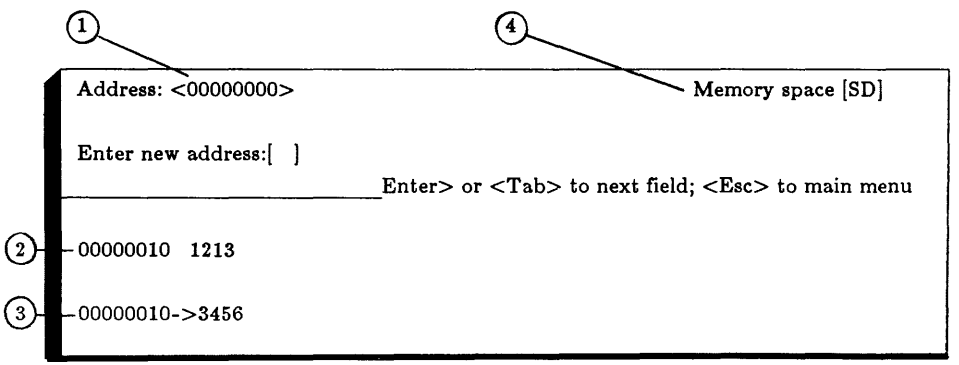
In some cases, a peripheral device is addressed as a memory location and it does not make sense to treat it as a block of memory. In fact, it may be harmful to read other peripheral devices while scrolling the PROBE display through memory. For these cases use the Single address subcommand. When the Display Single address command is selected the following screen appears:



A peripheral device may be treated as a 1, 2, or 4 byte type (i.e. Byte, Word, Long). Once the type is selected, this MENU BOX appears.



After selecting the peripheral for read or write, the following screen appears.



① Next you are prompted for the address of the peripheral:

Enter new address: [ ]

② If you are simply reading the peripheral, the address of the peripheral and data are shown in the DISPLAY WINDOW. The above prompt remains on the screen and you can select a new address or repeatedly read the same single address by typing <enter>.

③ If you are writing to the peripheral instead of reading, then the DIALOG BOX prompts you for a new value:

Enter new value:[ ]

The value is written to the single address when <enter> is typed. This prompt remains on the screen and you can write new values to the single address. You can repeatedly write the same value to the single address by typing <enter>. A read after write never occurs for Display Single address.

④ By typing <TAB> you can reach the memory space field to change the memory space of the single address. Type additional <TAB>s to recall the previous MENU BOXES and DIALOG BOXES.

DISPLAY SINGLE ADDRESS EXAMPLES

Display the word at the peripheral located at SERIAL$IO.

DSWRSERIAL$IO<enter>

Now read at this same location 3 more times.

<enter><enter><enter>

Write a 01 to the byte of the peripheral at SERIAL$IO

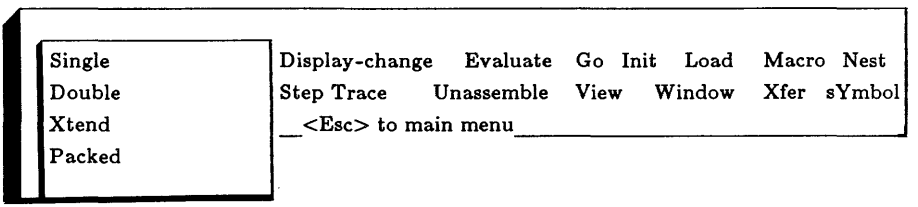
DSBWSERIAL$IO<enter>01<enter>

Write this byte to this peripheral again 3 more times:

<enter><enter><enter>

FLOATING POINT

If you have a 68881 floating point co-processor in your target, you can display and change memory of a floating point format. When you select the Display Float command, the following menu appears:



These menu options let you select the floating point data type and correspond directly to their definitions in the MC68881 Floating Point Coprocessor User's Manual supplied by Motorola, Inc. Next, you are prompted for the address of the data:

Enter new start address:[ ]

The data at this address is displayed in the floating point data type you selected. You are then prompted to enter a new value:

Enter new value:[ ]

If you do not have a 68881 in your target, the values displayed by this command are in hex. If you have the 68881, the conversions are done and the data is displayed in decimal in the formats described by the 68881 User's Manual. Values can be entered in the format in the 68881 Users manual. In addition, the values of NAN (not a number) and INFINITY are also accepted. Once the value, <enter> will select the address for the next value. In this command, each line on the screen shows only one floating point value, since some values will fill nearly an entire line. The cursor keys and highlighting work the same in this command as in all others.

The <TAB> can be used to invoke the prompt which lets you change the Memory Space field for address.

VERIFY CONDITION

Whenever PROBE writes to memory, it does a read to verify that the write actually took place. For some application, this is not desirable. To inhibit the read after write, select the Display Noverify command. The following prompt then appears:

Verify that memory has changed after each memory write: [ ] {Yes|No}

The verify condition is checked for the following commands:

Display-change- Byte, Word, Long
Assemble - Insert, Replace
Xfer - Move, Set

Bus timeout in "Display" command

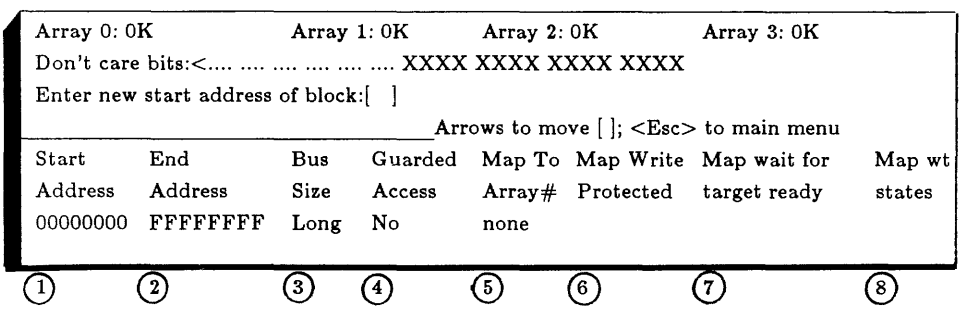
If the target system does not respond to a Display command within 250-300 milliseconds, an error message is printed. Note that this timeout feature is always enabled and is not affected by the setting of the "Hardware Watchdog timeout" command. That command only affects bus timeouts with the Go command which are monitored with PROBE's watchdog timer.

MEMORY MAPPING

The memory space as viewed from the 68020 cpu can have attributes which are provided by the PROBE. These attributes are assigned to the memory space in blocks. A block of memory has a start address and an end address. Blocks are not restricted to one size and each block can be a different size. The minimum size for a block is 64k bytes and the maximum size is the entire memory space of the 68020. The size of a block must be a multiple of 64k. Since blocks are defined by start and end addresses, they do not overlap. Most attributes for blocks can be set with or without MAP RAM boards.

DEFINING A BLOCK

The first time you select the Display Map command, the DISPLAY WINDOW shows each block and its attributes. The screen below is the initial default setting and is the screen you will see unless you have changed the attributes or initialized the blocks with the Init Load command.



One of the fields in the DISPLAY WINDOW is highlighted and the highlight may be moved with the cursor keys. Here is a description of the fields in the previous screen.

(1) When this field is highlighted, the DIALOG BOX prompts you for the starting address of the block. The starting address must be at the start of a 64k boundary (i.e. the lower two bytes must be 0).

Enter new start address of block:[ ]

② When this field is highlighted, you are prompted for the ending address of the block. The end address must be at the end of a 64k block (i.e. the lower two bytes must be FFFF)

Enter new end address of block: [ ]

New blocks are created by splitting old blocks or by consuming multiple old blocks into 1 new block. This is all done by setting new starting and ending addresses for the block which is highlighted.

Specifically, if, for the highlighted block, you set a new starting or ending address which overlaps an adjacent block,then the boundary between the adjacent blocks is moved. If the boundary moves inside an adjacent block, then the adjacent block becomes smaller and the highlighted block becomes larger. If the new boundary moves beyond the boundary of adjacent blocks, then the adjacent blocks are consumed into the highlighted block.

The other case is when the starting or ending address is within the highlighted block. In this case the highlighted block is split into two blocks. The two new blocks inherit the attributes of the highlighted block. You can change these attributes, of course, by moving the highlight and changing the attribute fields.

③ When this field is highlighted, you are prompted for BUS SIZE. BUS SIZE is the physical width of the data bus in the target for this block. The choices for this field are B, W, and L which mean 8, 16, or 32 bits.

Enter bus size for this block: [Long] {Byte| Word| Long}

Note that the target system may have several different BUS SIZES in its memory space. Blocks mapped to the target system must be set to the correct BUS SIZE. If this is not done, then PROBE Read/Write/Fetch/Any breakpoints will not work properly. PROBE breakpoints look at BUS SIZE for these breakpoints in order to find which physical data bus lines on the 68020 will receive the data. If a block is mapped to a MAP RAM array, this field is automatically set to Long by PROBE. Blocks

mapped to a PROBE array can only be changed to another BUS SIZE if Map Wait for Target Ready is select to yes.

(4) When this field is highlighted, you can guard all accesses (i.e. read, write, fetch.) to this block. During emulation, any access to a guarded block will cause a breakpoint. You do not need a MAP RAM board to MAP a block as guarded.

Guard all accesses to this block: [N] {Yes|No}

(5) You can only get to this field if a MAP RAM board is installed in your PROBE. When this field is highlighted, you are prompted to map this block to the PROBE MAP RAM. If you have a 512k byte MAP RAM board in your PROBE POD, then the top of the screen shows that you have four arrays of memory with 128k bytes of memory in each array.

Map this block to PROBE MAP RAM array:[none]{None|0|1|2|3}

You can put two 64k blocks or one 128k block into each array. If you put two 64k blocks into the array, the blocks must both be in the same 16 megabytes of memory space.

(6) You cannot get to this field unless the block is mapped to a MAP RAM array. When this field is highlighted you are prompted to determine if the block should be write protected Write protecting a block will not let the memory change if the program overwrites this memory. Write protecting memory is a way of simulating PROM with the MAP RAM.

Write protect this block: [No] {Yes|No}

(7) This is an important attribute for some systems. You can only get to this field if you have this block mapped to a MAP RAM array. The prompt for this attribute is:

Map wait for target system ready:[yes] {Yes|No}

If you set Wait for Target Ready ready for this block to Yes, then accesses to the MAP RAM for this block will wait for the DACK0 and DACK1 signals from the target as well as the MAP RAM before completing a memory cycle. The MAP RAM, will in this case, also responds in the same manner as the target memory

would respond (byte, word, long). If the MAP RAM is faster than the target memory, then the MAP RAM does not add additional wait states. Remember to set the BUS SIZE attribute for this block to match the bus size of the target.

Target systems which synchronize logic to the Address Strobe of the 68020, and become unsyncronized if the 68020 proceeds without waiting for the DACK0 and DACK1 signals from the target, should set Wait for Target Ready to Yes. The advantage is that the target system logic will not become unsynchronized with the 68020 when PROBE does accesses to the PROBE MAP RAM. The disadvantage is that waiting for the ready for both the target and MAP RAM will introduce an additional wait states to the access if the MAP RAM access time is longer than the target. Another disadvantage is that if the target system does not return the DSACK's then the target will hang. To get out of this hang state you must type <Esc> or set the PROBE watchdog timer. Target systems which do not synchronize logic with the 68020 do not need to wait for target system ready.

⑧ This attribute lets you choose the number of wait states for the MAP RAM. The PROBE software sets the lower limit for this attribute by checking the target clock frequency and MAP RAM board speed. It then automatically sets the number of wait states to the minimum value. You may increase the number of wait states to a maximum of 4. Once the number of wait states is chosen for the MAP RAM, it must be the same value for all blocks mapped to the MAP RAM. The table below shows the minimum number of wait states which PROBE chooses as a function as a function of clock frequency and MAP RAM access time.

MIN WAIT STATES VS CLOCK AND ACCESS TIME

	CPU CLOCK SPEED				
RAM SPEED	8 MHZ	12 MHZ	16 MHZ	20 MHZ	25 MHZ
70 ns	0	0	0	1	1
100ns	0	0	1	1	2
150ns	0	1	2	2	3

EXAMPLES OF DISPLAY MAP COMMAND

Assume the MAP is in its default state which looks like this:

```
Array 0: 128K          Array 1: 128K    Array 2: 128K          Array 3: 128K
Don't care bits:<.... .... .... .... .... .... XXXX XXXX XXXX XXXX
Enter new start address of block:[  ]
                                        Arrows to move [ ]; <Esc> to main menu
Start     End       Bus   Guarded  Map To  Map Write  Map wait for   Map wt
Address   Address   Size  Access   Array#  Protected  target ready   states
00000000  FFFFFFFF  Long  No       none
```

Create a block from 0 to 1FFFF which is mapped as a 32 bit bus, not guarded, mapped to PROBE MAP RAM, and not write protected. Create a block from 20000 to DFFFFFFF which is guarded. Create a block from E0000000 to FFFFFFFF which is has a bus size of 32 bits, and not guarded.

DM<enter>1FFFF<enter><enter><enter>1<enter>Y<enter><enter>
<enter>DFFFFFFF<enter><enter>Y<enter><enter><enter><enter>
<enter><enter>Y<enter>Y<enter><esc>

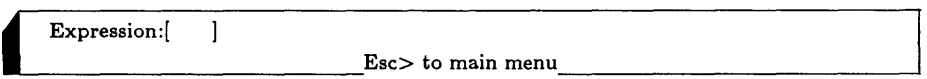
```
Array 0: 128K          Array 1: 128K     Array 2: 128K          Array 3: 128K
Don't care bits:<.... .... .... .... .... .... XXXX XXXX XXXX XXXX
Enter new start address of block:[  ]
                                        Arrows to move [ ]; <Esc> to main menu
Start     End       Bus    Guarded  Map To  Map Write  Map wait for  Map wt
Address   Address   Size   Access   Array#  Protected  target ready  states
00000000  0001FFFF  Long   No       1       No         Y             1
00020000  DFFFFFFF  Long   Yes      none
E0000000  FFFFFFFF  Long   No       none
```

EVALUATING EXPRESSIONS

This command lets you evaluate expressions and display the results in several different bases. The command is invoked by typing:

E for Evaluate

The following DIALOG BOX appears:



You can now type in any expression using the operators described at the start of this chapter. The results are displayed in the DISPLAY WINDOW in the following format:

HEX DECIMAL INTEGER ASCII BINARY

EXAMPLES OF THE EVALUATE COMMAND

Assume the symbol main is 20000000
Memory at 20000000 is AABBCCDD
Memory at 20000005 is 11223344
Memory at 11223344 is EEFF7788
Memory at 11223354 is 99005566

Evaluate expression: main
20000000H 536870912T +536870912T '...' 00100000,00000000,00000000,00000000

Evaluate expression: [main]
AABBCCDDH 286443439T -1430532899T '...' 10101010,10111011,11001100,11011101

Evaluate expression: [[main+5].l+10].w
00009900H 39168T +39168T ' ' 00000000,00000000,10011001,00000000Y

Evaluate expression: A0 (assume register A0 contains FFFFFFFF)
FFFFFFFH 4294967295T -1T ' ' 11111111,11111111,11111111,11111111Y

Evaluate expression: ((50*10)-1)/499
00000001H 1T +1T '.' 00000000,00000000,00000000,00000001Y

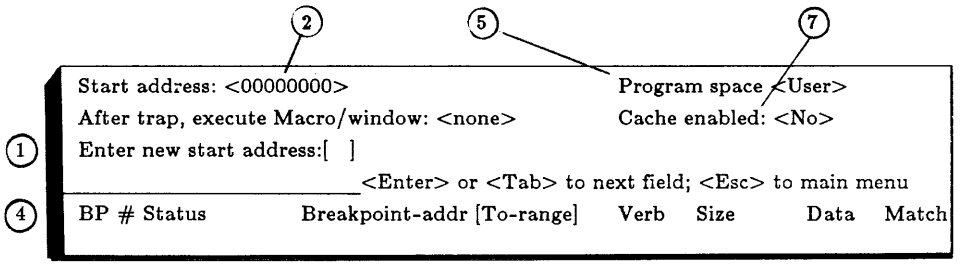
Evaluate expression: 'a'
00000061H 97T +97T 'a' 00000000,00000000,00000000,01100001Y

GO COMMAND

Program execution is started and non sticky breakpoints are set with the Go command. The breakpoints activated with the **B**reakpoint command are also set when the Go command is executed. It is invoked from the MENU BAR by typing

G for **G**o

The DISPLAY WINDOW looks like this:



(1) The first DIALOG BOX prompts you for for a new start address:

Enter new start address: [ ]

(2) If you type <enter> without entering a start address, the current PC (program counter) will be used as a default and is shown in the Start address < > field.

(3) Next you have the option of setting breakpoints or starting program execution. You are given the following prompt:

Start program execution now; [Yes] {Yes|No}

Simply typing <enter> will select Yes and program execution will start. If you select No, then the Breakpoint screen (see Breakpoint command) will be displayed. This lets you set breakpoints in the same way as with the Breakpoint command. Breakpoints set on this screen in the Go command are non-sticky breakpoints. They are only active when program execution starts with this Go command. They are not included in future Go commands unless you enter them again.

After entering the breakpoint on the breakpoint screen, type <TAB> to recall the previous prompt. This lets you set another non-sticky breakpoint or start program execution.

④ An abbreviated summary of the active sticky breakpoints which will be used with this Go command is shown on this screen for easy reference.

⑤ Typing additional <TAB> characters while in the Go command will bring up the following prompt:

Enter initial memory space: [User] {User|Supervisor}

This lets you start program execution in the User/Supervisor mode. If this prompt is not invoked, the default as shown on the screen is assumed.

⑥ Typing <TAB> again will bring up the following prompt:

Enier macro/window <AltKeyname>:[<none>]

This lets you invoke a macro , or a Watch Window after a breakpoint has occurred and program execution has stopped. If this prompt is not invoked, the default as shown on the screen is assumed. Note that Watch Windows and Macros can be invoked from the individual macros as well. The Watch Window invoked by the Go command preempts all others. Macros are executed for each breakpoint first followed by the macro invoked by the Go command.

⑦ Typing <TAB> again will bring up the following prompt:

Enable cache while executing [No] {Yes|No}

The program can execute with the 68020 PROBE cache enabled or disabled. This affects the information which is entered into the PROBE real time trace memory. See the Trace command for a more detailed discussion of the Trace data. If you choose No for this prompt, then the 68020 cache is not enabled during program execution. If you choose Yes for this prompt, then the 68020

cache is controlled by the target system hardware and least significant bit of the CACR register which is under target system program control.

Typing Esc at any point will terminate the Go command. Sequential breakpoints, breakpoint pass count and trace after trigger can only be set in the Breakpoint command and not in the Go command.

EXAMPLES OF THE GO COMMAND

Start program execution with the default start address. Only active breakpoints defined by the BP command are set:

G<enter><enter>

Start program execution with the default start address and set a breakpoint at MAIN:

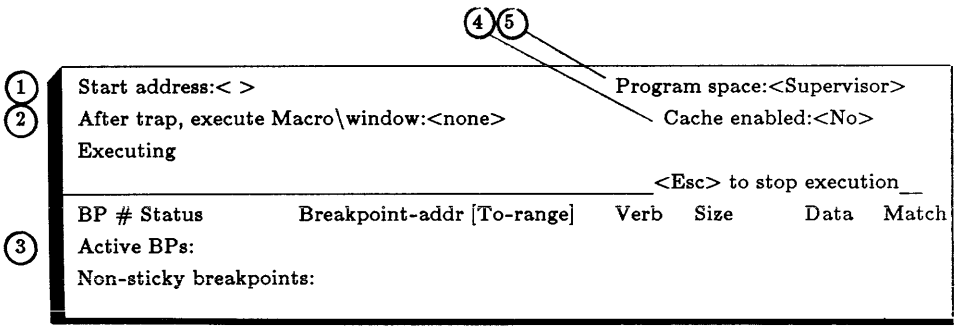
G<enter>NMAIN <Tab><enter>

Start program execution at the current program counter, set a breakpoint on writing to location BUFFERPTR, change the memory space to User, and execute the macro AltB when program execution stops:

G<enter>NBUFFERPTR<enter><enter><enter><enter>W
<Tab><Tab><Tab>U<Tab>AltB<enter><enter>

STATUS SCREEN AND WATCH WINDOWS DURING EMULATION

While the target program is executing, the Status screen below is displayed:



1. The start address for the current Go command is displayed.
2. This is the Macro or window which will execute when program execution is stopped either with a breakpoint or with the <Esc> key.
3. The current breakpoints are displayed in an abbreviated form. The display includes active sticky breakpoints as well as non-sticky breakpoints
4. This is the state of the cache control when program execution started.
5. Program execution was started in this memory space.

In addition, a Watch Window may be popped up by typing the *AltKey* for the window during execution. The window can display information in the target system. To do this, the PROBE periodically (about once every second) terminates target system execution and the information needed for the Watch Window is extracted from the target. Hitting a breakpoint or typing ESC terminates emulation along with the Go command and control is transferred back to the MENU BAR.

NOTES ON EMULATION

If the cpu executes a STOP instruction during emulation, it will behave as a normal STOP instruction. Since no Address strobes are produced while the cpu is stopped, the PROBE Watchdog timer will cause a breakpoint if it is enabled. You can also regain control of the system by typing <Esc>.

Bus errors which occur in the target are processed by the target as they normally would without PROBE intervention. If you want to trap a Bus error, then you should set a breakpoint on reading the Bus error vector or executing the first instruction of the error handling procedure.

If the 68020 stops executing because of a double Bus error or Halt line, and you type the <Esc> key to regain control, the following error message will occur:

No address strobes to target processor. Use Hw Reset command.

Type any key to clear this error message as you would any other message. Several other error messages may follow which must also be cleared with any key.

Could not stop execution of target processor. Use Hw Reset command
No address strobes to target processor. Use Hw Reset command.

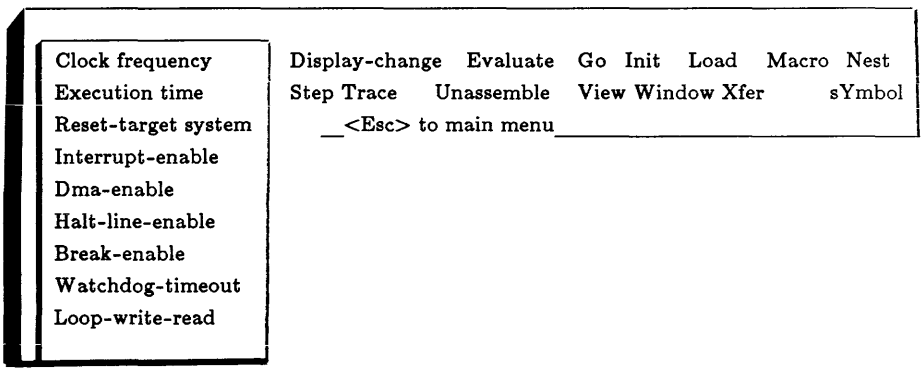
Since the 68020 has stopped, PROBE cannot communicate with it. You must use the Hardware Reset command to put the 68020 in a state which will allow the PROBE to talk to it.

HARDWARE CONTROL

This command provides control over certain hardware related aspects of the 68020. Specifically, it lets you reset the 68020, control the interrupt and DMA operation of the cpu, and read the clock frequency. This command is invoked from the Menu Bar by typing:

H for Hardware

The following Menu Box:



The subcommands for the Hardware command are:

Subcommand	Operation
Clock-frequency	Measure the target clock frequency
Execution time	Time between Go and breakpoint
Reset-target	Apply a reset to the 68020 cpu
Interrupt-enable	Enable interrupts during Go or Single step
DMA-enable	Enable DMA when not emulating.
Halt-line-enable	Enable Halt line when emulating
Break-enable	Do not stop emulation when breakpoint detected
Watchdog	Cause a breakpoint when there is no ready returned from the target for 10 ms.
Loop-write-read	Repeat read\write operation to target

READING TARGET CLOCK FREQUENCY

This subcommand, selected by typing C, simply returns the frequency of the clock in the target system.

EXECUTION TIME

When you use the Go command to start program execution, PROBE starts a timer. When a breakpoint or <ESC> occurs, the timer is stopped. This command lets you read timer. This is useful for measuring the execution time for sections of code. The timer is displayed as:

minutes:seconds.milliseconds,microseconds

RESET TARGET SYSTEM

This subcommand lets you reset the 68020 in the target. It will also reset any other hardware in the target connected to the reset line.

Reset the target system and processor registers:[No] {Yes|No}

"Unshadow" the rom in the User's 68020 system.

When the 68020 processor is reset, it reads the 4-byte pointers at memory locations 0 and 4 to set the initial stack pointer and PC. Since many systems have ram starting in location 0 this presents a problem because the ram will be uninitialized when the system powers up. Many systems solve this problem by "shadowing" their prom at location 0 after reset so that the correct PC and stack pointer are read. Then some hardware operations usually follow that disable prom addressing at 0 and enable the ram.

If the ram in the target system does not seem to be working, verify the prom has been "unshadowed" if necessary.

ENABLE INTERRUPTS IN TARGET SYSTEM

This subcommand lets you start target system execution with interrupts enabled or disabled whenever you are using the Go or Step commands. It provides the following prompt:

Enable interrupts while emulating: [Yes] {Yes|No}

PROBE disables interrupts by masking them from the 68020 cpu with hardware logic. It is useful to disable interrupts in the target system when spurious interrupts are happening and the system has not yet initialized interrupt vectors.

DMA

This subcommand lets you ignore or process DMA requests coming from the target system while the 68020 is not executing code in the target (i.e. not emulating). This lets other hardware in the target continue to run even though emulation has stopped.

Allow DMA requests to be serviced: [No] {Yes|No}

It is useful to disable DMA requests from the target in cases where the requests will not release the processor.

ENABLE HALT SIGNAL

This subcommand lets you enable the HALT signal to the 68020 from the target while the 68020 is executing code in the target. It provides the following prompt:

Enable HALT signal to target processor while emulating: [Yes] {Yes|No}

BREAKPOINT ENABLE

This subcommand lets you inhibit the breakpoint logic from the PROBE so that target program execution is not stopped when a breakpoint is detected. When a breakpoint is detected, a pulse is output on the BREAKPOINT DETECT signal in the POD. This signal is available through the Logic PROBES. See Appendix H for details. This signal can be used to trigger a logic analyzer or scope. By inhibiting the breakpoint from stopping program execution, the PROBE can use it's sophisticated breakpoint detect logic to provide triggers each time the breakpoint is detected. Note, that the trigger is not provided for software breakpoints (i.e. Execute). This subcommand provides the following prompt:

Break emulation when hardware breakpoint detected: [Yes] {Yes|No}

WATCHDOG TIMEOUT

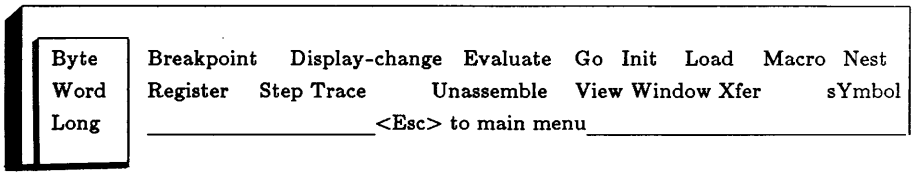
This subcommand will let you detect when the ready has not been returned from the target system for 10 milliseconds. If enabled, the watchdog timer causes breakpoint when the target system does not return ready. The following prompt is provided:

Enable watchdog address strobe timeout (no strobe in > 10 ms:[Y]{Y|N}

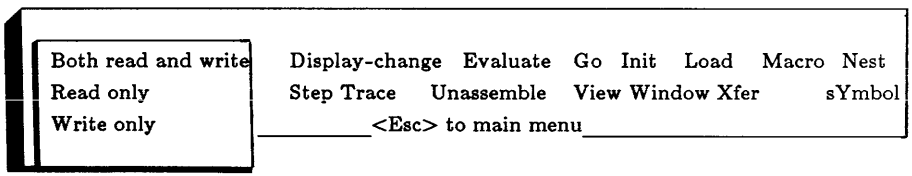
If you want to terminate emulation when the 68020 executes a STOP or the target does not return ready, then set the watchdog timer.

A LOOPING TEST FOR THE TARGET

This subcommand will let you repeatedly execute write/read operations in the target. This is very useful when a "sync" signal is needed to trigger a scope or logic analyzer. When this command is invoked, another MENU BOX lets you select the size of the operation.



The next MENU BOX lets you select from three different types of access:



Next you are prompted for the address and data for the operation:

Enter address for access loop:[ ]
Enter data to write to address: [ ]

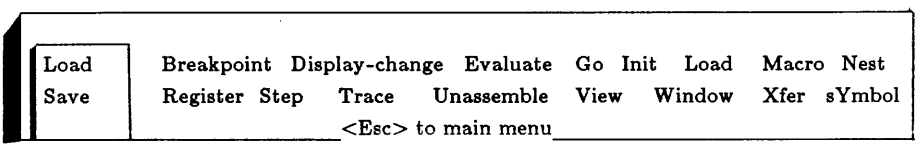
PROBE performs the operation at this address. Typing any key terminates the process.

INITIALIZATION

Parameters which you set in PROBE can be saved to a disk file and recalled in future debugging sessions with the Init command. This command lets you completely set up the PROBE for a debugging session. The command is invoked from the Menu Bar by typing:

I for Init

The subcommands for the Init command now appears in the Menu Box:

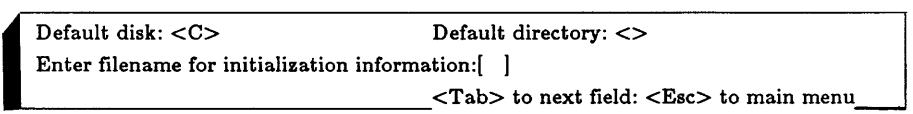


The Init command has the following Subcommands:

Command	Sub command	Operation
Init	Load	Load previously saved initialize conditions
	Save	Save initialize conditions

SAVING INITIALIZATION PARAMETERS

When the Save subcommand is selected, the following screen pops up:



The first DIALOG BOX prompts you for the filename where the initialization conditions are to be stored:

Enter filename for initialization information:[ ]

The default drive and directory shown on this screen are used if you do not specify them. If you cannot remember the name of the file you want to use then type * to display all files in the directory.

The following is saved by the Init Save command:

1) The entire state of the memory mapping as defined by the Display Map command.
2) Module information as set by the sYmbol command including:
   a) The name of all modules
   b) If symbols will be loaded for the module as defined with the sYmbol Load-module-selection command.
   c) If source level single stepping is to include or ignore a module as defined with the sYmbol Source-step-module-selection command.
   d) Filenames which are assigned to the module names as defined by the sYmbol Assign-module-to-file command.
3) Program filename used in the most recent Load command
4) Macro filename used in the most recent Macro Load command. If no Macro Load command has been used, then the most recent filename used in the Macro Save command is saved with Init.
5) Window filename used in the most recent Window Load command. If no Window Load command has been used, then the filename used in the Window Save command is used instead.

The information is stored as ASCII text in the initialization file. You may edit this text off line with a text editor. The definition of the contents of this file is shown in Appendix D.

LOADING INITIALIZATION PARAMETERS

When the Load subcommand is selected, the following screen pops up:

```
Default disk: <C>                    Default directory: <>
Enter filename for initialization information:[  ]
                                  <Tab> to next field: <Esc> to main menu
```

First you are prompted for the filename where the initialization conditions are stored:

Enter filename for initialization information:[ ]

The default drive and directory shown on this screen are used if you do not specify them. If you cannot remember the name of the file you want to use then type * and all files in the directory will be displayed.

Initialization information stored in a previously saved file can be loaded to completely set up PROBE for a debugging session. When the file is loaded, the following occurs:

1) The memory map is defined.
2) Modulename information is stored such as:
   a) Modulenames are put into PROBE'S modulename table.
   b) If symbols will be loaded for the modulename when the Load command is invoked.
   c) If source level single stepping is to include or ignore a module when the Step Source command is invoked.
   d) Filenames are assigned to the module names for use during Step Source commands.
3) A program is loaded.
4) Macros from a Macrofile are loaded. If PROBE has currently defined macronames which are the same as macronames found in the loaded Macrofile, then those macronames are left unchanged in PROBE.
5) Window definitions from a window file are loaded. If PROBE has currently defined windownames which are the same as windownames found in the loaded file, then those windownames are left unchanged in PROBE.

EXAMPLES OF THE INITIALIZE COMMAND

Display the files in directory \MAIN\DEMOFILES which have the extension .INI and save the current setup in an initialization file in this directory called DEMO.INI

IS\MAIN\DEMOFILES*.INI<enter>
<Home><Home><Home><Home><Home><Del><Ins>DEMO<enter>

To load the initialize file named demo.ini from drive C, directory \DEMOS type this key sequence:

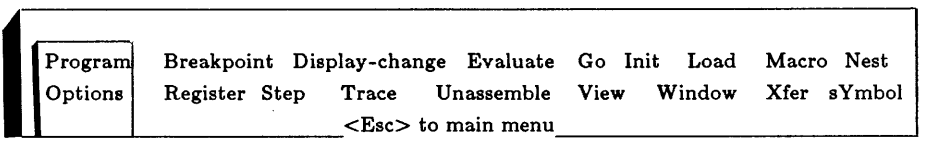
ILC:\DEMOS\DEMO.INI<enter>

LOADING PROGRAMS

Target system programs and symbol table are loaded with the Load command. The load command is invoked from the MENU BAR with:

L for Load

The subcommands for the Load command appear in the Menu Box:

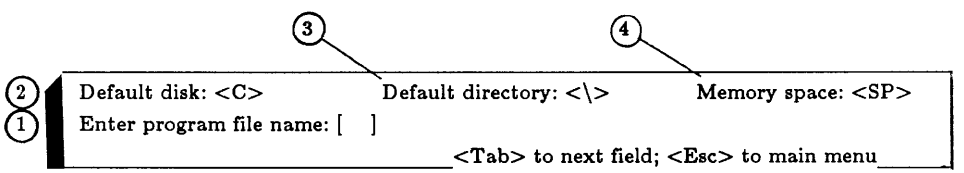


The Load command has the following Subcommands:

Command	Sub command	Operation
Load	Program	Load program
	Options	Set program load options

LOAD PROGRAM

The following screen appears:



(1) The first DIALOG BOX prompts you for the name of the file to load:

Enter program file name: [ ]

If you cannot remember the name of the program, then use the wildcard capability of PROBE to display files.

The program is loaded into the target system memory and the symbol table for the program is loaded into the PROBE symbol table. If memory blocks are mapped to the MAP RAM board, then they are used when the program is loaded. If the file contains only code and no symbols, the symbol table is not loaded.

② The default disk is shown in this field. To set a new default, type <TAB> and the following prompt appears:

Enter new drive letter: [ ]

③ The default directory is shown in this field. If this is not correct, they type <TAB> and the following prompt appears:

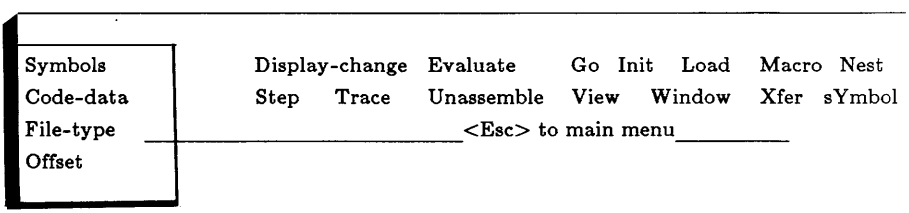
Enter new directory: [ ]

④ The default memory space to load into is shown in this field. You can change it by typing type <TAB> to get the following prompt:

Memory space:< > {0|UD|UP|UR|4|SD|SP|CPU}

LOAD OPTIONS

Load Options let you set parameters for the code and symbols which the Load Program and Initialize Load commands will use. When invoked, the following menu box appears:



Symbols

The Symbols option lets you load code without loading Symbols and has the following prompt:

Load symbols from program file:[Yes] {Yes|No}

Code

The Code-data option lets you load symbols without loading Code or data and has the following prompt:

Load code and data from program file:[Yes]{Yes|No}

File-type

The File-type option lets you specify the type of Object Module Format to expect from the file. The OMF's are described in Appendix G. If you have not made a selection with this command, PROBE tries to determine the OMF automatically when it loads the file. If you make a selection with this command and load a file which does not agree with this selection, PROBE reports an error message. The selections are shown here and described in more detail in Appendix G:

SubSub command	Operation
Auto-determine	PROBE tries to make automatic determination of OMF
S-records	Standard Motorola S records
(extended)Tek hex	or Tektronixextended Hex records
Binary-image	A binary image with no addresses or symbols
Unix-system V	Unix system V coff records
Ieee-binary-Coff	Modified coff records produced by Microtec Research compilers

Offset

The Offset option lets you specify a fixed offset which are to be added to the addresses for symbols, code and data during the load. This is very useful for relocatable code. The prompt is:

Enter offset added to symbol and code/data addresses:[ ]

EXAMPLES OF THE LOAD COMMAND

Load the program named FTOC.HEX from drive c directory EXE into the default memory space.

LPC:\EXE\FTOC.HEX<enter>

Display all of the files in directory \DEMO with a .HEX extension then load the program DEMO.HEX

LP\DEMO*.HEX<enter>
<Home><Home><Home><Home><Home><Del><Ins>DEMO<enter>

Set the program load options to no code.

LOCN

Add 10000H to all address during load.

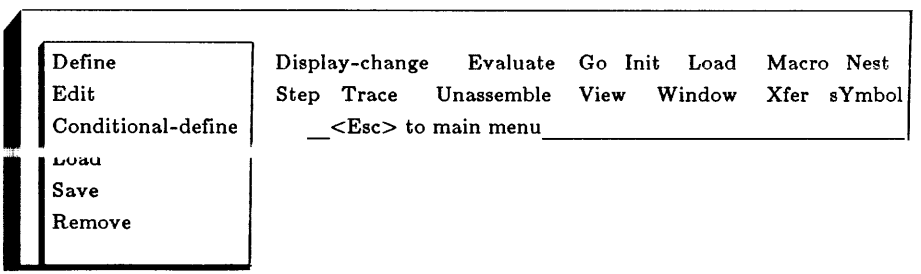
LOO10000H<enter><esc>

MACRO COMMANDS

Macros are a way to create your own commands and automate keystrokes which are repetitive. Macro commands are simply keystrokes which are assigned to one keystroke. Each macro has a name which is the keystroke which invokes it. The commands to let you define, delete, edit, and display macros are invoked from the MENU BAR by typing:

M for Macro

The subcommands for the Macro command appear in a MENU BOX and the screen looks like this.



The macro subcommands are shown here along with a short description of their operation.

Subcommand	Operation
Define	Define a new macro and assign it a name which is an *Altkey*
Edit	Change the definition of a current macro
Conditional	Define a macro which will test a specified condition before executing
Load	Load previously defined macros from file
Save	Save all currently defined macros to a disk file
Remove	Delete a currently defined macro.

DEFINING A MACRO

To start the definition of a macro type subcommand D. You are then prompted with:

Enter <AltKey> that will activate this macro: [ ]

An *AltKey* means hold down the key Alt and then type any other key. The *AltKey* becomes the macroname. The *Altkey* can be any key except *Alt0* thru *Alt9*, *Alt=* and *Alt-*. These are saved for special use within the macro The macro is defined as all key strokes until the macroname (i.e. *Altkey*) is typed again. The commands which define the macro also execute while the macro definition is in process. While the macro definition is in process, the macroname is displayed in the lower right area of the DISPLAY WINDOW to remind you that a macro is being defined.

If the macroname already exists, you are given the following prompt:

<AltKey> is a macro. Remove it: [Yes] {Yes|No}

If you answer Yes, the current macro definition is deleted and you can proceed redefining the macro. If No, you are returned to the main menu. To change a macro use the Macro Edit command.

You are also prompted to add a Macro description to the Macro. This is an ascii text string which describes your macro and can have up to 200 characters. The macro description is stored with the Macro. The Macro description can be viewed with the Macro Edit command. If you simply type <enter> in response to this prompt, no macro description is attached.

Macro description:[ ]

You cannot define a macro within a macro definition. PROBE commands are executed while the macro definition is in process.

PASSING PARAMETERS TO THE MACRO

Parameters can be included in the macro definition. A parameter is defined as:

Alt#<enter>

This means hold down the Alt key and type a number from 0 to 9. This is the reason that these *AltKeys* cannot be macronames. Up to 10 different parameters may be included in the macro definition, and each parameter may be used multiple times within the macro definition.

When the macro executes, it pauses the first time it encounters each different *Alt*# and waits for you to input the parameter. The parameter is specified as all keystrokes you type until <enter> is typed. If there are 10 different #'s, then the macro will pause 10 times.

The same *Alt*# may be used at several different places in the macro definition. When the macro executes, it will pause only the first time for this *Alt*# parameter definition. The macro will use this definition each time it encounters this *Alt*# during the current macro execution.

During definition of a macro which includes parameters (i.e.*Alt*#'s), the parameter specification is passed on to the command in process. Commands execute while the macro is being defined. Note that the <enter> which specifies the end of the parameter during the definition or execution of the macro is not passed on to the command in process. Also note that you may define a macro which has parameters without actually having to use real parameters. This is done by simply typing:

ALT#<enter><enter>

This puts the *ALT*# into the macro definition, and passes <enter> without a parameter to the command in process. This will typically produce an error message which you can simply ignore by typing any key.

A special *AltKey* is the *Alt-* key (hold down the Alt key and type the minus sign). If *Alt-* is encountered, then the macro always pauses at the *Alt-* during execution to wait for input. This lets you enter a new parameter for this special *AltKey* each and every time it is encountered. This is especially useful in conditional macros which are described later.

While parameters are being entered into the definition of a macro, or the macro is pausing during execution to receive the parameter specification, the *Alt#* for this parameter is displayed in the lower right of the screen. The definition of the parameters during macro execution is not maintained after the macro is finished executing. When this macro is invoked again, it will prompt you for the definition of each parameter. If you want the macro to always execute the same way, then do not use parameters.

NESTING MACROS

A macro may be defined which invokes other macros during execution. This is done simply by typing the appropriate *AltKey* which invokes the nested macro into the definition of the macro. During execution, the nested macro may have parameters passed to it. You can think of *Alt#*'s as global variable names which are recognized by both the outer level and nested macros.

PASSING PARAMETERS TO NESTED MACROS

Parameters may be passed to the nested macro in one of two ways. The first time an Alt # is encountered during macro execution, in either nested or outer level macros, it is specified. All further encounters of this parameter during execution of the outer level macro or the nested macro will use this specification for the Alt# (i.e. parameter).

The second way of passing a parameter to a macro lets the parameter change each time the outer macro invokes the nested macro. This is done by using the following form to redefine a parameter during macro execution:

<Alt=><Alt#A><Alt#B>

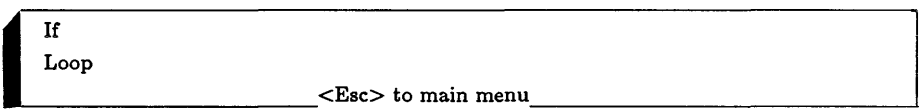
Alt= means hold down the *AltKey* and type the = key. *Alt#B* is the *Altkey* to be redefined during macro execution. *Alt#A* is the new *Altkey* definition for *Alt#B* during macro execution. When the *Alt=* is encountered by the macro during execution, *Alt#B* is replaced by *Alt#A*. This lets you define macro modules which can be reused multiple times in a macro execution while having their parameters changed on the fly by other macros which call the macro module.

DEFINING MACROS WHICH CONDITIONALLY EXECUTE

Macros can be defined to check for specified conditions before they will execute. The condition is tested each time the conditional macro is executed. This type of macro is defined by choosing the Conditional subcommand from the MENU BOX. You are then prompted with:

Enter <AltKeystroke> that will activate this macro: [ ]

The same rules for naming macros apply as were described previously. The next level of subcommand lets you choose the type of condition to test before execution when the macro executes. The following screen pops up to let you choose the conditions:

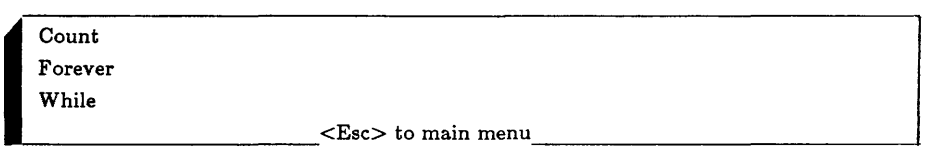


The subcommands for the Conditional subcommand are:

Subcommand	Operation
If	Execute macro If boolean expression is true
Loop	Execute macro Count, While, Forever on on boolean expression

LOOPING CONDITIONAL MACROS

If the Loop subcommand is invoked from the previous screen, then the following MENU BOX appears:



These are additional subcommands. These subcommands are defined as follows:

Subcommand	Operation
While	Continue macro execution While boolean expression is true
Count	Execute macro the number of times specified by Count
Forever	Continue macro execution until Ctrl Break key typed

Count

If you select the Count subcommand, the following prompt appears:

Loop count: [ ]

The macro will execute repetitively the number of times specified by the Loop count. Each time the macro reaches the end of its definition, it decrements the Loop count. When the Loop count reaches 0, this macro terminates execution. The Loop count can be an expression or simply a number. It can also be parameter passed from another macro. Since macros execute while they are being defined, the loop execution starts when the macro definition is ended (i.e. the *AltKey* is typed again.)

Forever

If you select the Forever subcommand, the macro will execute repetitively until Ctrl Break is typed.

While

If you select the While subcommand, the following prompt appears:

Condition : [ ]

The While condition is a boolean expression which is defined at the start of this chapter in the section called BOOLEAN EXPRESSIONS. The boolean expression is checked at the beginning of each execution of the macro. If it is true, the macro executes again. If it is false the macro execution is terminated. Since you may be defining a conditional macro at a time when the condition is not true, the macro definition ignores the condition so that you can continue the definition.

IF CONDITIONS

Another type of condition for the Conditional macro definition is the IF condition. When this subcommand is selected, you are prompted with:

Condition : [ ]

The IF condition is a boolean expression which is defined at the start of this chapter in the section called BOOLEAN EXPRESSIONS. The boolean expression is checked at the start of execution of the macro. If it is false, the macro execution is terminated. Since you may be defining a conditional macro at a time when the condition is not true, the macro definition ignores the condition so that you can continue the definition.

LOAD AND SAVE MACRO FILES

Macros can be loaded and saved from disk. The Load and Save subcommands prompt you for the filename:

Enter macro file name: [ ]

If the drive and directory are not specified, then the defaults shown on the screen are used. See the section in this chapter labeled USING WILDCARD CHARACTERS to display filenames.

DELETING A MACRO

A macro can be deleted by choosing the Remove subcommand from the MENU BOX. You are them prompted with:

Enter <AltKeystroke> that will activate this macro: [ ]

By specifying the key which would normally activate this macro in response to this prompt, it is removed.

EDITING MACROS

Once defined, macros can be edited by choosing the Edit subcommand from the MENU BOX. If you cannot remember the name of the macro you want to edit, they type *. This will display a list of all macronames and windownames currently assigned to *AltKeys*.

If the macro already exists, then the current definition of the macro is shown in the DISPLAY WINDOW as a sequence of keystrokes. A highlight field in the DISPLAY WINDOW can be moved with the cursor keys. The DIALOG BOX which lets you make changes to the macro definition looks like this:

[ ]

As the highlight field in the DISPLAY WINDOW is moved, the contents of the highlight filed is duplicated in the DIALOG BOX. You can use the PROBE edit keys to make changes, additions, or deletions to the keystrokes in the macro. You may put several keystrokes into the DIALOG BOX.

If you Edit a macro which does not exist, then the DISPLAY WINDOW only contains the name of the macro. The same DIALOG BOX as before is displayed. Key strokes are entered into the DIALOG BOX and transferred to the DISPLAY WINDOW when <enter> is typed. Using the Macro Edit command to define a new macro lets you assign pure ASCII text to a macro name. This is useful to simply save key strokes as with long complex symbolnames.

Since macros are saved in a file as simple ASCII text, they may be edited off line with your favorite text editor. The format of the macro text is described in Appendix D.

You can add comments to PROBE commands in the macro with the edit command. On a single line in the ASCII definition of a macro, all characters after a period until the end of the line are ignored during macro execution. For example, a comment is added to the first line of this LOOP While conditional macro.

<AltI>:LW.Macro to loop while D0 is not equal to 30

MACRO EXECUTION

After a Macro has been defined, it can be executed by simply typing the *AltKey* which is the macroname. You will see the macro execute on the screen. If a macro pauses waiting for you enter to enter a parameter, the Alt# for the parameter is in the lower right hand corner of the screen. The command in process which will accept the parameter is also shown on the screen.

If you want to bail out of macro execution, type Ctrl Break.

MACRO COMMAND EXAMPLES

Define a macro named *Alt*A that will load a program, display registers, and unassemble the first 5 instructions from the start of the program. The program name should be a parameter to the macro.

MD*ALT*A<enter><enter>L*ALT*0FTOC.HEX<enter>R<Esc>
U<enter>5<enter>*ALT*A

This macro can now load the file FTOC.HEX with the following key sequence.

*ALTA*FTOC.HEX<enter>

Define the same macro, but this time do not really load the file while the macro is being defined.

MD*ALT*A<enter><enter>L*ALT*0<enter><enter><Esc>R<Esc>
U<enter>5<enter>*ALT*A

Edit the previous macro to delete the U command.

ME*ALT*A<enter><enter><DnArrow><DnArrow><DnArrow>
<DnArrow><DelKey><DelKey><DelKey><DelKey><DelKey>
<DelKey><DelKey><DelKey><enter><Esc>

Define a macro named ALTB which is simply a string of characters which is the symbol named
\MAINMODULE\IOPROCEDURE\VARTEMP

ME*ALT*B<enter><enter><DnArrow>
\MAINMODULE\IOPROCEDURE\VARTEMP<enter><Esc>

Define a macro named ALTC with displays memory starting at an address passed to it from the previously defined macro ALTB. Display 15 words at this address, then display registers.

MD*ALT*C<enter><enter>DW*ALT*B<enter>+15<enter><Esc>R<Esc>*ALT*C

Define a macro *ALT*K that displays the file FTOC_IO.C if the contents of memory location FILEPTR is "I".

MC*ALT*K<enter><enter>I[FILEPTR]="I"<enter>VFTOC_IO.C*ALTK*

Define a macro which defines and activates a sticky breakpoint on writing to the long variable FAHR. After each write, check to see if the contents of FAHR is greater than the contents of the long variable FAHRMAX. If it is greater, stop the macro. If it is not, keep running the macro. Note that it would be better to define the breakpoint independently of this macro since it is defined through each loop.

MC*ALT*X<enter>Y<enter>LW[FAHR].L<[FAHRMAX].LBD0FAHR
<enter><enter><enter><enter>W<Tab>Y<esc>G<enter><enter>*ALT*X

Define a macros named 1 which will automatically open a file named ONE and display it at the last place it was displayed. By defining a macro like this for each file you want to display, you can quickly move between multiple files for display. Note that you can use the <ALT1> key to name the macro, since, because it is the macro name, it will not be interpreted as a parameter.

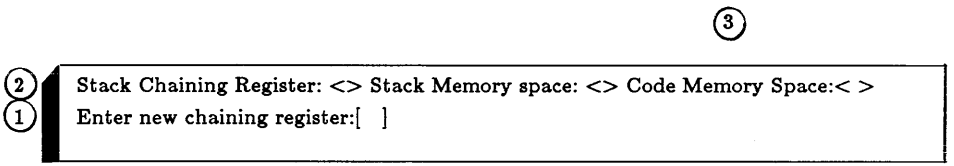
MD*ALT1*<enter><enter>VONE<enter>*ALT1*

NEST COMMAND

PROBE can analyze the stack to find return addressed so that procedure calling sequences can be determined. This command is invoked from the MENU BAR by typing:

N for Nest

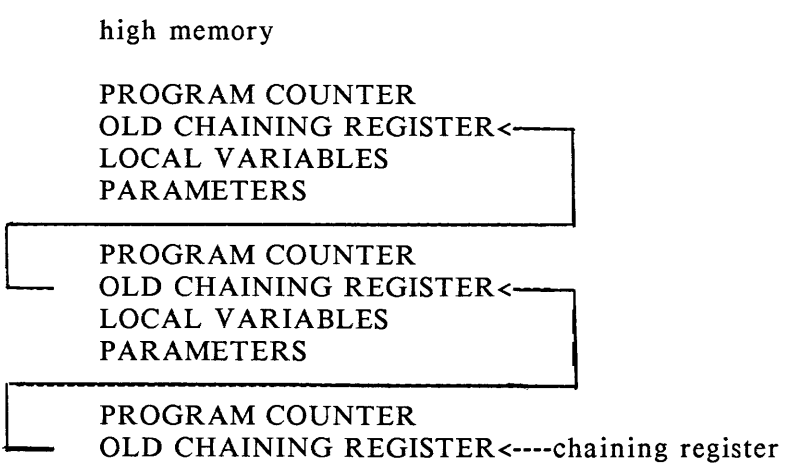
When the Nest command is invoked, the following screen appears:



(1) First you are prompted to enter the chaining register:

Enter new chaining register [ ]

Simply typing <enter> will select the default Stack Chaining register which is A6. The stack frames are assumed to look like this:



PROBE follows the chain and displays the current program counter and procedure calling sequence in the DISPLAY WINDOW. The symbols which match the calling instructions (or the closest previous symbol) are displayed along with the value.

(2) This is the default Stack Chaining Register.

(3) The default code and stack memory spaces are shown in these fields. You can change them default by typing <TAB> to get the following prompts:

Stack Memory space:

Memory space: <> {0|UD|UP|UR|4|SD|SP|CPU}

Code Memory space:
Memory space: <> {0|UD|UP|UR|4|SD|SP|CPU}

Note that in some systems, PROBE may track the stack frames into non-existent memory and a time out may occur. If it does the following message will appear:

Bus time out exception caused by access at address xxxxxxxx

You can clear this error message, as always by striking any key.

EXAMPLES OF USING THE NEST COMMAND

Display the procedure calling sequence based upon the default A6 register. A sample PROBE display is also shown.

N<enter>

PC IS 00000100=\FTOCSTART\START
CALLED FROM 00000108=\FTOCSTRT\START+00000008

QUIT COMMAND

PROBE software is terminated by typing:

Q for **Q**uit

The following prompt verifies that you really want to quit.

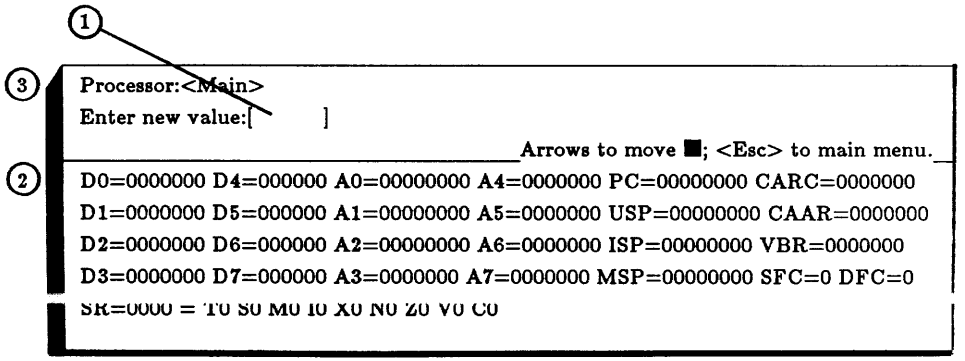
Return to DOS now: [Yes] {Yes|No}

REGISTER COMMAND

The registers and flags in the 68020 can be displayed and changed with the register command. You invoke this command from the MENU BAR by typing:

R for Register

The following screen appears:



① The DIALOG BOX prompts lets you change the value of one of the registers:

Enter new value: [ ]

② The register to be changed is indicated by the highlight field. The highlight field can be moved with cursor keys. Typing <Esc> gets you back to the MENU BAR.

③ This field shows you which processor the registers in the DISPLAY WINDOW is displaying. You can display the registers for the 68881 numeric or 68851 memory management unit coprocessors by typing <TAB> to get the following prompt:

Enter processor ID for register display:[Main]{Main|0|1}

Choose 1 for the 68881 and 0 for the 68851. PROBE must find these coprocessors in the target to display the registers. See the Motorola manuals for details on the 68881 and 68851 coprocessors.

When the target system is powered down, the starting PC and A7 (Stack pointer) registers are not automatically read when power is reapplied. You should use the "H R Y" (Hardware Reset Yes) command to get these initial register values.

The 68020 registers are defined as follows:

REGISTER/FLAG	DESCRIPTION
D0-D7	General purpose 32 bit registers.
A0-A6	32 bit address registers.
A7	Current stack pointer(either USP, MSP, or ISP)
PC	Program counter
USP	User stack pointer(A7 if S=0, M=x)
ISP	Interrupt stack pointer(A7 if S=1, M=0)
MSP	Master stack pointer(A7 if S=1, M=1)
SFC	Source Function Code -used on source operand for MOVES instruction.
DFC	Source Function Code -used on destination operand for MOVES instr.
VBR	Vector base register for interrupts
CACR	Cache control register
CAAR	Cache address register
SR	Status register
SR [T]	Trace enable (2 bit field)
SR [S]	Supervisor/User state (0=User, 1=Supervisor)
SR [M]	Master/Interrupt state (0=Interrupt, 1=Master)
SR [I]	Interrupt Priority Mask (3 bit field)
SR [X]	Extend flag
SR [N]	Negative flag
SR [Z]	Zero flag
SR[V]	Overflow flag
SR[C]	Carry flag

These register names are recognized by PROBE and can be used in expressions exactly as they are shown above. If you want to specify a hex number to PROBE which coincides with a register name, you must precede the hex number with 0. For example, D0 specifies the contents of a register, 0D0 is a hex number.

The upper byte of SR is only visible in supervisor mode. The lower byte of SR is visible at all times and may be called CCR (condition code register). This is the register that is used by non-supervisor programmers.

The display of the 68851 registers is shown below. The register names (16 and 32 bit only) for the 68851 are recognized by PROBE and can be used in expressions.

```
SRP=00000002#00000002= L/U0 LM=0000 SG0 DT2 CAL=00
CRP=00000000#00200480= L/U0 LM=0000 SG0 DT0 VAL=00
DRP=00410000#00200000= L/U0 LM=0041 SG0 DT0 SCC=FF
TC=00004000= E0 SRE0 FCL0 PS0 IS0 TIA4 TIB0 TIC0 TID0 AC=0083=MC1 ALC0 MDS3
PSR=0000=B0 L0  S0  A0 W0 I0  M0  G0  C0  N0 PCSR=8000= F1 LW0 TA0
BAC0=0008=BPE0 BSC08 BAD0=0000  BAC4=0000=BPE0 BSC00 BAD4=0268
BAC1=0000=BPE0 BSC00 BAD1=0000  BAC5=0020=BPE0 BSC20 BAD5=0000
BAC2=0020=BPE0 BSC20 BAD2=0000  BAC6=0000=BPE0 BSC00 BAD6=0000
BAC3=0000=BPE0 BSC00 BAD3=0020  BAC7=0000=BPE0 BSC00 BAD7=0000
SRP=00000002#00000002= L/U0 LM=0000 SG0 DT2
```

The display of the 68881 registers is shown below. The register names (16 and 32 bit only) for the 68881 are recognized by PROBE and can be used in expressions.

```
FP0=+NAN#7FFF0000#FFFFFFFF#FFFFFFFFFP4=+NAN#7FFF0000#FFFFFFFF
FP1=+NAN#7FFF0000#FFFFFFFF#FFFFFFFFFP5=+NAN#7FFF0000#FFFFFFFF
FP2=+NAN#7FFF0000#FFFFFFFF#FFFFFFFF FP6=+NAN#7FFF0000#FFFFFFFF
FP3=+NAN#7FFF0000#FFFFFFFF#FFFFFFFF FP7=+NAN#7FFF0000#FFFFFFFF
FPSR=00000000= N0 Z0 I0 NAN0    S0 Q00
BSUN0 SNAN0 OPER0 OVFL0 UNFL0 DZ0 INXO0 INXI0
Accrued     IOP0 OVFL0 UNFL0 DZ0  INX0
FPCR=00000000= BSUN0 SNAN0 OPER0 OVFL0 UNFL0 DZ0 INXO0 INXI0
Precision=Extended  Round-toward=Nearest
```

EXAMPLES OF USING THE REGISTER COMMAND

Change the value of register D1 to AAAA.

R<DownArrow>AAAA<enter>

Display registers and change the PC to the value specified by the macro *ALT*B.

R<RtArrow><RtArrow><RtArrow><RtArrow>*ALT*B<enter>

Change the value of the T flag to 1 and the value of the S bits to 7.

R<DnArrow><DnArrow><DnArrow><DnArrow><DnArrow>
<RtArrow><RtArrow>1<enter><RtArrow><RtArrow><RtArrow>7
<enter>

Change the least significant bit of the CACR register to 0 and leave all other bits in the register unchanged:

R<rt arrow><rt arrow><rt arrow><rt arrow>
<rt arrow>CACR&11111110<enter>

SINGLE STEP COMMAND

The single step command is very powerful in PROBE and lets you do many things. You can step instructions one at a time or in multiple steps, display windows while stepping, and continue stepping while events are true in the system. You can start single stepping program execution by typing:

S for Step

The subcommands for the Step command now pop up and the screen looks like this: (This display is for Source not PROBE)

```
Assembly-language
Source-level
                                        <Esc> to main menu
```

The subcommands are shown here. For PROBE, there are no subcommands.

Command	Sub command	Operation
Step	Assembly	Assembly language single step
	Source	Source level single step. This subcommand is only available with the 68020 Source Probe option.

When either subcommand is invoked the following screen appears:

```
                                                    3
(2) Start address: <    >                Program space: <User>
(1) Enter start address: [ ]
                                <Tab> to next field: <Esc> to main menu
```

(1) The first DIALOG BOX prompts you for the start address. If you simply type <enter>, the default start address (current program counter) is used. After the starting address has been selected, the screen below appears.

(2) This is the value of the current default start address.

(3) This is the current default program space. You can change this default by typing <Tab> to invoke the prompt which changes it.

Once single stepping has started, a screen similar to this appears:

```
(5) Steps to take for each <Enter>: <001
(6) After <Enter>, step while: <False>
(7) "B" to run to ■; "J" to run to instr after ■: <Enter> to step from 00000400
        <Tab> to next field above: PgUp/Dn Arrows to move ■; <Esc> to main menu
D0=0000002 D4=00000000 A0=0000072A A4=00000000 PC=00000400 CACR=0000000
D1=00000001 D5=00000000 A1=0000077C A5=00000000 USP=00000000 CAAR=00000
D2=00000000 D6=00000000 A2=00000722 A6=00000700 ISP=000006DC VBR=0000000
D3=00000000 D7=00000000 A3=00000000 A7=00006DC MSP=0000000 SFC=0 DFC=0
SR=2704=T0 S1 M0 I7 X0 N0 Z1 V0 C0

(3) \START:
(1) 00000400  LEA.l__   (00000708),A7
(2)  ^ ^ ^ ^    Op 1 value=00000708, address=00000708=\FtoCStrt\STACKTOP
    00000406 MOVE.l     A7,A6
    00000408 JSR        (00000444)
    0000040E BRA        00000400
(4) \FtoCStrt\DOSIGNEDDIVIDE:
    00000410 TST.l      D1
    00000412 BEQ        0000041A
```

(1) The instruction which matches the current program counter is indicated by the blinking cursor. The address, opcode and operands are shown on this line. *This instruction is not executed until the <enter> key is typed.* Note that the next five instructions in memory are shown below this instruction.

(2) These are the values of the operands of the current instruction. PROBE calculates the address of the operand, if appropriate, and retrieves the contents of memory. It then shows you the calculated operand address, any symbol or near symbol which

matches this calculated address, and the contents of the memory at this calculated address. This shows you the operand values of the current instruction before it actually executes (i.e. before you type <enter>). This saves you from having to bail out of the single step command to see the values of the operands. The ^^^^^ on this line points to the instruction for these operands ,i.e. the instruction just above the current line.
****

Note that if the operand is pointing to a memory mapped IO device which changes its contents when it is read that PROBE's advance read of the operands may affect your program.
****

③ Note that when you start the single step command, a highlight field spans the current instruction and its operands. This highlight field serves as a second cursor. The highlight field can be moved with the PgDn/PgUp and cursor keys. *When moving the highlight field up the DISPLAY WINDOW past the current program counter (i.e. blinking cursor) the previous single stepped instructions are displayed (not the code previous to this instruction.* When moving the highlight field down the DISPLAY WINDOW below the current program counter, your program (as disassembled from memory or viewed through a source code file) is shown. The blinking cursor remains at the instruction pointed to by the program counter and does not move with the cursor keys. Typing Ctrl PgUp moves the highlight field to the blinking cursor (i.e. next instruction to be executed.).. If the highlight field disappears when you step and does not come back, type <Esc> to regain control (if you get an error see later in this chapter for details.)

④ PROBE precedes an instruction address with a symbol if one matches the address. If an operands match symbols, the symbols are shown to the right of the operand.

⑤ This field determines the number of instructions to step for each typed <enter> key. See the section STEP COUNT later in this command.

⑥ See the STEP WHILE CONDITION section later in this command for this field.

⑦ See the next section BRANCHING... for this field.

BRANCHING TO INSTRUCTION AND AROUND SUBROUTINES DURING SINGLE STEP

During single stepping, PROBE can do more than just take the next single step when <enter> is typed. Two other keys, B and J, can be used to control program flow.

If you are paging through your program during single step and position the highlight field at an instruction, typing the B (for break) key will run the program from the current program counter and stop at this highlighted instruction. This is easier and faster than getting out of the single step command and setting a breakpoint at this instruction.

You can also position the highlight field on any instruction and type J (for Jump) to run the program from the current program counter until the instruction at the next *address* after the highlighted instruction. Why is this different than the B key. If you are currently at an instruction which is going to call a subroutine, and you want run through it real time until the program returns, then type the J (for Jump) key instead of the <enter> key. This will also work when you have already stepped into the subroutine by mistake. Simply move the highlight field back up the screen to the jump instruction which got you into the subroutine and type J to execute until the instruction after the jump. You probably cannot easily use the B key for this case (unless the routine is small and fits on the screen) since the code you were executing before the jump is no longer on the screen - but the jump instruction is.

If you type B or J and the highlight disappears and does not come back, then PROBE did not reach the target instruction and is still executing the program. To return control to the Step command, type <esc> and the screen will be updated to show you where you are currently executing.

STEP COUNT

Single stepping can be by a single instruction or multiple instructions. This lets you go through your program in larger steps. You can change the Step count by typing the <TAB> key until the following prompt appears:

Enter number of steps to be taken for each <enter>:[ ]

Simply typing <enter> at this prompt does not change the number of steps taken and returns you to single stepping.

STEP WHILE CONDITION

Single stepping can be programmed to continue automatically While a a specified condition is true. You can set this condition by typing <TAB> until you get the following prompt:

Enter condition to test for end of stepping: [ ]

The condition is a boolean expression which was defined at the start of this chapter in the section called BOOLEAN EXPRESSIONS. After the While condition is entered in response to the above prompt, typing <enter> will launch the single stepping. Stepping will continue automatically as long as the condition is true. The condition is tested after each single step. If the While condition goes False, stepping stops. You are still in the single step command, however, and typing <enter> will again take the number of steps shown in the Count field. Once the While condition is False it is cleared from its field on the PROBE screen. If you type <TAB> or <ESC> during the automatic single stepping, then single stepping will stop and the While condition is set to False.

The following is a summary of the keys which operate in the single step command:

Key	Operation
PgUp or Up arrow	Display previous pages (up to 4k buffer) of single step operations. Highlight field moves with display but not blinking cursor
PgDn or Dn arrow	Unassemble upcoming instructions or scroll forward through source file. Highlight field moves.
<enter>	Execute the number of instructions specified in the Count field
b	Set an Execute breakpoint at the highlight field and do a Go command until this breakpoint is encountered.
j	Run real time and set a breakpoint at the instruction following the highlighted instruction
ESC	Terminate single stepping and pop down all windows.
<TAB>	Invokes DIALOG BOXES to change step count or step While condition

WATCH WINDOWS DURING SINGLE STEP

While single stepping, you can pop up one or more Watch Windows. These windows are defined by the Window command and are popped up with a single *AltKey*. These windows are updated after each single step, therefore, you can keep an eye on anything in the target while you are single stepping. To single step while the Watch Windows are active, wait until you have invoked the step command and the highlight field appears before popping up the window.

SOURCE CODE STEPPING

You can single step the program via source statement lines by choosing the Source subcommand for the Step command. During the source stepping process, PROBE finds highlevel language modulename\line numbers in the symbol table which match executable instructions. It then uses the files assigned to the modulenames from the sYmbol-Assign-module-to-file command or those derived from loading an Initialization file. A sample DISPLAY WINDOW is shown here:

```
Steps to take for each <enter>: <0001>
After <Enter>, step while: <False>
"B" to run until ■; "J" to run until instr after ■; <Enter> to step from 00000000
____<Tab> to next field above; PgUp/Dn, Arrows to move [ ]; <Esc> to main menu
12. *c__temp = f__temp - 32;
113. *c__temp = *c__temp * 5;
114. *c__temp = *c__temp / 9;
115.
116. return;}
41.__ printf("%5d %5d\n", fahr, celsius);
42.
43. /*--------------------------------------------------
44. Go to next line of table.
45. --------------------------------------------------*/
46. fahr = fahr + step;
```

(1) The line of source code which matches the current program counter is indicated by the blinking cursor. This instruction is not executed until the <enter> key is typed. The lines of source code after this instruction are also shown in the DISPLAY WINDOW.

(2) This is the high level language linenumber for this line of code.

(3) Note that when you start the single step command, a highlight field spans the current instruction. This highlight field serves as a second cursor. The highlight field can be moved with the PgDn/PgUp and cursor keys. When moving the highlight field up the DISPLAY WINDOW past the current program counter (i.e. blinking cursor) the previous single stepped instructions are displayed (not the code previous to this instruction. When moving the highlight field down the DISPLAY WINDOW below the current program counter, your source code is shown. The blinking cursor remains at the instruction pointed to by the program counter and does not move with the cursor keys. Typing Ctrl PgUp moves the highlight field to the blinking cursor (i.e. next instruction to be executed.)

The same fields which were described in the Step Assembly language command apply to the Step Source code command. If you use the B key to branch to a line of source code in the DISPLAY WINDOW

which does not represent an executable instruction (i.e. it may be a comment), then PROBE will stop at the first executable line of source code after the line where you typed the B.

You may also limit the modules through which you single step while executing all code outside these modules in real time. This is done using the sYmbol command and choosing the Source-step-module-selection subcommand. This provides an automatic method of ignoring parts of you code which are automatically debugged and which you do not want to step into.

WHAT HAPPENS DURING SINGLE STEPPING

Assembly language stepping

During single step, PROBE/3 performs bus cycles in the target system before actually executing the desired instruction when you type <enter>. There are 3 reasons these cycles are needed:

1. PROBE displays the next 5 instructions in memory from the current program counter. To do this, PROBE reads the target system memory (or MAP RAM) and disassembles the memory into instructions.
2. PROBE displays the contents of memory operands that are referenced by the instruction to be executed. To do this, the PROBE reads the current values of the operands from memory in the target system.
3. PROBE/3 implements Step Assembly language by setting HWexecute breakpoints at the next instruction to be stepped. PROBE analyzes the current instruction (i.e. where the blinking cursor is) and sets a HWExecute breakpoint at all possible points that this program could go to. When you type <enter>, PROBE does a Go and starts executing your program until one of these HWExecute breakpoints is hit. To determine the address of these breakpoints, the PROBE reads target system memory (or MAP RAM). Some examples are:
   A) Indirect jumps through memory:
      JMP ([5,A0],D0,8) - [5,A0] will be read
   B) Instructions that may generate exceptions:
      MOVE #2700, SR - Priv. excep. vector read

Bus errors during Single Step

Only one bus cycle will occur in the target system for each of the references to target memory (or MAP RAM) described above. If a bus error happens during one of these references, control will return to PROBE before any stack operations or exception vector reads occur. A message is displayed on the 68020 Probe screen indicating the memory address of the cycle at fault and processing of the step command continues for each of the previous 3 listed cases.

1. Disassembly and display of the current and the following instructions is not completed. Stepping can continue even if reading the current instruction caused the bus error. If the instruction to be stepped caused the bus error, no breakpoints will be set because PROBE could not determine what the current instruction was, and therefore could not determine the possible next instructions to be executed.
2. Printing memory based operands of the next instruction: A question mark is printed as the value of the operand. Stepping may continue by typing <enter>.
3. Determining possible next instructions to be executed: A breakpoint is not set for the next instruction address that was being calculated. Stepping may continue by typing <enter>.

Bus timeout during Single Step

If the target system does not return DSACK's during single step operations, PROBE reports a "bus timeout" error message. There are several ways in which this could happen.

1. Since PROBE does a Go and sets HWExecute breakpoints during single step, the processor may get an interrupt. If the interrupt service routine in the target crashes the target system hardware may not return DSACK's. To eliminate this problem, use the Hardware Interrupt No command to disable interrupts to the 68020.
2. Many target systems boot up with PROM shadowing low memory and then switch the PROM to high memory. If the PROBE attempts to step with the PROM still shadowed on low address locations, a bus timeout error may appear. This is because the PROBE steps by setting a breakpoint on every instruction that

may execute after the stepped instruction executes. To determine the locations of each of these target instructions, the PROBE may reference target system memory. For example, if the instruction might generate an exception, a breakpoint is set by PROBE on the first instruction in the exception handler. Therefore, if the first instruction to be stepped is "Move #2700, SR", it is possible that a privilege exception may occur. Before stepping, the privilege vector is read to determine what the next instruction location would be if the exception occurred. If the PROM has not been "unshadowed", the hardware may not respond to that location and a 68020 Probe timeout may occur.

Interrupts during Single Step

For Assembly language single step, PROBE starts program execution at the current Program Counter and sets HWExecute breakpoints at all possible next instructions. This lets the target processor service real time interrupts in the background between steps without interfering with the single step process. If you do not want to service interrupts between steps, use the Hardware Interrupt command to lock out interrupts from the target. If you want to single step an interrupt procedure , use the Go command to set a breakpoint at the start of the interrupt procedure and then start single stepping. If the target services an interrupt and does not return, then the highlight field will disappear and the processor will continue to execute code in the target. You can regain control in this case by simply typing the <Esc> key. If the interrupt procedure crashes and causes a double bus fault or stops the processor while you are stepping, typing the <Esc> key will regain control and you will probably get the following message:

No address strobes to target

This means that the 68020 is shut down and you can only regain control by doing a Hardware Reset command.

PROBE real time trace logic is operating during program execution while single stepping. If the 68020 is servicing interrupts during single step, then real time trace of program execution between steps is available for display. This is also a useful display to look at if the "No address strobes" message occurs during single step.

No Address Strobes during Single Step

When the 68020 gets a double bus fault, the HALT line is held low by external logic, or is held up the the BGACK, then the 68020 does not produce address strobes to the target and the following message is displayed.

No address strobes to target

These operations could happen during single stepping. If the error is caused by the BGACK line, use the Hardware Dma Enable command to mask the BR line from the 68020. If the error is caused by interrupts which are happening in the background between steps, use the Hardware Interrupt Enable command to mask interrupts from the 68020. If the error is caused by the Halt line, use the Hardware Halt line enable command to mask this line from the 68020.

Source stepping

PROBE implements source level single stepping by setting a software breakpoint at each instruction which has a source line number, then executing a GO command. This is why eliminating modulename/line number information in the symbol table or ignoring it with the Symbol-Step-Module command limits source stepping.

EXAMPLES OF USING THE SINGLE STEP COMMAND

Step by Assembly language the program starting from location \main. Make the single step command operate while D0 is <> 5.

SA\MAIN<enter><Tab><Tab>D0 <> 5<enter><enter>

Start stepping by Assembly language from the current PC. Set the step count to 5. After stepping has started pop up the window assigned to the key *ALTZ*

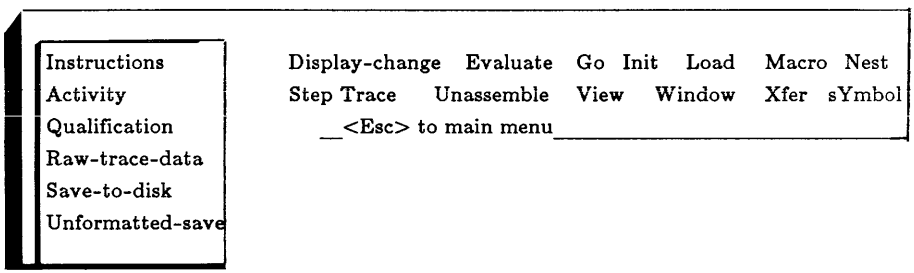
SA<enter><Tab>5<enter>*ALTZ*<enter>

TRACE COMMAND

This PROBE command displays the real time program execution before (and optionally after) the breakpoint was detected. The trace command is invoked from the MENU BAR by typing:

T for Trace

The subcommands for the Trace command now appear in a MENU BOX and the screen looks like this.



The subcommands are:

Subcommand	Operation
Instructions	Display trace data with prefetch filtered
Activity	Display trace with prefetch not filtered
Qualification	Qualify trace data to a region of memory
Raw-trace-data	Display trace data in hex format
Save-to-disk	Save data to disk human readable form
Unformatted-save	Save data to disk machine readable form

TRACE INSTRUCTIONS

The Trace Instructions command provides the most useful form of the trace display. PROBE analyzes the trace data which was collected in real time and processes it to produce an easy to understand trace display. The processing does the following:

1. The 68020 pipeline (not cache) has been modeled in PROBE software. PROBE analyzes the trace data and tosses out prefetched but unexecuted instructions so you don't have to guess which instructions executed and which did not.
2. PROBE analyzes the trace data and displays the memory reference cycles directly under the opcodes which executed them. If PROBE did not do this, you would have to do this yourself mentally. This is because the 68020 pipeline fetches opcodes many bus cycles earlier than the memory reference cycles which go with those opcodes.

The trace shows assembled opcodes and operands, data transferred during execution cycles, stack operations, and interrupt cycles. Program symbols are included in the trace data to make the identification of program operation easy to understand. During program execution, the cache could have been enabled or disabled during program execution with the Go command. Trace data is viewed after the detection of a breakpoint or emulation has been terminated with the <Esc> key. If the cache were disabled during program execution, a screen similar to the following appears:

```
Search address: <any>        Space:<Any>          Verb:<Any>      Data:<any>
<.... .... .... .... .... .... .... ....>
Begin search of trace:[No] {Yes|No}
__PgUp/PgDn/Arrows move within memory; <Tab> to next field; <Esc> to main menu
Cache enabled during execution.        Trace not qualified.
 45.       Fahr <= Upper;
 000004C0 :SP       MOVE.l          (A2),D0
 00000722 :SD          READ - 0000                  \FAHR
 00000724 :SD          READ - 0050                  \FAHR+00000002
 000004C2 :SP       CMP.l           (0000071A),D0
 0000071A :SD          READ - 0000                  \UPPER
 0000071C :SD          READ - 0064                  \UPPER+00000002
 000004C8 :SP       BLE             0000047E
 50        Compute (Fahr, &Celsius);
 0000047E :SP       PEA.l           (00000726)
 000006E8 :SD          WRITE - 00000726             \FtoCStrt\STACK+000000E0
 00000484 :SP       MOVE.l          (A2),-(A7)
B 00000722 :SD          READ - 0000                  \FAHR
S 00000724 :SD          READ - 0050                  \FAHR+00000002
 000006E4 :SD          WRITE - 00000050             \FtoCStrt\STACK+000000DC
```

(1) This line shows the instruction address, memory space, opcodes and operands.

(2) Additional memory reference cycles used by the instruction are shown on this line with their address, type of cycle, and data on the bus during the cycle. If the address of the memory reference cycle matches or is near a program symbol, the symbolname is shown to the right.

(3) A "B" in this column indicates that this is the cycles which caused the breakpoint. If a "Pass count" condition was used in setting the breakpoint, there may be multiple "B"'s in the display.

(4) A "S" in this column indicates that this cycles matches the fields specified for a search. Note that both B and S can be in this column.

⑤ Procedure names or high level language line numbers which match the address field of an instruction are shown before the instruction. If using the SOURCE rather than the PROBE version of the software, then the actual program source code precedes the assembly language.

⑥ This line shows you if the trace data was taken with the cache enabled or disabled and if the trace qualify condition was enabled or disabled during the trace. Controlling the cache and trace qualification is described later.

⑦ You can scroll and search for data in the trace display. This is described later in this command.

TRACE ACTIVITY

The Trace Activity command shows you the trace display in a form similar to the Trace Instructions. The difference is that PROBE does not filter out unexecuted prefetch or place memory reference cycles directly under the instructions which caused them. It simply displays the opcodes and memory reference cycles on the bus in the order in which they occurred. *When this command is invoked the following message appears on the screen to remind you of these facts.*

> Data cycles were not matched with prefetched instructions for this display.
> There are 3 major side effects of this:
> 1) Data cycles will probably not appear with the instruction that generated them.
> 2) Instructions following those that can cause a transfer of control may not have actually been executed.
> 3) The trace display software may print the target of a jump instruction as the wrong word (low vs. high) of a 4-byte instruction fetch.
> Press any key to begin display

The Trace Activity display can happen even with the Trace Instructions command for certain conditions. This could happen for the following reasons:

1. cache was enabled when the trace data was taken
2. trace qualify condition was active when the trace was taken
3. PROBE could not accurately analyze the the trace data to filter out prefetch and tie memory reference cycles to opcodes

TRACE RAW

The Trace Raw command is a hex display of the real time trace data. The display of the data appears under these columns:

```
Cycle  Address  Spc  Data  Strb  RW  If  Rn  Ep  Bk  BP  Rs  P  G  BgBe  Log  G24  G32
```

Cycle	number of the cycle (0-7ff) in the trace data.
Address	32 bits of 68020 addresses.
Spc	base 10 decode of the Function Code lines
Data	32 bits of 68020 data. For any given cycle the data here may not be valid. The next field, Strb, indicates which portion of the data bus contains valid data.
Strb	binary, active lo field. 0111 indicates that only the 8 most significant data bits (d24-d31) are valid; 1100 (d0-d15) valid. These bits are not valid at all during eps.
RW	the r/w line; 1 = read, 0 = write. It indicates that a read from super or user program space occurred
Rn	indicates the PROBE was emulating the target or stopped for interrogation mode by the PROBE user interface.
Ep	indicates transition from the emulation mode to the interrogation mode.
Bk	indicates that a hardware breakpoint was detected.
BP	when this field is a 1, it indicates that hardware breakpoints were active.
Rs	reserved
P	performance mode timer overflow bit.
G	when this field is a 1, it indicates that indicates a guarded access occurred.
Bg	bus grant
Be	bus error
Log	hex field that shows the state of the PROBE logic lines.

G24
G32

hex fields that contains a collection of bits described as follows: The g24 and g32 fields contain misc. bits that were not decoded on the trace display because of lack of room. The bits are labeled general 24 (lsb) thru g31 (msb) and g32 (lsb) thru g39 (msb).

```
G24 IS DSACK0/
G25 IS DSACK1/
G26 IS IPL0/
G27 IS IPL1/
G28 IS IPL2/
G29 IS USED BY POD S/W
G30 IS SIZ0
G31 IS SIZ1
G32 IS USED BY POD S/W
G33 IS OCS
G34 IS CDIS/
G35 IS IPEND/
G36 IS AVEC/
G37 IS RMC/
G38 IS USED BY POD S/W
G39 IS HALT/
```

SAVE-to-DISK

This command lets you save the trace to a disk file. You could also use the Xfer Log command to open a log file to disk then display the trace data, but this is much faster. The data is saved to disk in a human readable form. This command prompts you for the filename to save the data.

UNFORMATTED-SAVE

This is a command which you use to save information to a disk file which you can send to Atron. Atron has modeled the 68020 pipeline in order to filter unexecuted instructions and tie memory reference cycles to instructions. If this model has bugs, you can help us find them by saving trace data when the Trace Instructions command fails (i.e. gives you Trace Activity display instead). With this trace save data, we can exactly recreate the state of the PROBE when it fails in your target. There are other situations when the PROBE appears to be doing "something funny" in your target which we can identify at Atron with this Trace Save data. When something gets screwed up in the PROBE, do a Trace Unformat Save and send it to:

Atron
Attn: Technical Support
20665 Fourth St.
Saratoga, Ca. 95070

When we receive the information, we will report our results, work arounds, or corrections to you as soon as we find them.

SEARCHING TRACE DATA

While you are in any of the Trace commands which display data in the DISPLAY WINDOW, the following keys let you scroll through the trace display:

PgUp/PgDn and cursor keys move you through the trace data.
Ctrl PgUp moves you to the start of the trace data.
Ctrl PgDn moves you to the end of the trace data.

PROBE also has a built in editor which lets you search trace data fields for specific events. The DIALOG BOX prompts you with:

Begin search of trace:[No]{Yes|No}

If you type Y, PROBE starts searching from the current location in the DISPLAY WINDOW until the end of the Trace data for matches between the trace data and the other fields in the DIALOG BOX. If you type N, then PROBE lets you set these other fields in the DIALOG BOX. You are given the following series of prompts which let you set these fields for the search of the trace data.

Enter address to search for: [ ]
Enter end address of range to search for:[ ]
Enter don't care bits (".|"X")
Enter memory space to search for:[Any] {Any|0|1|2|3|4|5|6|7|8|9}
Enter verb to search for: [Any] {Any|Read|Write}
Enter data size to search for:[Any] {Any|Byte|Word|Long}
Enter data value to search for:[ ]
Enter don"t care bits(".|"X")
Begin search of trace [No] {Yes|No}

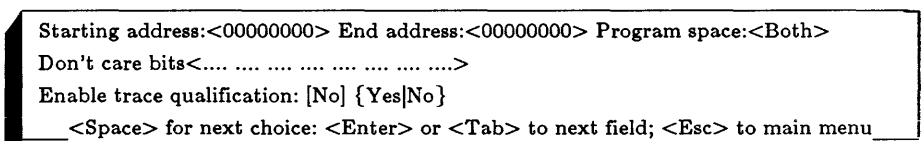
Typing <enter> in response to any of these fields takes the default in the < >. If a search is completed successfully, the trace data is positioned in the DISPLAY WINDOW with the character "S" on the line which matches the search. You can continue the search from here by simply typing Y again in response to the prompt. If the search is not successful, the following message is displayed:

Specified values not found from current locations to the end of trace.

The search fields you specify stay as defaults for the next Trace display. There is an S in each line which matches the search condition in the trace display. The search can be terminated at any time during the search with the Ctrl Break key.

TRACE QUALIFICATION

The 68020 PROBE has one qualified trace region which lets you limit the real time trace data. This lets you optimize the trace data by including only interesting trace information. A qualified trace region saves instruction fetches and their following memory reference cycles into the real time trace memory only if the fetched address falls between the starting and end addresses of the qualified trace region. The cache is always disabled while the program passes through the qualified trace region. Outside of the region, the cache can be enabled or disabled depending upon the control you set with the Go command and/or the hardware control of the 68020 by the target (see Controlling the 68020 cache). With the cache normally enabled, you can have the simultaneous benefit running your application as fast as possible while viewing more detailed trace information in the qualified trace region. When the Trace Qualified is selected the following DIALOG BOX appears:



(1) First you are prompted to enable or disable the trace qualification during execution:

Enable trace qualification: [No] {Yes|No}

(2) If you pick Yes for trace qualification, you are then prompted for the starting then ending address of the qualified trace region:

Enter new starting address :[ ]
Enter new end address :[ ]

You may enter any type of address expression. The end address may also be of the form:

+ number

In this case the end address becomes start address+number.

③ If you type <Tab>, you are then prompted for the program space to be used in trace qualification. In this case, the space is limited to the choices shown in the prompt:

Enter program space for qualification: [Both] {User| Supervisor| Both}

Note that since the 68020 does a lot of prefetching of instructions, that you should not put the difference between the starting and ending address less than 20 bytes. In addition, to see the memory reference cycles of a particular instruction, you should not make the end address of the qualified trace regions closer than 12 bytes past the end of the instruction. If you do, the trace may turn off too soon because of the 68020 prefetch and you will miss the details of the memory reference cycles for the instruction.

Real time trace data taken with a qualified trace region enabled can be displayed with the Trace Activity command but not the Trace Instructions command.

CONTROLLING THE 68020 CACHE

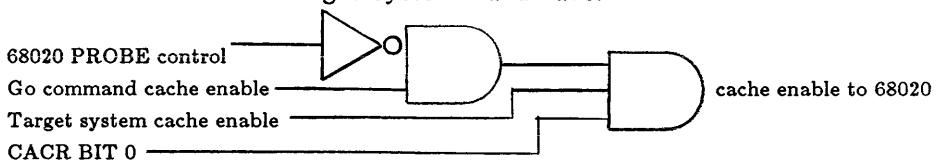
There are four things which control the state of the cache during program execution:

Go command cache enable field
Hardware line in the target
Bit 0 of the CACR register
PROBE Trace Qualify condition

1. PROBE can disable the 68020 cache at the start of program execution with the Go command. In this case, PROBE can display Trace Instructions since instructions are always fetched from the bus. If the cache is disabled with the Go command, the cache cannot be enabled via control of the target system hardware or CACR bit.

2. PROBE can enable the 68020 cache at the start of program execution with the Go command. In this case, PROBE can display Trace Activity but not Trace instructions. This is because 68020 may execute repetitively from the cache without using the bus. If the cache is enabled with the Go command, the cache can still be disabled via control of the target system hardware or register CACR bit 0 or a qualified trace region.

Here is a simplified logic diagram of how the cache is controlled by the PROBE and the target system hardware.

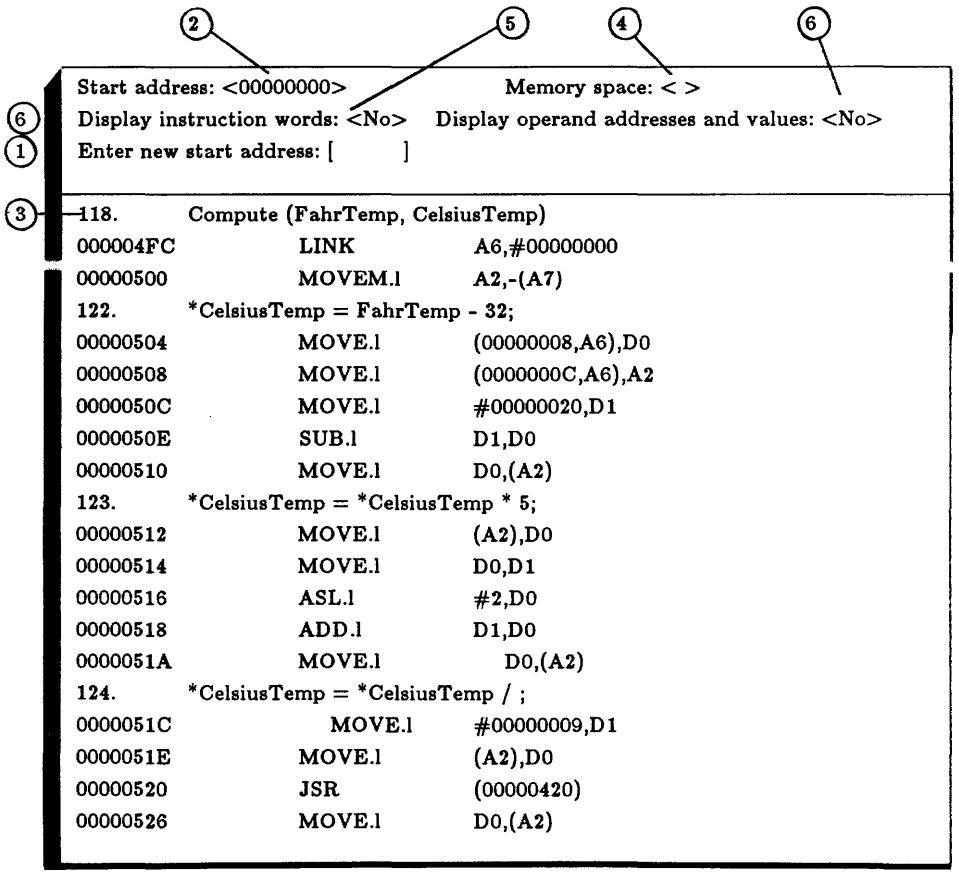


UNASSEMBLE COMMAND

Memory can be displayed as 68020 assembly language instructions with the Unassemble command. You can scroll through memory to display code. This command is invoked from the MENU BAR by typing:

U for Unassemble

A screen similar to the following screen pops up:



① The first DIALOG BOX prompts you for the starting address of the memory to disassemble:

Enter new start address: [ ]

If <enter> is typed with no start address then the default start address is assumed. Next you are prompted for the number of instructions to unassemble:

Enter number of instructions:[ ]

If <enter> with no input is typed for this field then the default is a screenful of instructions. This prompt remains on the screen and the next page full of instructions is displayed each time <enter> is typed. If you type the PgDn or Down Arrow, then more unassembled instructions are shown in the DISPLAY WINDOW. If you type the PgUp key, previous instruction are shown, but you cannot go previous to the very first instruction you unassembled since executable code in this direction is indeterminate.

② This is the current default start address. It is current program counter if a Go or Step command has been executed previous to this command. It is set to the last address of a previous Unassemble command display if no Go or Step command have been recently executed.

③ Note that symbols or linenumbers which match the address fields are included in the unassembly to simplify the display. If you are running the SOURCE level version of PROBE, the source code which is associated with the unassembled is also shown in the DISPLAY WINDOW.

MORE DISPLAY DATA DURING UNASSEMBLE

Several other fields can be set which control the information being displayed by the Unassemble command. Typing <TAB> will bring up the prompts which let you set these fields which are described below.

④ The memory space default is shown in this field. The following prompt for this field lets you change it.

Memory space: < >: {0|UD|UR|4|SD|SP|CPU}

⑤ The hex equivalent for the unassembled instruction can be shown along with the instruction by answering yes to this prompt.

Display instruction words: [Yes} {Yes|No}

The reason that this is a choice rather than a default is that the much more of the DISPLAY WINDOW is taken up displaying this additional information which is needed in only a few instances. If Yes is chosen for this prompt then the instruction looks like this:

```
instruction address     hex equivalent of instruction
^^^^^^^^                        instruction
```

The ^^^^^^^ indicates which instruction address is tied to the instruction in the DISPLAY WINDOW.

⑥ The operand addresses and values for the unassembled instruction can be shown along with the instruction by answering yes to this prompt:

Display operand addresses and values: [No] {Yes|No}

The reason that this is a choice rather than a default is that the much more of the unassembly display is taken up displaying this additional information which is needed in only a few instances. Note that operands in the display are based upon the current contents of memory and registers. The operand values may be very different when the instruction actually executes. Use this option cautiously - when PROBE references target memory a bus error may caused. If Yes is chosen for this prompt then the disassembled instruction looks like this:

```
                        instruction
^^^^^^^^  first operand address and/or value [symbol matching operand]
^^^^^^^^  second operand address and/or value [symbol matching operand]
```

Program symbols and linenumbers which match the operands are shown in this display. If Yes is chosen for the prompts in 5 and 6 then the unassembled instruction looks like this:

instruction address hex equivalent of instruction
^^^^^^^^ instruction
^^^^^^^^ first operand address and/or value [symbol matching operand]
^^^^^^^^ second operand address and/or value [symbol matching operand]

NOTES ON THE DISASSEMBLER

If an A trap instruction is encountered by PROBE (i.e. 0000Axxxx), then PROBE disassembles the instruction as follows:

? Axxxx = symbolname

The A trap is often used as a subroutine call. By associating a symbolname with your A trap, then PROBE can easily show you the name of your subroutine during disassembly. This is very useful in a target system such as the Apple MAC II (tm).

EXAMPLES OF THE UNASSEMBLE COMMAND

Unassemble memory starting from line number 76. Display the operands along with the instructions. Display a screen full.

U#76<enter><Tab><Tab><Tab>NY<enter>

A screen like the one shown here will be displayed.

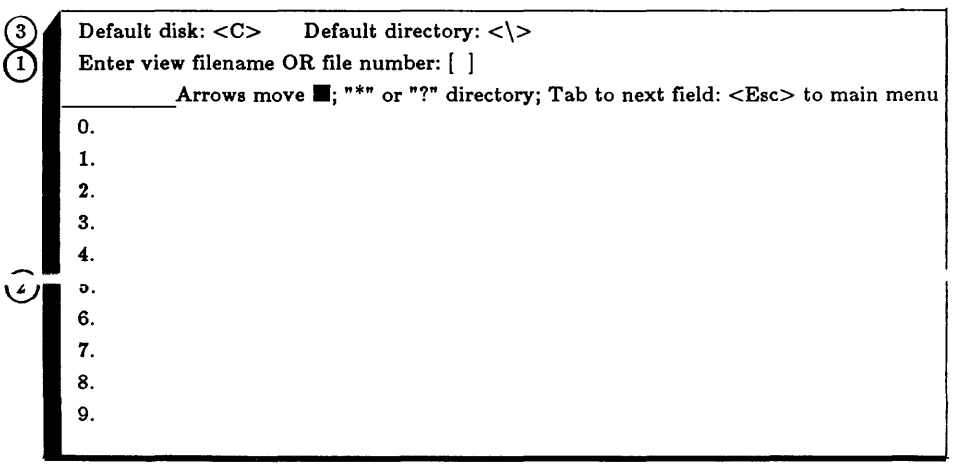
```
\main#76
000AD36  MOVE.L     (000000D6),D0
^^^^                   OP1  value=0000000A, address=000000D6= \FAHR
0000AD3A            CMP.L              (000000D2),D0
^^^^                   OP1  value=00000064, address=000000D2= \UPPER
                    OP2  value=000000A
0000AD3E            BGT                0000AD68
^^^^                   OP1  address=0000AD68 = /FTOCM#49
0000AD40            MOVE.L             #000000D8,-(A7)
^^^^                   OP2  address = 0000F586
0000AD46            MOVE.L             (000000D6),-(A7)
^^^^                   OP1  value=0000000A, address=000000D6= \FAHR
^^^^                   OP2  address = 0000F574
```

VIEW COMMAND

Files can be viewed from PROBE while debugging with the VIEW command. This command is invoked from the MENU BAR by typing:

V for View

The following screen appears:



① The DIALOG BOX first prompts you for the filename or file number:

Enter view filename: [ ]

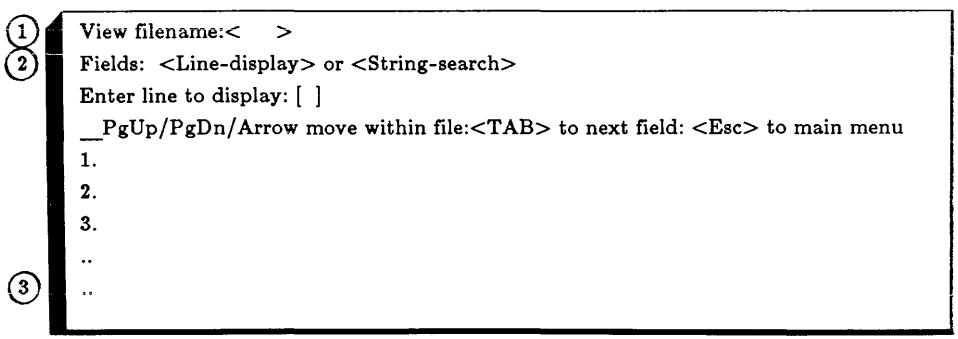
If you do not include the drive and directory for the filename then the defaults shown on the screen will be used. If you cannot remember the name of the file, then use the wild card capability of PROBE to display files.

② The filenames you have previously viewed are shown in the DISPLAY WINDOW and they are assigned a number. Moving the highlight field to the filename and typing <enter> selects the file. If there are already 10 files assigned on the screen and you want to open another, you will be prompted for the number of the file (0 to 9) close. When a file is opened and displayed in the

DISPLAY WINDOW, it is automatically positioned to the point in the file where it was previously viewed. This lets you move quickly between 10 files without repositioning the cursor in each file.

③ This is the default drive and pathname Type <TAB> to get the DIALOG BOXES which let you change the defaults.

Once the filename is chosen, the file is displayed in the DISPLAY WINDOW shown here.



① This field shows you the name of the file shown in the DISPLAY WINDOW. You can use the cursor and paging keys to reposition yourself within the file. Typing <Ctrl><PgUp> or <Ctrl><PgDn> will move you to the beginning or end of the file.

② The DIALOG BOX prompts you to:

Enter line to display:[ ]

The linenumber you chose in the displayed file is positioned to the center of the DISPLAY WINDOW. By typing <Tab> you bring up the next DIALOG BOX:

Enter string to search for:[ ]

You can enter a character string to search for in the file. When the string is found, it is positioned in the center of the DISPLAY

WINDOW. A message also tells you the line number in which the string was found.

③ PROBE displays a linenumber along with the text in this area. PROBE counts carriage return characters to determine the line numbers in the file. Note that this is the same linenumber which is produced by the compiler for use by the Source Step command.

EXAMPLES OF USING THE VIEW COMMANDS

Display all files in the current pathname. The key sequence is:

V*.*

Displays all files in drive a:

VA:*.*

Display all files in the current pathname with a .HEX extension:

V*.HEX

View the file with the pathname a:\srcfiles\main.c:

Va:\srcfiles\main.c

Change the default drive and directory to a:\tempfiles:

V<Tab>a<enter>\tempfiles<enter>

View the file named FTOCIO.C. Display line #103 in this file.

VFTOCIO.C<enter><Tab>103<enter>

WINDOW COMMAND

You can create your own custom data displays called Watch Windows which can be popped up over the DISPLAY WINDOW. These Watch Windows of data are defined by the Window command. The Window command assigns the Watch Window to any *AltKey* (i.e. hold down ALT while you type any other key). The window can be popped up at any time by typing the AltKey. It can be popped down by typing the same *AltKey* again. If more than one window is popped up at a time, they "stack" one under the other. If a command is in process under the pop up window, it is temporarily suspended until all the Watch Windows are popped down. An exception is the Step command. In this case the program can continue to be single stepped even with the Watch Window popped up. During each single step, the Watch Windows are updated. Another exception is the Go command. If a Watch Window is popped up during a Go command, emulation is periodically stopped and the Watch Window is updated.

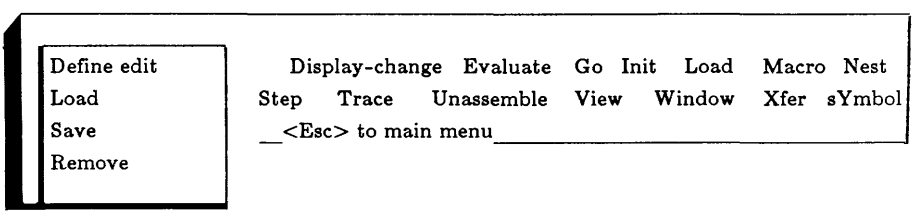
If a Window is being popped up during the definition of a Macro, the *AltKey* which pops the window down is not passed on to the macro. This has the effect of pausing the macro when the window is displayed. You then pop down the window with the appropriate *AltKey* and the macro continues execution.

The *AltKey* which popped up the Watch Window is shown in the window. When multiple windows are popped up, the must be popped down starting with the one on the bottom of the screen.

The Window command is invoked from the MENU BAR by typing:

W for Window

The subcommands for the Window command are displayed in the MENU BOX and the screen looks like this:



The subcommands for the Window command are:

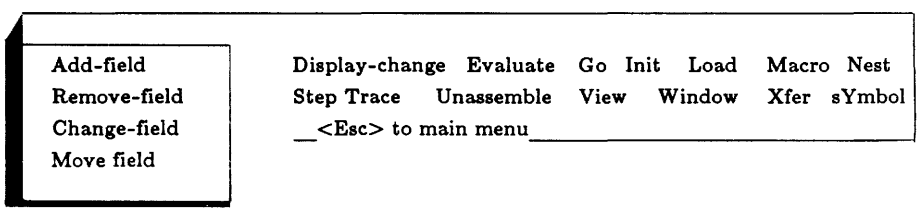
Subcommand	Operation
Define-edit	Define or edit a pop up Watch Window
Load	Load a file of predefined windows
Save	Save current windows definitions to disk
Remove	Delete currently defined window

DEFINING AND EDITING WINDOWS

When the Define subcommand is selected, the following prompt appears:

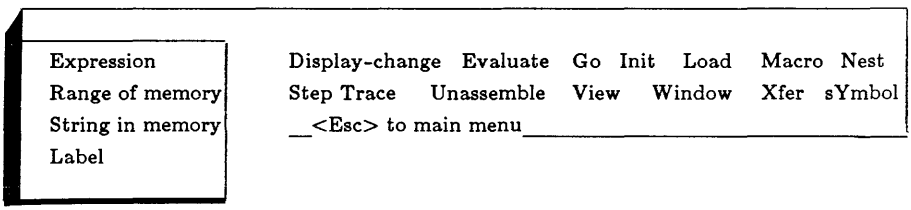
Enter <AltKey> that will activate this window:[ ]

An AltKey means hold down Alt and type any other key. This *AltKey* becomes the key which will pop up the window. This *AltKey* may have already been assigned to a window. In this case, the subcommands let you edit the window definition. If you enter the wild card key *, the names of all windows are displayed. Once entered, the Define Edit subcommand displays another MENU BOX.



ADDING A FIELD

A field is an area of the window which displays the data. When the Add-Field subcommand is selected, another MENU BOX appears.



The size of the field in the window is automatically adjusted to fit the data put into the field. For each of the subcommands in the previous screen, you are prompted for the DATA TYPE and then given the ADDITIONAL DIALOG BOXES shown below:

SUB COMMAND	DATA TYPE	ADDITIONAL DIALOG BOXES
Expression	Byte-hex	*Expression:[ ]*
	Word-hex	"
	Logn-hex	"
	Decimal	"
	Signed-decimal	"
	ASCII	"
	sYmbol	"
	Float-register	"
Range-of-memory	Byte	*Enter start address:*[ ] *Enter end address[ ]*
	Word	"
	Long	"
	Single	
	Double	
	[illegible]	
	Packed	
String-in-memory	Zero-terminated	*Enter address of string:*
	Length-defined	*Enter address of string:* *Enter length of string:*
Label		*Enter label characters:*[ ]

Each field in a Watch Window has a DATA TYPE. These are shown in the previous table and they are explained below. A field which contains an Expression will first evaluate the expression then put the data into the field in the DATA TYPE you select for the expression.

DATA TYPE	DESCRIPTION
Byte	Byte value in hex
Word	Word (two bytes) value in hex
Long	Long (four words) value in hex
Decimal	Long value in decimal no sign extension
Signed-decimal	Long value in decimal sign extended
ASCII	Byte value in ascii
sYmbol	Symbolname which matches expression
Float-register	Floating point value or float register
Range-of-memory	Range of memory in hex
Single	Single precision floating point
Double	Double precision floating point
Xtend	Extended floating point
Packed	Packed decimal
String-in-memory	Ascii string 0 terminated or length defined

At the end of choosing the DATA TYPE for the field and filling in the answers to the PROBE prompts, the final DIALOG BOX for the Add Field subcommand appears:

Place ■ in location for field and press <Enter>

Use the cursor keys to move the solid block cursor shown in the DISPLAY WINDOW to the position within the window where the field is to be displayed. If other fields are already defined for this window, they are shown so that the new field will not conflict with the current fields. You should insure that a field is not positioned such that it writes over another currently defined field. If you do, the data from one field will overwrite the data of other fields.

REMOVING A FIELD

If the Remove-a-field subcommand to the previous screen is selected, the following DIALOG BOX appears:

Place ■ in field to be removed and press <Enter>

The solid block cursor can be moved with the cursor motion keys to the start of the field you want to delete for this window. Deleting this field leaves the remaining fields in the window in place.

CHANGE A FIELD

If the Change-a-Field subcommand is selected, the following DIALOG BOX appears:

Place ■in field to be changed and press <Enter>

The solid block cursor can be moved with the cursor motion keys to the field you want to change in this window. The solid block moves only to the starting position of each field with the cursor keys. Typing <enter> brings up the prompts from the Add-a-field subcommand so that you can edit and make changes to the field. Only the DATA TYPE can be changed for the field.

MOVE A FIELD

If the Move Field subcommand is selected, the following prompt appears:

Place ■ in field to be moved and press <Enter>

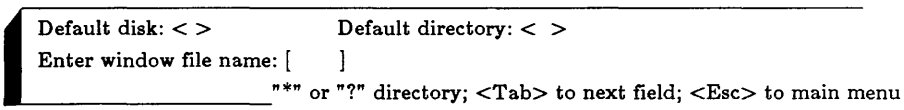
The solid block cursor can be moved with the cursor motion keys to the field you want to move within this window. Typing <enter> then gives you this prompt:

Place ■ in location for field and press <Enter>

Move the solid block cursor where you want the field to move then type <enter>.

LOADING AND SAVING WINDOWS

Currently defined windows can be saved to a disk file with the Save subcommand. These window can be loaded again with the Load subcommand. For either subcommand, the following screen appears:



(1) The DIALOG BOX prompts you for the filename to load or save the windows.

Enter window file name:[ ]

If you do not specify the drive and pathname for the file, the defaults will be assumed.

(2) This is the default disk drive and directory. By typing <TAB>, you can invoke prompts to change these defaults.

Note that the currently defined windows can also be saved with the Initialize Save command and loaded with the Initialize Load command. If Windows are currently already defined or loaded in PROBE, then Windows from new loads will not redefine the currently defined windows. If you want to replace the current definitions with new, remove the specific Watch Window names.

REMOVING A WINDOW

A window can be deleted by selecting the Remove-a-window subcommand. The following prompt appears:

Enter <AltKey> that will activate this window: [ ]

If you cannot remember the *AltKeys* for the windows, type * and all currently active window and macro *Altkeys* will be displayed. Entering A for this prompt will delete all currently active windows.

OTHER NOTES ON WINDOWS

If you try to define a window which is already assigned to a Macro, the following prompt appears:

<AltKey> is a macro. Remove it:[Yes] {Yes|No}

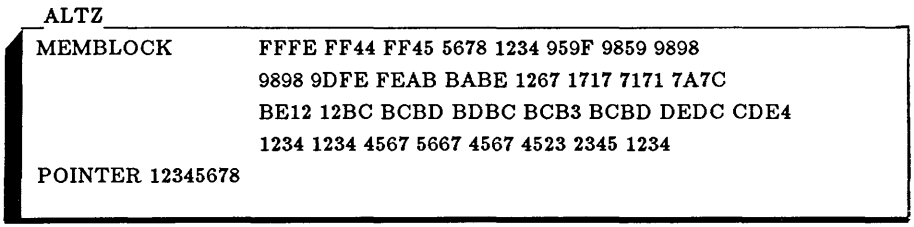
You can then elect to delete the macro in favor of the window.

You can also edit Windows off line with your favorite text editor. See Appendix D for doing this.

EXAMPLES OF USING THE WINDOW COMMANDS

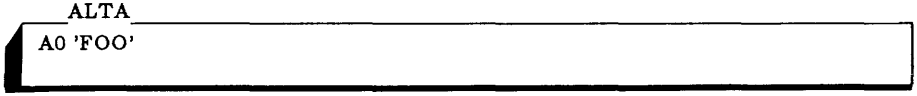
Define a window named AltZ which display the block of memory 3F words long which starts at location \ARRAY. Put a label on this array called MEMBLOCK. Then print the Long word which is pointed to by the variable at \VARPOINTER. Put a label on this called POINTER. A sample of this window is shown below.

WD<AltZ><enter>ALMEMBLOCK<enter><enter>ARW\ARRAY
<enter>+3F<enter><RtArrow><RtArrow><RtArrow><RtArrow>
<RtArrow><RtArrow><RtArrow><RtArrow><RtArrow><enter>
ALPOINTER<enter><DnArrow><DnArrow><DnArrow><DnArrow>
<DnArrow><DnArrow><LtArrow><enter>ARL[\VARPOINTER]
<enter>+1<enter><RtArrow><RtArrow><RtArrow><RtArrow>
<RtArrow><RtArrow><RtArrow><RtArrow><enter><Esc>



Define a window named *AltD* which displays the contents of register A0 in ASCII. Put a label A0 in front of the data. (Note that a window called *AltR* which displays the contents of all registers in a long word format is included in file of predefined windows included with the PROBE distribution diskette). Assume the contents of register A0 is an ascii 'FOO' for this example.

WD<AltD><enter>ALA0=<enter><enter>AEAA0<enter><RtArrow>
<RtArrow><RtArrow><enter><Esc>



Define a Window named Alt A which displays the value of a long pointer represented by the symbol POINTER and label it POINTER =. Then display the 0 terminated string pointed to by this long pointer.

WD<ALTA><enter>ALPOINTER=<enter><enter>ASZ[POINTER].L
<rtarrow><rtarrow><rtarrow><rtarrow><rtarrow><rtarrow>
<rtarrow><rtarrow><enter>

Suppose you program calculates target address to which it will jump. Suppose the target address which will be reached have symbols associated with the addresses. Assume the target address is the contents of the A6 register. Define a window which will show which symbol matches the contents of A6. Name the window AltX

WD<ALTX><enter>AEYA6<enter><esc>

Load the file of windows from the file REG.WIN.

WL REG.WIN<enter>

Save all currently defined windows in a file called WINSAVE.WIN. First change the current directory to \WINDOWS and the current drive to B.

WS<Tab>B<enter>\WINDOWS<enter>WINSAVE.WIN

Remove the window assigned to AltD.

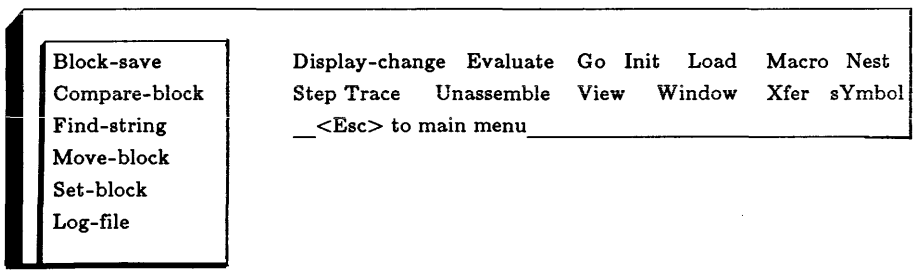
WR<AltD><enter>

XFER COMMAND

The Xfer command lets you save, compare, find, move, and initialize blocks of memory in the target system. It also lets you redirect IO. This command is invoked from the MENU BAR by typing:

X for Xfer

The subcommands for the Xfer command are displayed in the MENU BOX and the screen looks like this:

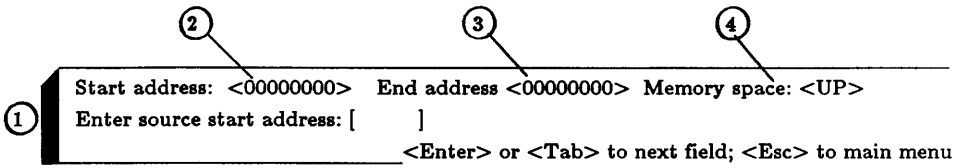


The Xfer command has the following subcommands:

Subcommand	Operation
Block	Save a range of memory to disk file
Compare	Compare two blocks of and report differences
Find	Search memory for a string
Move	Move block of memory to new location
Set	Fill block of memory with a string
Log-file	Enable or disable redirection of PROBE output to a log file.

SAVING A BLOCK OF MEMORY

The Block-save subcommand saves a block of target memory to a disk file in S record format. The memory is read from the target as bytes. When invoked, the following screen pops up:



① The first DIALOG BOX prompts you for the starting address of the block:

Enter source start address: [ ]

You may enter any type of address expression or simply type <enter> to accept the default start address shown on the screen.

Once entered, you then prompted for the ending address:

Enter source end address: [ ]

You may enter any type of address expression. The end address may also be of the form:

+ number

In this case the end address becomes start address+number. If you simply type <enter> without entering a new end address, you get the default end address shown in < >.

② This is the default start address.

③ This is the default end address.

④ The default memory space is shown on the screen. You may change it by typing <TAB> to get the following prompt:

Memory space: <UP> {0|UD|UP|UR|4|SD|SP|CPU}

Once the end address is entered the following screen appears:

```
Default disk: <C>                    Default directory: <   >
Enter save range filename: <     >
                    "*" or "?" directory: <Tab> to next field; <Esc> to main menu
```

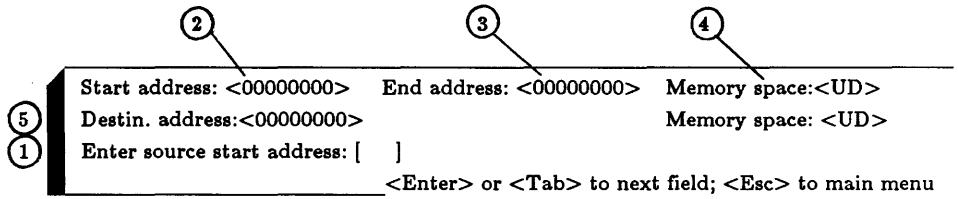
You are prompted for the file to save the block of memory:

Enter save range filename:[ ]

If you do not enter the drive and pathname for the filename, the defaults on this screen will be used. Type <TAB> to bring up the prompts which will let you change these defaults.

COMPARING BLOCKS OF MEMORY

A block of memory called the source block can be compared to another block called the destination block with the Compare subcommand. The memory is read from the target as bytes. When invoked the following screen appears:



(1) The DIALOG BOX prompts you for the starting address of the source block:

Enter source start address: [ ]

You may enter any type of address expression or simply type <enter> to accept the default start address shown on the screen. Once entered, you then prompted for the ending address:

Enter source end address: [ ]

You may enter any type of address expression. The end address may also be of the form:

+ number

In this case the end address becomes start address+number. If you simply type <enter> without entering a new end address, you get the default end address for the source block.

(2) This is the default start address for the source block.

(3) This is the default end address for the source block.

(4) This is the default memory space for the source block. You may change it by typing <TAB> to get the following prompt:

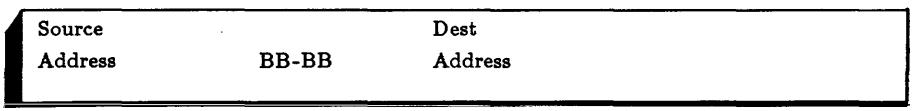
Memory space: <UP> {0|UD|UP|UR|4|SD|SP|CPU}

(5) Once the end address for the source block is entered you are prompted for the address of the destination block:

Enter destination address:[ ]

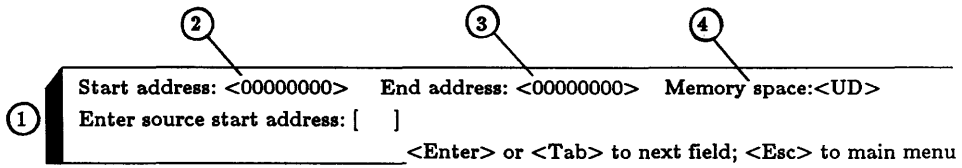
The default destination address shown on the screen can be chosen by simply typing <enter> in response to this prompt. The default memory space for the destination block is shown on the screen and can now be changed by typing <TAB> to get the appropriate prompt.

Once the destination address has been entered, PROBE compares the source block of memory to the block starting at the destination address byte by byte. Miscompares are displayed in the DISPLAY WINDOW in the following format:



FINDING A STRING IN MEMORY

A block of memory can be searched for a string by choosing the Find-string subcommand. The following screen appears:



① You are first prompted for the starting address of the block:

Enter source start address: [ ]

You may enter any type of address expression or simply type <enter> to accept the default start address shown on the screen. Once entered, you then prompted for the ending address:

Enter source end address: [ ]

You may enter any type of address expression. The end address may also be of the form:

+ number

In this case the end address becomes start address+number. If you simply type <enter> without entering a new end address, you get the default end address for the block.

② This is the default start address

③ This is the default end address

④ This is the default memory space. You may change it by typing <TAB> to get the following prompt:

Memory space: <UP> {0|UD|UP|UR|4|SD|SP|CPU}

Once the end address for the block is entered you are prompted for the string to search for

Enter list:[ ]

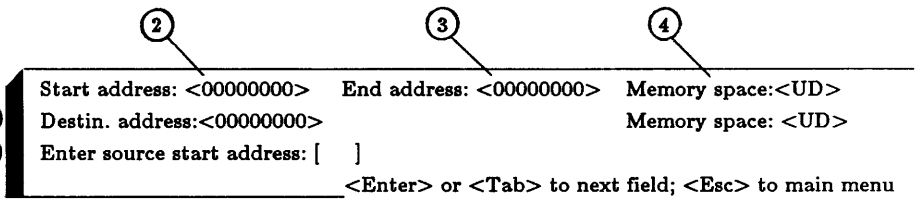
The list may be hex bytes, or an ASCII string in quotes. The memory is read from the target as bytes. Once entered, the following prompt appears:

Report successful matches(yes) or unsuccessful matches(no):[yes]

If you answer yes to this prompt, then each time the string compares successfully to the block of memory, the starting address of the string match is printed in the DISPLAY WINDOW. If you answer no to this prompt, then each time the string does not compare successfully to the block of memory, the starting address of each miscompare is printed in the DISPLAY WINDOW.

MOVING A BLOCK OF MEMORY

A block of memory called the source block can be moved to a new memory location called the destination address the Move subcommand. When invoked the following screen appears:



(1) The DIALOG BOX prompts you for the starting address of the source block:

Enter source start address: [ ]

You may enter any type of address expression or simply type <enter> to accept the default start address shown on the screen. Once entered, you then prompted for the ending address:

Enter source end address: [ ]

You may enter any type of address expression. The end address may also be of the form:

+ number

In this case the end address becomes start address+number. If you simply type <enter> without entering a new end address, you get the default end address for the source block.

(2) This is the default start address for the source block.

(3) This is the default end address for the source block.

(4) This is the default memory space for the source block. You may change it by typing <TAB> to get the following prompt:

Memory space: <UP> {0|UD|UP|UR|4|SD|SP|CPU}

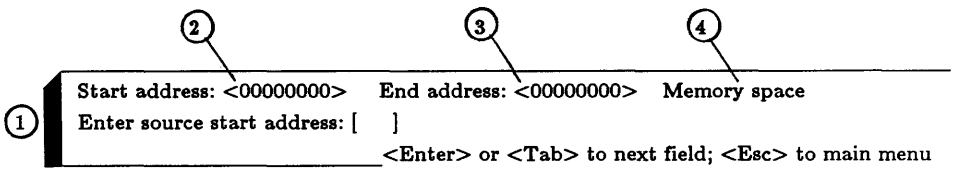
(5) Once the end address for the source block is entered you are prompted for the destination address :

Enter destination address:[ ]

The default destination address shown on the screen can be chosen by simply typing <enter> in response to this prompt. The default memory space for the destination address is shown on the screen and can now be changed by typing <TAB> to get the appropriate prompt. Once the destination address has been entered, PROBE moves the source block of memory to the the destination address byte by byte. The memory is read from the target as bytes. If the start of the destination overlaps the source range, the data is moved starting with the high address end of the source block.

INITIALIZING A BLOCK OF MEMORY

A block of memory can be initialized to a string by choosing the Set-block subcommand. The following screen appears:



(1) The DIALOG BOX prompts you for the starting address :

Enter source start address: [ ]

You may enter any type of address expression or simply type <enter> to accept the default start address shown on the screen. Once entered, you then prompted for the ending address:

Enter source end address: [ ]

The ending address is an address expression. It may also be:

+ number

In this case the end address becomes start address+number. If you simply type <enter> , you get the default end address.

(2) This is the default start address

(3) This is the default end address

(4) This is the default memory space. Type <TAB to change it:

Memory space: <UP> {0|UD|UP|UR|4|SD|SP|CPU}

Next you are prompted for the string to search for.

Enter list:[ ]

The list may be hex bytes, or an ASCII string in quotes. Once the list has been entered, PROBE fills the memory repeatedly with the list. The final copy of the list written into memory may only be a partial since the block size may not be an even multiple of the list size. The list is treated as a sequence of bytes.

REDIRECTING PROBE OUTPUT TO A LOG FILE

You can redirect PROBE output to a log file. This lets you save the history of a debugging session. A log file can be an AT disk file, printer, communications port on the AT, or file on a remote file server on an AT network. PROBE simply calls DOS with the filename handle. DOS then redirects the output to the specified device. When this subcommand is selected, the following prompt appears:

Enable listing to log file [No] {Yes|No}

If you select No, any current open log file is closed. If you select Yes, the following screen appears:

```
Default disk: <C>  Default directory: < >
Enter new log file name: [  ]
```

You are prompted for the name of the log file. The default disk drive and directory are shown on this screen. Type <<TAB>> to bring up the prompts which let you change these defaults. Here are the devices you can specify for a log file:

Filename	Description
filename	DOS filename either local or remote on a network
lpt1:	Lineprinter 1 attached to the AT
lpt2:	Lineprinter 2 attached to the AT
com1:	Com1 port on At
com2:	Com2 port on AT

Insure that the line printer is attached and turned on, however, otherwise PROBE will wait on the print device indefinitely.

EXAMPLES OF USING THE XFER COMMANDS

Save a block of memory to disk from \ARRAYSTART to \ARRAYEND in a file called SAVEBLOCK. Use the UD memory space. Use drive D with directory \TEMPSAVE.

XB\ARRAYSTART<enter>\ARRAYEND<Tab>UD<enter>
SAVEBLOCK<Tab>D<enter>\TEMPSAVE<enter><enter>.

Compare a block of memory starting at the defaults shown on the screen with the block at \ARRAYSTART in the default destination memory space.

XC<enter><enter>\ARRAYSTART<enter>

Find the string in memory "Now is the time" in the user data space starting at 30000000 and ending at 3FFFFFFF. Report only successful matches.

XF30000000<enter>3FFFFFFF<enter>"Now is the time"<enter>Y

Move the block of memory from \ARRAYSTART to \ARRAYSTART+3FF into \NEWARRAY. Use the default memory spaces.

XM\ARRAYSTART<enter>+3FF<enter>\NEWARRAY<enter>

Fill the block of memory starting at \ARRAYSTART and ending at \ARRAYEND with the string "Now is the time".

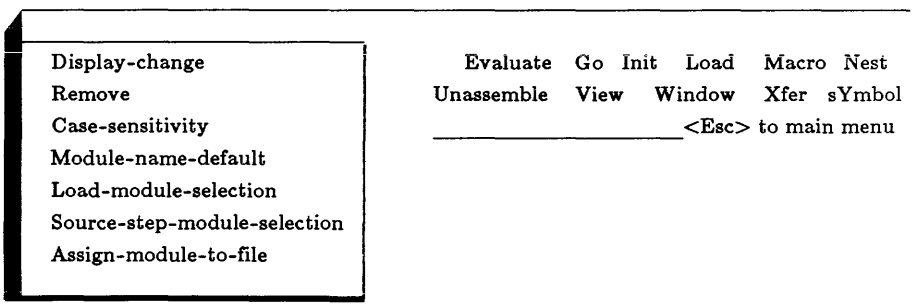
XS\ARRAYSTART<enter>ARRAYEND<enter>"Now is the time"<enter>

SYMBOL COMMANDS

PROBE loads symbols into the AT memory when the program is loaded into the target ram or MAP RAM with the Load command. PROBE is compatible with a number of object module formats. See the Appendix G under Compatible Object module formats for more details. The sYmbol command lets you do many things with the PROBE symbols. It is invoked from the MENU BAR by typing

Y for sYmbol

The following MENU BOX is displayed:

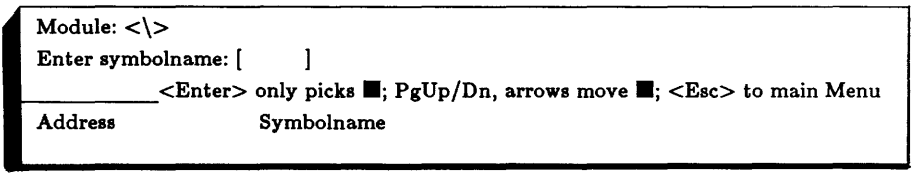


The subcommands for the symbol command are shown here along with a short description of their operation.

Subcommand	Operation
Display-change	Display, change or delete symbols
Remove	Delete symbols or all symbols in module
Case-sensitivity	Ignore or use case in symbols
Module-name-default	Assign default symbol prefix
Load-module-selection	Load symbols from only from selected modules
Source-step-module-selection	Source step only in selected modules
Assign-module-to-file	Display/change filenames assigned to modulenames

DISPLAY AND CHANGE SYMBOLS

Symbols are displayed by selecting the Display subcommand. When invoked, the following screen appears:



(1) PROBE groups symbols by modulename. The symbols for the default Module on the screen are displayed in the DISPLAY WINDOW. The default modulename after the program has been loaded is \ which means no modulename (i.e. public symbols). The value of the symbol is shown with the symbol. The first symbolname in the DISPLAY WINDOW is also highlighted. If there is more than a screenful of symbols, you may use the PgUp/PgDn keys to scroll through the symbols for this module. The DIALOG BOX prompts you for the symbolname to be displayed:

Enter symbolname: [ ]

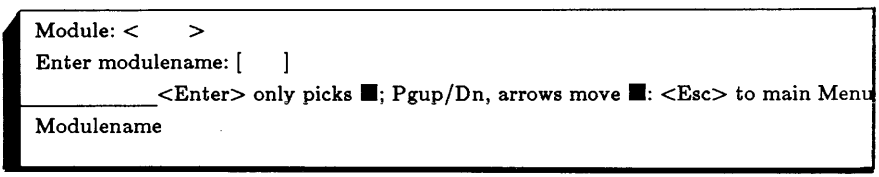
Entering a symbolname for this prompt searches the symbol table for the symbolname and displays the symbol along with its value. This lets you find a symbol in a very large symbol table. If a symbolname is entered without specifying a modulename, the current default modulename shown on the screen is assumed. Typing <Enter> without a symbolname in response to this prompt will choose the symbolname which is currently highlighted in the DISPLAY WINDOW. Entering a new currently undefined symbolname will insert the symbolname before the highlighted symbol.

After the symbolname is chosen, the DIALOG BOX lets you change or define the value of a symbol:

Enter new symbol value:[ ]

After entering a value or typing <Enter> without entering a value, you are returned to the previous prompt so you can continue to display or define new symbols.

② This is the current default modulename. To change the default modulename type <TAB> and the following screen will appear:



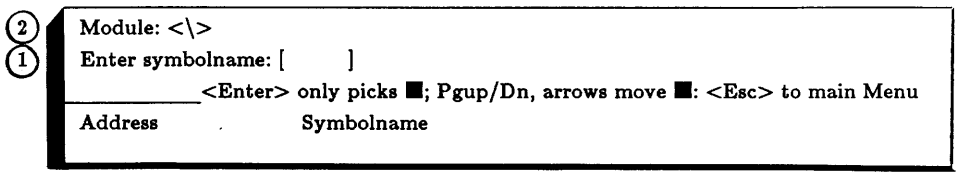
All modulenames are displayed in the DISPLAY WINDOW and one is highlighted. The following prompt appears:

Enter modulename[ ]

You can enter a new modulename by typing it in or moving the highlight field to the selected modulename and typing <enter>. The default can also be \ which means no modulename. Public symbols do not have a modulename and therefore their default modulename is \. After the default modulename is entered, you are returned to the prompt in item 1 above.

DELETING SYMBOLS AND MODULES

Symbols are deleted by selecting the Remove subcommand. When invoked, the following screen appears:



(1) You are first prompted for the symbolname for the symbols to be deleted:

Enter symbolname: [ ]

Typing <Enter> without a symbolname in response to this prompt will choose the symbolname which is currently highlighted in the DISPLAY WINDOW. Entering * as the symbolname will invoke the following prompt:

Remove all symbols in this module: [No] {Yes|No}

Choosing Y will delete all symbols for this module. This also removes the modulename from future displays of symbols.

(2) This is the current default modulename. To change the default modulename type <TAB> and the following prompt:

Enter modulename[ ]

You can enter a new modulename by typing it in or moving the highlight field to the selected modulename and typing enter. If you select *, then the following prompt appears:

Remove all symbols:[No] {Yes|No}

This will delete all symbols for all modules as well as all modulenames.

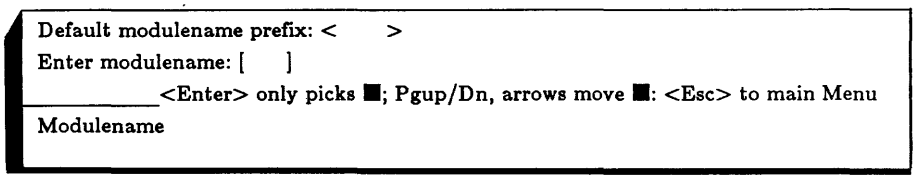
CASE SENSITIVITY

Some compilers create different symbolnames if the case for any of the characters of the symbol are different. Other compilers ignore case. You can choose to use or ignore case with the Case-sensitivity subcommand. When this subcommand is invoked, the following prompt appears:

Upper case characters equal to lower case: [Yes] {Yes|No}

SELECTING THE DEFAULT MODULENAME

It is tedious to type in the complete modulename and symbolname prefix for every symbol if you are always working within the same module. PROBE lets you define a default modulename and prefix which will automatically be included in front of symbols. This is done with the Module-default subcommand which displays the following screen:



All currently loaded modules are displayed on the screen and the following prompt appears:

Enter modulename: [ ]

One of the modulenames on the screen is highlighted. The cursor keys move the highlighted field. You can simply type <enter> to select the highlighted modulename or type in a new modulename. Once entered, the following prompt appears:

Enter rest of modulename prefix:[ ]

This lets you add an additional symbolname prefix to be used as the default prefix for all symbols. This is very useful for variables which have scope. You can also simply type <enter> without additional prefix information.

If no modulename is specified when you specify symbol or linenumber, then the current default prefix is used. After the symbol table is loaded, the default prefix is public symbols (\). You can define a new default prefix with the sYmbol-Modulename-default command.

HOW THE DEFAULT MODULENAME WORKS

The default modulename is inserted automatically by PROBE in front of all symbols in an expression when PROBE interprets the expression (the insert is not visual). A default prefix can be defined to include the modulename and any other scope of prefix which you want to include as a default in front of your symbols. If you include a prefix when you use a symbol, and the default prefix overlaps the prefix you have put in front of your symbol, the overlapping parts are ignored. PROBE iteratively scans the symbol and tries to match it to entries in the symboltable. PROBE uses the following algorithm:

> If the specified symbol starts with \ look only in public symbols
> First look for \defaultmodulename\Name
> Next, look for \defaultmodulename-1element\Name
> Iterate this until you reach \Name

EXAMPLE: Assume the following symbols are in the symbol table:

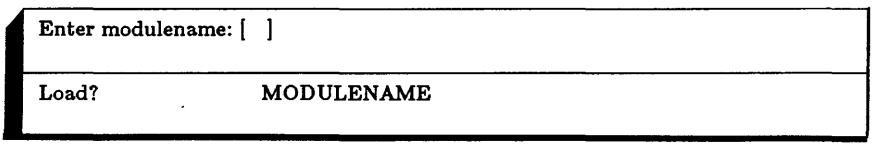
\Sym1
\Sym2
\Mod1\Sym1
\Mod1\Sym3
\Mod1\Block1\Sym3
\Mod1\Block2\Sym2
\Mod1\Block2\Sym4

Assume the default modulename is \Mod1\Block2. PROBE would then search this symbol table as follows when you enter the symbolnames shown:

SYMBOL	PROBE SEARCHES FOR
Sym1	\Mod1\Block2\Sym1
	\Mod1\Sym1
Sym3	\Mod1\Block2\Sym3
	\Mod1\Sym3
Block1\Sym3	\Mod1\Block2\Block1\Sym 3
	\Mod1\Block1\Sym3
Sym4	\Mod1\Block2\Sym4
\Block2\Sym4	\Block2\Sym4 - not found
\Sym1	\Sym1

SELECTIVELY LOADING SYMBOLS

The loading of symbols into the symbol table can be limited to specified modulenames. This is done with the Load-Module-selections subcommand. When invoked the following screen appears:



The current modules are shown in the DISPLAY WINDOW. These may have been previously loaded from the PROBE initialization file. If the PROBE Load command is used before this command is used, then the default assumed is that all modules and all symbols will be loaded into the PROBE symbol table and all modulenames are loaded into this selection list table. Once this subcommand is used, only symbols which appear in the selected modules will be loaded in future loads. The selected modules could also have been specified with the Init Load command. The following prompt appears:

Enter modulename: [ ]

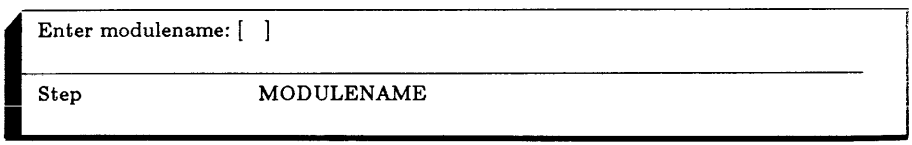
You can now enter new modulenames into this list or select yes or no for the state of the loading of symbols for a module. One of the modulenames in the DISPLAY WINDOW is highlighted. Entering a modulename selects this module or enters a new modulename in the list if it is not currently defined. Or, the highlighted modulename is selected by typing <enter>. Once selected, the next prompt enables or disables the loading of symbols into the symbol table when the Load command loads the program into the target system.

Allow symbols to be loaded for this module:[Yes] {Yes|No}

You remain in this command to change other modules until you type <Esc>. Note that you can delete modulenames from the selection list by using the sYmbol Remove * command to delete all the symbols in a module.

LIMITING SOURCE LEVEL SINGLE STEPPING TO SPECIFIED MODULES

SOURCE can single step a high level language program by statements with the Step Source command. It may be desirable to limit the single stepping of source code to only specified modules. When the program goes outside of the specified modules, it runs real time until it gets back into the selected modules. This can be done with the Source-step-module-selection subcommand. When invoked the following screen appears:



The current modules are shown in the DISPLAY WINDOW. These may have been previously loaded with the Init Load command. If the PROBE Load command is used before this command is used, then the default assumed is that all modules will be single stepped with the Step Source command. Once this subcommand is used or the selected modules have been loaded with the Init Load command, only symbols which appear in this list will be source stepped. The following prompt appears:

Enter modulename: [ ]

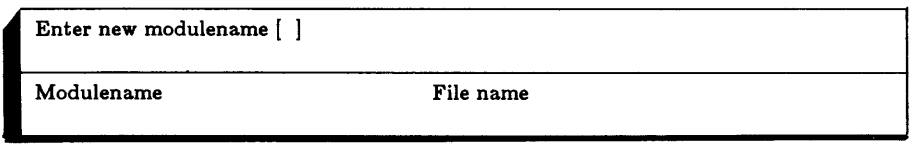
You can now enter new modulenames into this list or change the single stepping for a module. One of the modulenames in the DISPLAY WINDOW is highlighted. Entering a modulename selects this module or enters a new modulename in the list if it is not currently defined. Or, the highlighted modulename is selected by typing <enter>. Once selected, the next prompt enables or disables the source single stepping for this module.

Source-step at lines in this module:[Yes] {Yes|No}

See the Step command in this chapter for how the Source Step command works.

ASSIGNING MODULENAMES TO SOURCE FILES

For source level single stepping or for including source code in Trace and Unassemble displays, source files must be assigned to modulenames . To do this select the Assignments subcommand and the following screen is displayed.



The current modules are shown in the DISPLAY WINDOW. These may have been previously loaded with the Init Load command. If the PROBE Load command is used before this command is used, and the object module format includes the assignments then, this table receives default initializations. The following prompt appears:

Enter modulename: [ ]

You can now enter new modulenames into this list or change the source file assignment for a module. One of the modulenames in the DISPLAY WINDOW is highlighted. Entering a modulename selects this module or enters a new modulename in the list if it is not currently defined. Or, the highlighted modulename is selected by typing <enter>. Once selected, the next prompt lets you assign the source filename

Enter new file name: [ ]

You remain in this command to change other modules until you type <Esc>.

EXAMPLES OF USING SYMBOLS COMMANDS

Display all symbols for the module \FTOCIO

YD<Tab>FTOCIO<enter>

Starting from the where the previous example leaves you on the screen, display the symbol Getval and change its address to Getval+1.

GETVAL<enter>GETVAL+1<enter>

Starting from the where the previous example leaves you on the screen, go down three symbols and change that symbols value to 0.

<DnArrow><DnArrow><DnArrow><enter>0<enter>

Delete all symbols in module \FTOCIO and delete the modulename.

YR<Tab>\FTOCIO<enter>*Y

Delete all symbols in all modules.

YR<Tab>*<enter>Y

Set the Case sensitivity to treat upper and lower case identically.

YCY

Change the default modulename to FTOCIO.

YMFTOCIO<enter><enter>

Set up PROBE so that symbols from modulename \FTOCIO will not be loaded when the program is loaded.

YLFTOCIO<enter>N

Unselect a module named FTOCIO from the list of modules which can be source level single stepped (i.e. the code will run real time through this module during source single step.)

YSFTOCIO<enter>N

Assign a module named FTOCIO to a file called a:\sourcefiles\FTOCIO.C for the purpose of single stepping.

YAFTOCIO<enter>A:\SOURCEFILES\FTOCIO.C

APPENDICES

APPENDIX A
PROBE ERROR MESSAGES

These are the error messages which PROBE will display. To clear the error message and resume keyboard input to PROBE, strike any key. The error messages are arranged in this Appendix as nearly as possible to alphabetical for easy reference.

"Access denied to file."
The specified file could not be accessed because of a privilege violation.

"Array has no memory. It cannot be used in map."
The array selected to be mapped to user memory does not contain any memory (see array size at the top of the screen) This array is not available for mapping to the user system.

"Array may not cross 16Mbyte boundary."
Mapped memory arrays are not allowed to cross a 16Mbyte boundary. i.e. The start address "xxvw0000" and the end address "xxyzFFFF" of the block must have the same "xx" value.

"Attempt to read past end-of-file."
An invalid read request was made to the file.

"Attempted access to guarded memory at address xxxxxxxx"
The memory access at address xxxxxxxx is flagged as guarded in the Display Map command.

"Attempted division by 0."
Division by 0 was attempted in the expression.

"bbbbb: base address should be "xy000": xy=C4,CC,D0,D4,D8,DC"
The base address listed in the PROBE.CNF file ("bbbbb") is not a valid board base address.

"Bad drive request: Abort, Retry, Ignore?"
See "DOS critical error"

"Baud rate must be 110, 150, 300, 600, 1200, 2400, 4800, or 9600"

For Probe/1, the com port used to communicate with the target system may only be set to the above baud rates. The com port is always set to 8 data bits, 2 stop bits, no parity.

"Block will not fit in array. NOTE: Array may not cross 16Mbyte boundary."

This block will not fit in the specified array. This may be caused by:

1) Not enough RAM in this array to cover the region.

2) The array is used in another region or regions and cannot fully cover those regions as well as the selected region.

3) The array would have to cross a 16Mbyte boundary (xx000000) to cover the other regions as well as the selected region.

"Board does not respond at address xy000"

The base address is valid but the board does not respond at that address. This may mean:

1) The Atron Break/Trace boards are not installed in the computer.

2) The Atron Break/Trace boards are installed in the computer but this address conflicts with another board in the computer. try changing the base address in the PROBE.CNF file.

3) The Atron Break/Trace boards have failed. Run the diagnostics.

3) The Atron Break/Trace boards were ordered with the special modification to place their selection in I/O space. If this is the case, make sure the ", pppp" option is also added to the ADDR option in the PROBE.CNF file.

"BP x " error in sticky BP

(see BP command)

"Bus time out exception caused by access at address xxxxxxxx"

The memory access at address xxxxxxxx timed out. The bus error may be caused by the target system if it is capable of causing a bus error time out. It also may have been caused by the PROBE software when it was detected that the memory access was taking too long.

"Cannot communicate with floating point co-processor #n"
The CPID option in the PROBE.CNF file specified this co-processor as floating point but communication with that co-processor could not be established in the target system.

"Cannot communicate with PMMU co-processor #n"
The CPID option in the PROBE.CNF file specified this co-processor as a PMMU but communication with that co-processor could not be established in the target system.

"Chaining register must be A0-A6"
The only valid registers for use as a stack chain with the nest command are the registers A0, A1, A2, A3, A4, A5, and A6.

"Com port must be 1 or 2"
For Probe/1, the com port used to communicate with the target system must be 1 or 2 for COM1: or COM2:

"Could not communicate with master 8031. Use Hw Reset command"
Communications could not be established with the 8031 on the Atron break/trace boards in the computer. Try the HW Reset command. If this occurs again, run the diagnostics.

"Could not communicate with slave 8031 in pod. Use Hw Reset command"
Communications were established with the 8031 on the Atron break/trace boards in the computer. However, the computer could not communicate with the 8031 in the Atron personality pod. Try the HW Reset command. If this occurs again, it may be caused by:
1) Loose cable connections at the break/trace boards in the back of the computer.
2) Loose cable connections at the personality pod.
3) The personality pod has failed. Run the diagnostics.

"Could not communicate with target processor Use Hw Reset command."

Communications were established with the 8031 on the Atron break/trace boards in the computer and with the 8031 in the Atron personality pod. However, communications could not be established with the target processor. Try the HW Reset command. If this occurs again, it may be caused by:
1) The target system is turned off.
2) The buffer assembly has become disconnected from the target system.
3) The target system is not providing clocks or a processor for the personality pod.
4) The personality pod has failed. Run the diagnostics.
5) A double bus fault occurred while accessing the target system, either while interrogating the target system or while executing code in the target system. In either case, the target processor has entered the SHUTDOWN state and must be reset.

"Could not detect start of target processor execution. Use Hw Reset command"

Attempted to start execution of target processor instructions but could not detect the fact that execution had actually begun. Try the HW Reset command.

"Could not stop execution of target processor. Use Hw Reset command"

The target processor did not respond to the interrupt level 7 produced by the Atron personality pod. The HW Reset command will re-establish communications with the target system but the target processor's internal registers will be lost.

"Could not open heap overflow file on disk."

The heap overflow file "ATRON.HEP" could not be opened on disk in the directory listed in the PROBE.CNF file, in the directory containing the PROBE.CNF file, or in the default directory. The symbol table, macro table, and window table will not be allowed to overflow to disk.

"Could not write instruction words at this address."
An attempt was made to write the instruction at the assemble address but that area of memory was not RAM. This will only occur if Display-change Noverify-state is set to read-after-write.

"Communication failure with pod. Use Hw Reset command"
A general communication failure occurred while communicating with the pod. Try the HW Reset command or the diagnostics.

"Co-processor ID must be: 0 <= ID <= 7"
Co-processors must have an ID field from 0 to 7.

"Co-processor specified is not floating point"
The floating point register name is a valid register name. However,the CPID option in the PROBE.CNF file did not specify this co-processor as a floating point co-processor.

"Co-processor specified is not PMMU"
The MMU register name is a valid register name. However, the CPID option in the PROBE.CNF file did not specify this co-processor as an PMMU co-processor.

"CRC error: Abort, Retry, Ignore?"
See "DOS critical error"

"Destination more than 32K away from source."
DBcc instruction must branch within 32K of assemble address.

"DOS critical error: Abort, Retry, Ignore?"
DOS detected an error while accessing the disk. You may type:"A" or <Esc> to abort the operation,"R" to retry the operation, "I" to ignore the error and continue.

"Drive not ready: Abort, Retry, Ignore?"
See "DOS critical error"

"Duplicate symbol name found but not stored in symbol table."
While loading symbols, duplicate names were found. The new names and their values were discarded and the values in the symbol table were not changed.

"Error accessing heap overflow file on disk. Heap may be corrupted."
The symbol table, macro table, or window table overflowed to the heap overflow file on disk. When an attempt was made to access the overflowed data, a non-recoverable disk error occurred (e.g. the diskette containing the overflow file has been removed.) If the symbol table, macro table, or window table were being loaded, then the symbol table, macro table, and window table may all be corrupted.

"Execution stopped by bus error in user system and stack. Use Hw reset command"
The target processor responded to the interrupt level 7 produced by the Atron personality pod only after the pod caused a bus error. The bus error occurred in the target system, with all of the appropriate information being put on the target stack, before the interrupt was recognized. Therefore, the current PC and SP reflect the system state in the bus error exception handler.

"File not found."
The specified file was not found and could not be opened.

"File system error."
A general DOS file system error was detected.

"First operand specified is illegal."
The first of the operands specified or its addressing mode is not allowed for this instruction.

"Illegal operand specified."
One of the operands or addressing modes specified is not allowed for this instruction.

"Illegal floating point format"
The input floating point number is not specified correctly: [digits] ['.' [digits]] ['E' digits]

"Invalid file access."
An invalid access was made to the file.

"Less operands are required."
There are more operands in the typed instruction than are required for this instruction (e.g. MOVE D0,D1,D2).

"Line number specification must be single decimal number."
You may not specify expressions as line number -- just a value.

"Macro name may not be changed while editing macro."
All information about a macro except its name may be changed.

"Macro nesting > 5. Macro will not execute."
The macro nesting (macros starting other macros) has exceeded 5. No more macros may start until the current macro has finished.

"Map table is full. No new entries may be created."
There are a maximum of 19 map regions.

"More operands are required."
There are more operands required for this instruction than were found in the typed instruction (e.g. MOVE D0).

"Must assign at least one breakpoint to one BP."
When the sequential BP is active, at least one breakpoint must be assigned to one BP.

"Must be: 0:00.000,000 <= Time <= 71:34.967,295"
The maximum time allowed by the hardware timer is just over 71 1/2 minutes.

"Must be: 0 <= BKPT # <= 7 or 0 <= TRAP # <= F"
Breakpoints are number 0 to 7, Traps are numbered 0 to F.

"Must be: 1 <= Shift/rotate factor <= 8"
Shift/rotate immediate instruction must have a value from 1 to 8.

"Must be: 1 <= Pass Count <= 00FF"
The pass counter is a one byte counter and, thus, may have values in the range of 1 to FF.

"Must specify a size."
This instruction with these operands must have a size (".b", ".w", ...).

"Must be: 4 >= wait states >= n"
The maximum number of wait states that may be selected for the map arrays is 4. The minimum is computed as a function of RAM speed on the map array and clock speed to the target processor. The selected number of wait states must be between these values (inclusive).

"Must be: 1 <= Step Count <= FFFF"
The step counter (number of steps to take for each <Enter>) is a 16 bit value.

"No address strobes to target processor Use Hw Reset command."
See "Could not communicate with target processor."

"No clocks to target processor. Use Hw Reset command"
See "Could not communicate with target processor."

"No Vcc to target processor. Use Hw Reset command"
See "Could not communicate with target processor."

"Non-sticky Breakpoint "
error in non-sticky BP (see GO command)

"Non-sticky Breakpoint ".error in non-sticky BP
(see GO command)

"Operator expected but not found in expression."
An operator was expected in the expression between the operands.

"PMMU did not accept new value of register xxx,..."
The PMMU signaled an exception when the new value of the listed register(s) was written to the co-processor. The only registers that may be aborted are: TC, SRP, CRP, and DRP.

"Printer out of paper: Abort, Retry, Ignore?"
See "DOS critical error"

"Read fault: Abort, Retry, Ignore?"
See "DOS critical error"

"Region must have StartAddress <= EndAddress"
Map regions must have a start address that is less than their end address.

"Register name specified is not a floating point register"
The register name specified in the expression is not an 8, 16, or 32 bit floating point register.

"Register name specified is not a PMMU register"
The register name specified in the expression is not an 8, 16, or 32 bit MMU register.

"Register name specified is not a floating point register"
The register name specified in the expression is not a valid floating point register for display in the Float-register field of the window.

"Second operand specified is illegal."
The second of the operands specified or its addressing mode is not allowed for this instruction.

"Sector not found: Abort, Retry, Ignore?"
See "DOS critical error"

"Seek error: Abort, Retry, Ignore?"
See "DOS critical error"

"Specified size not allowed."
The specified instruction size (".b", ".w", ".l", ...) is not allowed for this instruction with these operands.

"Specified values not found from current location to the end of trace."
The specified search values were not found starting at the end of the screen and searching to the end of valid trace memory. Try <Ctrl><PgUp> then executing the search again.

"Symbol not found: xxxxxxxx"
The specified symbol ("xxxxxxx") was not found in the symbol table and cannot be displayed on the screen.

"String not found in file."
The string to be searched for was not found in the file. Note that upper- and lower-case characters are identical in this search.

"Symbol/Macro/Window allocation table is full."
The symbol table, macro table, and window table in memory and on disk is full of information. All symbols from the load file may not have been loaded.

"Symbol/Macro/Window allocation table is full."
The symbol table, macro table, and window table in memory and on disk is full of information.

"Symbol/Macro/Window allocation table is full."
The symbol table, macro table, and window table in memory and on disk is full of information.

"There were more '(' than ')'."
The parenthesis were mis-matched and the expression could not be properly evaluated.

"There were more '[' than ']'."
The dereference operators were mis-matched and the expression could not be properly evaluated.

"Third operand specified is illegal."
The third of the operands specified or its addressing mode is not allowed for this instruction.

"Time is: [[[minutes:] seconds.] milliseconds,] microseconds"
Time must be specified as MIN ':' SEC '.' MSEC ',' USEC

"Too many '('s to be parsed."
The expression evaluator can parse an expression with up to 10 open '(' or '['.

"Too many open files."
There are too many files open in DOS at the current time. PROBE will have at most 5 files open at any one time. This problem may be fixed by changing or increasing the "FILES = XXXX" parameter in the CONFIG.SYS file. (If omitted, DOS defaults to FILES=8).

"Trace qual BP".error in trace qualification range

"Unknown unit: Abort, Retry, Ignore?"
See "DOS critical error"

"Unknown command: Abort, Retry, Ignore?"
See "DOS critical error"

"Unknown instruction mnemonic."
The mnemonic listed is not a legal 68020, 68851, or 68881 instruction.

"Unknown media: Abort, Retry, Ignore?"
See "DOS critical error"

"Unrecognized object module format. Cannot load this program."
The currently recognized object module formats are: Motorola S-record format, Tekhex, Extended Tekhex, Atron binary, Microtec IEEE binary COFF format, Unix (tm) System V COFF format

"Valid qualifier values are "0", "1", or "X""
When entering qualifier bits (Logic Lines or IPL lines), the only valid values are '0' for logic 0, '1' for logic 1, or 'X' for don't care.

"Valid don't care bits are "." or "X""
When entering don't care bits, the only valid values are '.' for care and 'X' for don't care.

"Value expected but not found in expression."
A value (see definition of value) was expected but not found in the expression. This may mean that the symbol specified in the expression could not be found.

"Value written to memory is different from value read back."
The value written to memory (in the command input area) is different from the data read back from memory (displayed in the highlighted field.) This will only occur if Display-change Noverify-state is set to read-after-write.

"Write fault: Abort, Retry, Ignore?"
See "DOS critical error"

"Write fault or disk full."
The disk is full or write protected and the specified file could not be written. Note that the data that was being SAVED has not been saved.

"Write-protect error: Abort, Retry, Ignore?"
See "DOS critical error"

"<AltKey> is not a macro key."
The <AltKey> typed is not a macro and, thus, cannot be killed.

"<AltKey> is not a window key."
The <AltKey> typed is not a window and, thus, cannot be killed.

"-- Must have breakpoint address or Logic verb to activate."
An address must be specified for all verbs except Logic.

"-- Must have data size and data value for Logic verb."
The Logic verb must be accompanied by data size and value for logic lines to break on.

"-- May not have TO address or DATA with Execute verb."
Execute and HWExecute breakpoints may not be range or data breakpoints. Thus, there must be no TO address or DATA specified.

"-- Must have data value if data size <> none."
If a data size is listed, then a data value must also be listed.

"-- Range start address > range end address."
Start address of range must be below the end address.

"-- Too many execution breakpoints (16 max)."
There are a maximum of 16 breakpoints with the Execute verb.

"-- Too many HW execution breakpoints (4 max)."
There are a maximum of 4 breakpoints with the HWExecute verb.

"-- Too many HW breakpoints (4 max; Range=2, TraceQual=2)."
There are a maximum of 4 breakpoints with verbs other than Execute or HWExecute. Note that a range breakpoint (address TO address) takes 2 HW breakpoints and that trace qualification takes 2 HW breakpoints.

"-- Execution breakpoint address is not RAM."
The software execution breakpoint instruction address was not a RAM address and the breakpoint could not be written. Use the HWExecute verb.

"-- Range breakpoint may not cross megabyte boundary."
Range breakpoints (address TO address) are not allowed to cross a one megabyte boundary (MMMxxxxx TO MMMyyyyy -- MMM must be the same). Use two range breakpoints (MMMxxxxx TO MMMFFFFF and NNN00000 TO NNNyyyyy).

"-- More BPs are required for this sequential condition."
2. A arms B, reset-by C A and B required
3. A arms B arms C, reset-by D A, B, and C required
4. A arms (B or C), reset-by D A, B, and C required
5. (A or B) arms C, reset-by D A, B, and C required
6. A to B time greater than:<xxxxx> A and B required

"-- Less BPs are required for this sequential condition."
2. A arms B, reset-by C D not allowed
6. A to B time greater than:<xxxxx> C and D not allowed

"-- Breakpoint A may only be range for conditions 1 and 2."
Breakpoint A is allowed to be a range BP only for conditions 1 (A or B or C or D) and 2 (A arms B, reset-by C).

"-- Breakpoints B, C, D may only be range for condition 1."
Breakpoints B, C, and D are allowed to be a range BP only for condition 1 (A or B or C or D).

"-- Timeout condition only allowed for 2 hardware BPs."
Condition 6 (A to B time greater than:<xxxxx>) is only allowed if both breakpoint A and B are hardware BPs (verb is not Execute or HWExecute).

"-- Only conditions 1 and 2 allowed with trace qualification"
If trace qualification is enabled, only the breakpoints using 1 or 2 BPs are allowed. These are condition 1 (A or B or C or D) and condition 2 (A arms B).

"-- Only 2 breakpoints may be assigned with trace qualification"
Only two breakpoints may be assigned to BPs when trace qualification is enabled since trace qualification uses the other 2 breakpoints.

"-- Long data value only allowed on long (32 bit) bus."

"-- Long data must start on long word boundary (A1A0=00)"
A long data value can only be detected by the PROBE if it starts on a long word boundary on a 32 bit bus. That is, the entire long word access must take place in one bus cycle.

"-- Word data value not allowed on byte (8 bit) bus."
"-- Word data must start on word boundary (A0=0)"
"-- Word data must not start on end of long (A1A0<>11)"
A word data value can only be detected by the PROBE if it starts on a word boundary on a 16 or 32 bit bus. That is, the entire word access must take place in one bus cycle.

"-- Range breakpoint data size must match bus size."
In order for a range data breakpoint to correctly sense the data cycle, the data size being written must match the bus size listed in the

APPENDIX B
MAINFRAME COMPATIBILITY

The PROBE is compatible with the systems shown in Table B-1.*

Table B-1. Hardware Compatibility

SYSTEM	MANUFACTURER
AT	IBM
COMPAQ PORTABLE 286	COMPAQ
Tandy 3000	TANDY
Vectra	Hewlett Packard
SPERRY IT	SPERRY
LEADING EDGE AT	LEADING EDGE
FARADAY BOARD	FARADAY ELECTRONICS
WYSE AT	WYSE

For compatibility with the Compaq 386, the Breakpoint/Trace boards must be modified by Atron to put the boards into the IO space rather than the memory space of the Compaq 386. Call Atron for this modification.

*Compatibility with other systems will be added in the future. Contact Atron for additional information.

APPENDIX C
CONFIGURATION FILE

The configuration file is an ASCII file named PROBE.CNF, and the file provides PROBE system information to be used during some commands. You can use a text editor to change this file. Your text editor should store PROBE.CNF as ASCII text and should not include other hidden text editor control information in PROBE.CNF. PROBE.CNF specifies the following:

1. The base address of PROBE hardware.
2. The heap overflow file for symbol table expansion.
3. Programmable hardware features on PROBE
4. The instruction word that will be used for software execution breakpoints.

Here are some definitions for parameters which will be used in describing the configuration file. No distinction is made between upper- and lower-case characters.

Parameter	Definition
[c]	specifies 0 or more alphanumeric characters which does not include <Space>, '=', Cr, Lf
[s]	specifies 0 or more <Space> characters
["0"]	specifies 0 or more "0" characters
<"c">	specifies zero or more <Space> characters, followed by one <Space> or <"c"> character, followed by zero or more <Space> characters. There must be a <Space> or a <"c"> character. No distinction is made between upper- and lower-case characters.
{=}	specifies either a <Space> character or a '=' character i.e. there must be a <Space> or an '=' character.

PROBE.CNF PARAMETERS

Here are the parameters in the PROBE.CNF file:

1) "ADDR"[c]{"="}["0"]xy[c] [[s] "," [s] pppp]
   The Base Address 0xy000H of the Breakpoint/Trace boards in the AT memory space is set by writting the value 0xy000H to memory location FFFF0 in the AT. The following base addresses are allowed by PROBE. If omitted, the default base address is 0D0000 (xy=D0).

   0C4000 0CC000 0D0000 0D4000
   0D8000 0DC000

   EXAMPLES:

   ADDR=D4 sets base address to 0D4000
   ADDR D8000 sets base address to 0D8000
   ADDRESS 0C40 sets base address to 0C4000

   **********NOTE**********
   Some AT clones do not actually write data to address FFFF0 in the AT memory space. Some example systems which do are not are the Compaq 386. In these systems, the base address of the 68020 PROBE must be set by writing to IO space in the AT. If ",pppp" is specified, PROBE sets the base address of the Breakpoint/Trace boards with a write to port "pppp". Two PALs must be changed and a modification to the master break/trace board must occur for the ",pppp" option to work correctly. Call Atron for for these modifications.

   EXAMPLE:

   ADDRESS= 0D4, 100 sets base address to 0D4000
   selected by I/O write to 0100.
   ****

2) "HEAP"[c]{"="}drive-directory
   Put the heap overflow file "ATRON.HEP" in the specified drive-directory. If omitted, the file will be opened in the same drive-directory where the PROBE.CNF file was found. If there is no PROBE.CNF, the file will be opened in the default drive-directory at the time of starting PROBE.

Example: HEAPFILE=C:\Obj file is C:\OBJ\ATRON.HEP
HEAP \Misc file is \MISC\ATRON.HEP

3) "COLOR"[c]{"="}FC[c] [{","}BC[c]]
Set the Foreground Color and Background Color of the monitor. This configuration switch is ignored for monochrome monitors. Colors are:

'R' -- Red 'G' -- Green 'B' -- Blue
'C' -- Cyan 'Y' -- Yellow 'M' -- Magenta
'W' -- White 'K' -- blacK

If omitted, the color is set to "W, K" (white on black). If the background color is omitted, it is set to black.

Examples
COLOR=Y,K Yellow on black
COLORS Blue White Blue letters on white background
COLOR green Green letters on black background

3) "CPID"[c]{"="}CHOICE [{","}CHOICE [{","}CHOICE [{","}CHOICE ...]]]
Set the co-processor types for each co-processor id. For each co-processor id (0 to 7), the corresponding entry in the list is its co-processor type.

CHOICE --> "F"[c] for floating point co-processor
"M"[c] for memory management unit
co-processor any other for no co-processor

If CHOICE is omitted, it defaults to none.

EXAMPLES

```
CPID = MMU Float FLOAT          ; Co-processor 0 = MMU
                                ; Co-processor 1 = float pt
                                ; Co-processor 2 = float pt
                                ; all others not used
```

```
CpId m,f,n,f,x,f      ; Co-processor 0 = MMU
                      ; Co-processor 1 = float pt
                      ; Co-processor 2 = not used
                      ; Co-processor 3 = float pt
                      ; Co-processor 4 = not used
                      ; Co-processor 5 = float pt
                      ; all others not used
```

If CPID is omitted from the file, it defaults to:

CPID = MMU,FloatingPoint,None,None,None,None,None,None

4) "FLOAT"[c]{"="}FloatID
Set the co-processor to be used by the PROBE software for translation of floating point formats. Note that this co-processor is not dedicated to PROBE use. The internal state is saved before use by PROBE software and restored after use. However, this co-processor must be an actual Motorola 68881, software emulation of this co-processor cannot be used by the PROBE software.

FloatID --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | any_other

EXAMPLE:

```
FLOAT=1                 ; Use co-proc #1
FLOATPROC 3             ; Use co-proc #3
FLOAT -1                ; Do not use any co-proc
```

If not listed, the FloatID defaults to co-processor #1. If a FloatID of any number but {0..7} is used, if this co-processor is not designated as FLOATING_POINT in the CPID configuration option, or if this co-processor cannot be communicated with, no floating point translation will be done by the PROBE software.

6) "DONTCARE"[c]{"="}{"X"|"."}{"X"|"."}...{"X"|"."}
Specify the address don't care bits in the target system. This may be used, for instance, if the target system hardware does not decode all address bits (A31-A24 being the most common). A "X" or "x" denotes a don't care bit while a "." denotes a care bit. A space (" ") may be included anywhere and will be ignored. This specification will be the default value for don't care bits in the

software (i.e. in the Display-Map command, the Breakpoint command, etc.). The bits are listed from highest order address bit to lowest order, with the first bit listed corresponding to A31. Any omitted lower-order bits (for example, if only 16 bits are listed) are set to care (".").

EXAMPLE: Set A31-A24 don't care and A23-A0 to care.

DontCare = XXXXxxxx ....

Set A31-A28 to care, A27-A20 to don't care and A19-A0 care

DontCareAddressBits .... xxxx xxxx ....

4) "MCOUNT"[c]{"="}BR,BW,WR,WW,LR,LW
Do not modify this configuration parameter unless notified to do so by Atron.

5) "GCOUNT"[c]{"="}xx
Do not modify this configuration parameter unless notified to do so by Atron.

6) "EXECBP"[c]{"="}["0"]wwww
Specify the instruction word that will be used for execution breakpoints. The current default is wwww=484F (the BKPT #7 instruction). This value may be changed to any value from 4848-484F (BKPT #0-7). If this value is changed, 3 jumpers in the pod and a PAL in the pod need to be changed to match this value. This is available for users who are already using BKPT #7 and cannot allow the 68KPROBE to use it.

Example:

EXECBP = 4848	sets the exec bp to BKPT #0
EXECBP 484A	sets the exec bp to BKPT #2

7) "BAUD"[c]
[c] can be 2400,4800,9600. This parameter only applies to PROBE/1 versions. If omitted, or if any other value is listed, 9600 baud will be used. No parity, 8 data bits, and 2 stop bits are always used and cannot be changed.

8) "COM" [c][s][c]
   [c] is either 1 or 2 for COM1: or COM2:. This parameter only applies to PROBE/1 versions. If omitted, or if any other value is listed, COM1: will be used.

DEFAULT CONFIGURATION FILE

The default PROBE.CNF file as supplied on your distribution diskette does not contain any settings, therefore, PROBE assumes the defaults described earlier. If PROBE does not find the file PROBE.CNF in the current directory or in a DOS "PATH" spec, then these default parameters are used.

APPENDIX D
TEXT FORMATS FOR MACROS, WINDOWS,AND INITIALIZATION FILES

Macros, Windows, and Initialization files are stored as text files. Macro files are are created with the Macro Save command. Windows are created with the Window Save command. Initialization files are created with the Initialize Save command. Macros and windows can be edited on-line with the Macro Edit and Window Edit commands. Macro, Window, and Initialization files can also be edited offline with a standard text editor and stored as text files. Be sure that your editor only stores the file as pure ASCII text and does not include additional control codes. The formats for these files is described below.

MACRO FILE FORMATS

In macro editing, both on- and off-line, the special keyboard keys are specified exactly as described below:

SPECIFICATION	DESCRIPTION
<Enter>	Enter key.
<Esc>	Esc key.
<Tab>	Tab key.
<Bs>	<- backspace key.
<Home>	Home key.
<End>	End key.
<PgUp>	PgUp key.
<PgDn>	PgDn key.
<CtrlHome>	Home key with the Ctrl key held down.
<CtrlEnd>	End key with the Ctrl key held down.
<CtrlPgUp>	PgUp key with the Ctrl key held down.
<CtrlPgDn>	PgDn key with the Ctrl key held down.
<CursorUp>	up arrow key.
<CursorDown>	down arrow key.
<CursorLeft>	left arrow key.
<CursorRight>	right arrow key.
<Ins>	Ins key.
<Del>	Del key.
<F1> to <F10>	function keys.

<Alt?> alt key. Possible values for '?' are: any letter A..Z, Space (i.e.<Alt >) any number 0..9 - or = F1 to F10

The format for macros in a macrofile are:

MACRONAME [':' MACROTYPE]
MACRODEFINITION
BLANKLINE

WHERE:

MACRONAME	an AltKey	
MACROTYPE	'I'	for a conditional macro
	'L' 'F'	for a loop forever macro
	'L' 'C' COUNT	for a loop count macro. COUNT is an expression
	'L' 'W' CONDITION	for a loop while macro. CONDITION is a boolean expression which is either TRUE (not 0) or FALSE (0)
MACRODEFINITION		keystrokes for macro
BLANKLINE	a line containing only a Cr (or Cr,Lf) or the End-of-file to signal end of macro.	

EXAMPLES: This macro file contains the macros AltI and AltT. Macro <AltI> is a macro which performs a Display Single-address Long-word Write to location CD000000 with data 4C0B000. Macro <AltT> will loop while (D0 < 10). It will execute macro <AltI>, then Unassemble from location 80000114 for 5 instructions.

```
<AltI>
<Esc>dslwCD000000<Enter>
04C0B000<Enter><Esc>

<AltT>:LWd0<10
<AltI>
u80000114<Enter>5<Enter><Esc>
```

NOTE: The MACRODEFINITION may contain Cr (or Cr,Lf) characters at any point. These characters are ignored when being read in, unless the Cr is the only character on a line.

WINDOW TEXT FILE FORMATS:

The format for Windows in a window file are:

WINDOWNAME
FIELDSPEC
FIELDSPEC
FIELDSPEC
.
.
NULLFIELD

WHERE:

WINDOWNAME altkeyname

FIELDSPEC
ROW ',' COL
{EXPRESSION_SPEC | STRING_SPEC | RANGE_SPEC | LABEL_SPEC}

ROW a hex number in the range 0..18. Row for field from start of window.

COL is a hex number in the range 1..4F. Column for field from start of window Note: the top, left corner of the window is 0,1.

EXPRESSION_SPEC is 'E' EXPRTYPE
expression

EXPRTYPE is {'B' | 'W' | 'L' | 'A' | 'D' | 'S'}
Note: byte, word, long, ASCII, decimal, signed-decimal

STRING_SPEC {LENGTH_STRING | ZEROTERM_STRING}
LENGTH_STRING 'S' 'L'
expression for address of string
expression for length of string

ZEROTERM_STRING 'S' 'Z'
expression for address of zero-terminated string

RANGE_SPEC 'R' RANGETYPE
expression for start address of range
expression for end address (or '+' length) of range

RANGETYPE {'B' | 'W' | 'L'} for byte, word, long.

LABEL_SPEC 'L' label string

NULLFIELD FF,FF

EXAMPLES: The window will be opened by typing <AltW>. In the top left corner of the window (location 0, 1) the Label "D0=" will be printed. Then, at location 0,4 (right after the '=') the register d0 will be printed as a Long-word Expression. Next, at location 1,1 (the start of the next

line), the String which starts at address "\StringAddress" and whose Length is contained in the 16-bit variable "\StringLength" will be printed. Finally, the Range of Bytes starting at location "\BufferStart" will be printed on the next line of the window. The range is 8 bytes long.

```
<AltW>
0,1
L
D0=
0,4
EL
d0
1,1
SL
\StringAddress
[\StringLength].w
2,1
RB
\BufferStart
+8
FF
```

Thus, the window display would look like:

```
D0=55AA55AA
StringAddress
00 01 02 04 55 33 22 11
```

INITIALIZE FILE FORMATS

The initialization file consists of blocks for each set of information in the file. The blocks of information are saved:

MAP block
MODULE block
PROGRAM block
MACRO block
WINDOW block

The blocks are saved in the order shown above. Each block begins with a header to denote the type of block. The information for that block follows the header. The information is terminated by a blank line. All blocks are saved when the Init Save filename command is issued.

When the blocks are loaded with the Init Load filename command, the order of the blocks in the file does not matter. The file is searched and information loaded in the same order as the information is stored by the Init Save command. If there are two block specifications for the same block type (e.g. two MAP blocks), then only the first block is loaded. If there is no block specification for any block, then that information is not changed in the current context.

Here are some definitions for parameters which will be used in describing the initialization file. No distinction is made between upper- and lower-case characters(except for module name).

Parameter	Definition
c	specifies any non-space character
[c]	specifies 0 or more non space characters
[s]	specifies 0 or more <Space> characters
{,}	denotes any number of space characters (0..n), followed by a space or a ',' character, followed by any number of space characters (0..n)

The blocks stored in this file are described as follows:

MAP Block:

"MAP" [c] [s] "=" [s] <Enter>
Start {,} End {,} BusSize? {,} Guard? {,} Map? [{,}, Write?][{,}Wait?]
Start {,} End {,} BusSize? {,} Guard? {,} Map? [{,}, Write?
<Blank Line>

where

Start and End are address expressions
BusSize is {"B" | "W" | "L"} [c]
Guard?, Write? are {"Y" | "N"} [c]
Map? is a digit indicating mapped memory array number
Wait? is a digit indicating number of wait states in MAP RAM

Examples: This example sets a map region from 00000000 to 000FFFFF to be a 32 bit, unguarded, unmapped region. It also set a region from 80000000 to 8000FFFF to be an 8 bit, unguarded region. This second region is mapped to the PROBE MAP RAM boards and is write protected

```
MAP =
0,fffff,Long , n, n
80000000,8000ffff,b,n,0,yes,y
```

MODULE Block:

"MOD" [c] [s] "=" [s] <Enter>
modulename {,} Loadable? [{,} Stepable? [{,} filename]]
modulename {,} Loadable? [{,} Stepable? [{,} filename]]
...
<Blank line>

where

modulename or filename are c [c]
Loadable? and Stepable? are {"Y" | "N"} [c]

Examples: This example tells the PROBE symbol table manager to neither load nor source step the symbols in the module FtoCIO. The module FtocM should have all symbols loaded, should be allowed to be source stepped through, and is assigned to the file C:\SOURCE\FTOCM.C.

```
MODULES =
FtoCIO, No, No
FtocM, Yes, y, C:\Source\FtocM.C
```

PROGRAM Block:

"PROG" [c] [s] "=" [s] <Enter>
filename [MemSpace["," "LoadOffset[["," "LoadSymbols?["," "LoadCode["," "FileType]]]]]

where

filename is c [c]
MemSpace is
{"0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"UD"|"UP"|"UR"|"SD"|"SP"|"CPU"}
LoadOffset is 00000000 to FFFFFFFF
LoadSymbols is Y or N
LoadCode is Y or N
FileType is {'A','S','B','U','I'}

The defaults are MemSpace is 6 or 2 depending upon state of Supervisor bit, all others default to current setting from Load Options command in PROBE software.

Examples: This example first loads the program "\OBJ\FTOC.HEX" into Supervisor Program space. It then loads the file "FTOC.DAT" into the current load space (which, in this example, will still be set to SP).

PROGRAM =
\obj\ftoc.hex, sp
ftoc.dat

Load FTOC.S into memory space 6. Add 0 to symbols and loadd addresses. Load symbols with program load both code and data as an S record file

PROG=
FOTC.S,6,00000000,Y,Y,S

Load FTOC.S into memory space 6 and add 800000000 to symbol and load addresses. Load symbols but not code or data. Automatically determine file type to load.

FTOC.S,6,800000000,Y,N,A

Load code from FTOC.S with a 0 offset *then* load symbols from FTOCS.S and offset all addresses by 800000000.

MAC Block:

"MAC" [c] [s] "=" [s] <Enter>
filename
filename
...
<Blank line>

where

filename is c [c]

Examples: This example loads macros from the file "\OBJ\FTOC.MAC".

MACROS=
\obj\ftoc.mac

WIN Block:

"WIN" [c] [s] "=" [s] <Enter>
filename
filename
<Blank line>

where:

filename is c [c]

Examples: This example loads windows from the file "\OBJ\FTOC.WIN". It then loads more windows from the file "\REG.WIN".

WINS=
\obj\ftoc.win
\reg.win

APPENDIX E
FILES ON YOUR PROBE DISKETTES

There are several files on your PROBE diskettes which may or may not be needed depending upon what you are doing. Only those used for "EXECUTING PROBE SOFTWARE" are required. A list of these files and their purpose is given below:

PROBE FILES	VERSION	DESCRIPTION
PROBE.EXE	1.0	Executable file for "Running" PROBE software
PROBE.CNF	N/A	Default config file
ATRON.HEP	N/A	Default heap file
FTOC.S	N\A	Demo program executable file
FTOC.INI	N\A	Demo program initialization file
FTOC.WIN	N\A	Demo program window file
FTOCM.C	N\A	Demo program C source file module 1
FTOCIO.C	N\A	Demo program C source file module 2
FTOCSTRT.A68	N\A	Demo program assy lang source file module 3
REG.WIN	N\A	A window file which displays registers
CONF020.EXE	1.0	Diagnostic confidence test
ABS2BIN.EXE	1.0	Binary image to loadfile utility (see Apn G)

SOURCE	VERSION	DESCRIPTION
SOURCE.EXE	1.0	Executable file for "Running" SOURCE
PROBE.CNF	N/A	Default configuration file

NOTE: N/A means there is no applicable version number for this file.

APPENDIX F
LANGUAGE COMPATIBILITY

68020 PROBE allows you to use the symbolic information from your program during debugging instead of absolute numbers. The symbolic debugging information is passed to the PROBE from the compiler using controls which are discussed here. This symbolic information may consist of public variables, public procedures, functions, subroutines, modulenames, and high level language line numbers. Some compilers will also produce symbols for local variables and procedures.. In addition, if the 68020 SOURCE PROBE version of the software is running on, then source level debugging can be achieved.

The sections which follow describe the considerations when using PROBE with C, Pascal, and Assembly language. You may want to go directly to the section which applies to you. A description of the compiler and linker controls required for several different manufacturers is described.

USING MICROTEC RESEARCH(TM) LANGUAGES WITH PROBE

MICROTEC RESEARCH C - PC BASED

Here are sample batch files which you can run on your PC to generate symbolic debugging information. See the Microtek Research manual for details.

```
MCC68k /cpu=68020 /debug/ /nolp %1.c
asm %1
asm68k %1.src, %1.obj, %1.lst /b
lod 68k @%1.cmd, %1.map, %1.cof
```

the command file referenced by the %1.cmd would look something like this:

```
chip 68020
format ieee
name programname
load object module list
```

an example command file for the applications example in chapter 4 would be

```
chip 68020
format ieee
name ftoc_68020
order ?????09, ????13, ????14
sect ?????09 = $00000400
list d,s,t,x
load ftocstrt.obj, fotcm.obj, ftocio.obj
end
```

other compatible Microtek Research software

VAX	C, Pascal, Assembler
PC/AT	ASSEMBLER, Pascal
SUN MICROSYSTEMS WORKSTATION	C, Assembler

COMPATIBLE GREENHILLS SOFTWARE

VAX	C
SUN MICROSYSTEMS WORKSTATION	C

COMPATIBLE UNIX SYSTEM V SOFTWARE

PC/AT BASED FROM MOTOROLA	C, Asy using COFF records
VAX BASED FROM MOTOROLA	C, Asy using COFF records
PROPRIETARY WORKSTATIONS	C, Asy using COFF records

SUN MICROSYSTEMS

C, Asy	User must convert a.out to COFF

APPENDIX G
OBJECT MODULE FORMATS

PROBE is compatible with a number of object module formats. PROBE can load the code, data, and symbolic debugging information for these formats. You can tell PROBE which format to load with the Load Options File-type command. You can then select any of the types shown here:

Automatic - determine
S-records (ext)tekhex
Binary-image
Unix(tm)System V
Ieee-binary-coff

A summary of these formats is described next. If you choose the Automatic-determine File-type, PROBE looks at the first character in the File to determine the File-type automatically.

S RECORDS, TEKHEX, EXTENDED TEKHEX

PROBE supports the following hex formats:

Motorola S-Records	(records start with 'S')
Tekhex records	(records start with '/')
Extended Tekhex records	(records start with '%')

S-RECORD FORMAT

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

S-RECORD CONTENT

When viewed by the user, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first character

representing the high-order 4 bits, and the second the low- order 4 bits of the byte.

The 5 fields which comprise an S-record are shown below:

type record length address code/data checksum

Where the fields are composed as follows:

FIELD	PRINTABLE CHARACTERS	CONTENTS
type	2	S-record type--S0,S1,etc.
record length	2	The count of the character pairs in the record,excluding the type and record length.
address	4,6, or 8	The 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 teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S -record).
checksum	2	The least significant byte of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

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 numbers generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

S-RECORD TYPES

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. An S-record-format module may contain S-records of the following types:

S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records.

S1 A record containing code/data and the 2-byte address at which the code/data is to reside.

S2 A record containing code/data and the 3-byte address at which the code/data is to reside.

S3 A record containing code/data and the 4-byte address at which the code/data is to reside.

S5 A record containing the number of S1, S2, and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.

S7 A termination record for a block of S3 records. The address field may optionally contain the 4-byte address of the instruction to which control is to be passed. There is no code/data field.

S8 A termination record for a block of S2 records. The address field optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

S9 A termination record for a block of S1 records. The address fields may optionally contain the 2-byte address of the instruction to which control is to be passed.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3- or 4-byte address. Normally, only one header record is used, although it is possible for multiple header records to occur.

EXAMPLE

Shown below is a typical S-record-format module, as printed or displayed:

```
SOO6OOOO4844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The module consists of one SO record, four S1 records, and an S9 record. The S0 record is comprised of the following character pairs:

S0 S-record type SO, indicating that it is a header record

06 Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.

0000 Four-character 2-byte address field, zeros in this example.

4844 ASCII H, D, and R - "HDR".

52

1B The checksum.

The first S1 record is explained as follows:

S1 S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.

13 Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.

00 Four-character 2-byte address field; hexadecimal

00 address 0000, where the data which follows is to be loaded.

The next 16 character pairs of the first S1 record are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the program are written in sequence in the code/data fields of the S1 records:

```
OPCODE          INSTRUCTION
------          -----------
285F            MOVE.L  (A7)+,A4
245F            MOVE.L  (A7)=,A2
2212            MOVE.L  (A2),D1
226A0004        MOVE.L 4(A2),A1
2490008         MOVE.L FUNCTION(A1),D2
237C            MOVE.L #FORCEFUNC,FUNCTION(A1)
```

The balance of this code is continued in the code/data fields of the remaining S1 records, and stored in memory location 0010, etc.

2A The checksum of the first S1 record.

The second and third S1 records each also contain $13 (19) character pairs and are ended with checksums 13 and 52, respectively. The

fourth S1 record contains 07 character pairs ans has a checksum of 92.

The S9 record is explained as follows:

S9 S-record type S9, indicating that it is a termination record.

03 Hexadecimal 03, indicating that three character pairs (3 bytes) follow.

00 00 The address field, zeros.

FC The checksum of the S9 record.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted. For example, the first S1 record above is sent as:

type	length	address	code/data	checksum
S	1	130000	285F	2 A

5 3 3 1 3 1 3 3 3 0 3 0 3 0 3 0 3 2 3 8 3 5 4 6 ...3 2 4 1

0101 0011 0011 0001 00

EXTENSIONS TO HEX RECORD FORMATS

In addition to the above hex-record object module formats, PROBE can also support the following extensions in the format of the file:

$$
SymbolName SymbolAddr....SymbolName Symbol Addr (these are publics i.e. no modulename)
$$ ModuleName
SymbolName SymbolAddr SymbolName Symbol Addr ...
SymbolName SymbolAddr ...
...
$$ ModuleName
SymbolName SymbolAddr SymbolName Symbol Addr ...
SymbolName SymbolAddr ...
...
$$ (this is the end of symboltable. Additional blank lines may occur after this but nothing else.)

Here are some definitions for the previous format:

ModuleName	the name of the module that produced the symbols
SymbolName	the name of the symbol to be defined
LineNumber	a symbol name of the form: '#' DecimalLineNumber.
SymbolAddr	the hex address of the symbol (with or without leading '$') (with or without leading '0's) (with or without trailing 'H')

S-Records or Tekhex records or Extended Tekhex records may follow. There may be any number of spaces before each symbol name and between each symbol name and its corresponding address. Each line may have any number of symbols but must be less than 100 characters long. Each symbol may be of any length as long as it and its address fit on a line. The "$$ <nothing>" line marks the end of the symbol table and the start of the hex object module with one exception. If a "$$ <nothing>" line is encountered before any symbols have been loaded, then the following list of symbols is assumed to contain PUBLICS.

Note that Extended Tekhex format may define symbols in two ways. It may define the symbols using the "$$" format described here. It may also include Symbol Definition (Type 3) records in the object module.

Example 1:

```
$$
PublicSymbol1 $00000400    PublicSymbol2 $00000402
PublicSymbol3 $00000404    PublicSymbol4 $00101000
$$ Module1
Sym1 $500   Sym2 $505   Sym3 $0600
#10 $80000000 #20 $80000008
$$
S0.... (Start of S-Records)
```

```
This file defines 7 symbols and 2 line numbers:
\PublicSymbol1 at 00000400
\PublicSymbol2 at 00000402
\PublicSymbol3 at 00000404
\PublicSymbol4 at 00101000
\Module1\Sym1  at 00000500
\Module1\Sym2  at 00000505
\Module1\Sym3  at 00000600
\Module1#10    at 80000000
\Module1#20    at 80000008
```

Example 2:

```
$$ Module1
Sym1 $00000400   Sym2 $00000402   Sym3 $00000404
Sym4 $00101000
$$ Module2
Sym1 $00000500   Sym2 $00000505   Sym3 $00000600
#10 $80000000 #20 $80000008
$$
/32.... (Start of Tekhex records)

This file defines 7 symbols and 2 line numbers:
\Module1\Sym1 at 00000400
\Module1\Sym2 at 00000402
\Module1\Sym3 at 00000404
\Module1\Sym4 at 00101000
\Module2\Sym1 at 00000500
\Module2\Sym2 at 00000505
\Module2\Sym3 at 00000600
\Module2#10   at 80000000
\Module2#20   at 80000008
```

BINARY IMAGE LOAD FILE FORMAT

A utility program named Abs2Bin is included on the PROBE distribution diskettes which takes an absolute binary memory image and produces an output file in binary loadable format which can be loaded by PROBE. This utility is invoked by:

```
Abs2Bin InputFile [OutputFile] [/aaddress]
```

InputFile is the absolute memory image and OutputFile is the file loadable by PROBE. If omitted, OutputFile defaults to the same file name as InputFile with extension .BIN. A load address can be included in the OutputFile if the /aaddress option is specified. If omitted the load address defaults to 00000000. NOTE: There must be no space between the 'a' and the ADDRESS.

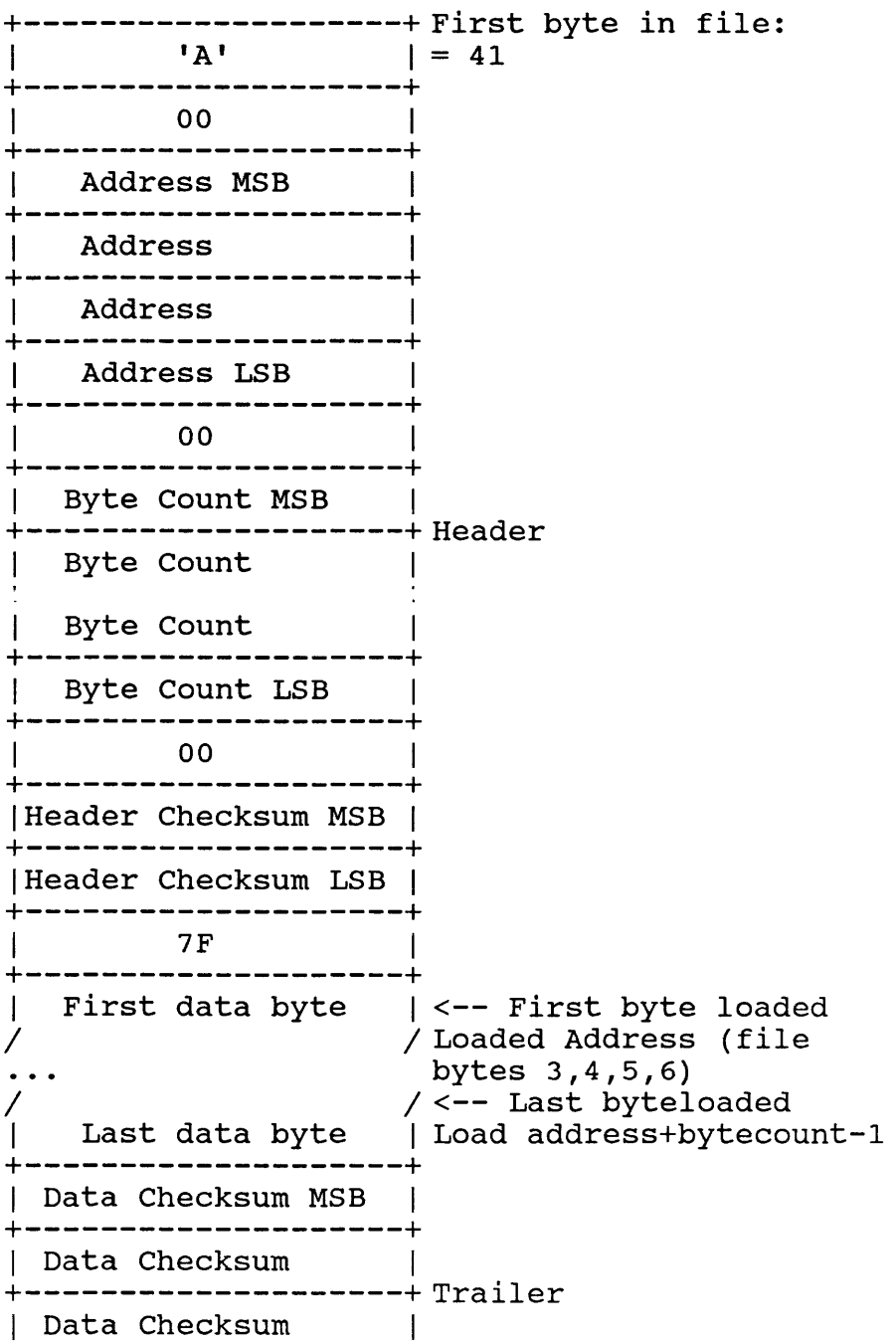
EXAMPLE To load an absolute memory image, contained in the file \OBJECT\ROM.ABS, at address FF400000 and produce the output file \OBJECT\ROM.BIN:

```
Abs2Bin \Object\Rom.Abs \Object\Rom /aff400000
```

When in the PROBE software, you can load this program with the following PROBE command:

```
LPROM.BIN
```

Here is a picture of the file format for the OutputFile.


First byte in file:
= 41
'A'
00
Address MSB
Address
Address
Address LSB
00
Byte Count MSB
Header
Byte Count
Byte Count
Byte Count LSB
00
Header Checksum MSB
Header Checksum LSB
7F
First data byte
<-- First byte loaded
Loaded Address (file
bytes 3,4,5,6)
<-- Last byteloaded
Load address+bytecount-1
Last data byte
Data Checksum MSB
Data Checksum
Trailer
Data Checksum

```
+--------------------+
| Data Checksum LSB  |
+--------------------+
|        1A          | End-of-file mark
+--------------------+
```

Header Checksum is the 16 bit sum of header bytes (not negated or processed).

Data Checksum is the 32 bit sum of all data bytes (not negated or processed).

Note that none of the checksum bytes are included in the checksum calculation.

UNIX SYSTEM V COFF RECORDS

Coff records are defined in Chapter 8 of the UNIX SYSTEM V SUPPORT TOOLS GUIDE Release 2.0. This manual is available from AT&T and is commonly available from computer book stores.

MICROTEK RESEARCH (Ieee binary coff)

This format is a proprietary format from Microtek Research of Santa Clara Ca.

APPENDIX H
LOGIC PROBES

Logic PROBES are plugged into the POD see Chapter 1. They provide four inputs to the breakpoint and trace logic. The logic lines designated as 0, 1, 2, and 3 in the Breakpoint screen and shown under the Log column of the Trace Raw screen. The Logic PROBES also connect to the Breakpoint Detect output which provides a pulse when Read, Write, Fetch, or Any breakpoint verbs are detected (see Breakpoint command and Hardware Breakpoint Enable command). The Logic PROBES are not keyed and can be plugged into the POD in either direction. In order for the labels on the Logic Probes to match the software, plug these Probes in as shown below:

Color	Label on Logic Probe	POD connector PL18 Pin#	Software Designation
Brown	LOG0	1	Logic Line 0
Red	LOG1	2	Logic Line 1
Orange	LOG2	3	Logic Line 2
Yellow	LOG3	4	Logic Line 3
Green	BREAK	5	Breakpoint Detect

APPENDIX I
PROBE POWER SUPPLY REQUIREMENTS

PROBE has been designed to plug into a standard IBM PC AT with the following configuration.

640k dynamic ram
1 floppy disk drive
1 hard disk drive
1 floppy disk/hard disk controller
1 monochrome, color graphics adapter, or EGA

If you have additional boards plugged into you target system, it may not be able to supply adequate power to PROBE. The 68020 PROBE/3 which includes Breakpoint/Trace boards, POD and 1/2 Mbyte MAP RAM draws 13 amps of +5V from the AT computer. The 68020 PROBE buffer assy draws 3 amps of +5V from the target socket.

APPENDIX J
ELECTRICAL CHARACTERISTICS OF 68020 POD

1. The following signals go directly from the 68020 to the target socket. They are not intercepted or gated by the 68020 probe. They are buffered at the 68020 buffer assy and sent to the 68020 pod. The buffer load is:

   input high current .02 m.a.
   input low current -.5 m.a.

   The following signals are affected:
   A0 thru A31 FC0,FC1,FC2 SIZ0,SIZ1 ECS,OCS RMC IPEND CLK

2. The 32 data bus lines are intercepted by 74ALS245 tristate buffers which have the following specs:

   input high current .02 m.a.
   input low current -.1 m.a.
   propagation delay 3 ns min, 10 ns max

   Note: This additional delay decreases the address valid/ control valid to data valid 68020 cpu spec by 10 ns max. The 68020 cpu at room temp provides address and control faster than data sheet worst case spec and this 10 ns degradation is not seen in most applications.

3. The control signals AS, DS, DBEN, and R/W are intercepted by a 74AS257 which has the following specs:

   Tristate current +/- .05 m.a.
   Output hi current 15 m.a.
   Output lo current -48 m.a.
   propagation delay 1 ns min, 6 ns max

   These signals are held hi when not emulating. See the note above regarding system timing degradation.

4. The BERR, CDIS, DSACK0, and DSACK1 signals are intercepted by a 74AS257. The pertinent specs are:

input high current .02 m.a.
input low current -.5 m.a.
propagation delay 1 ns min, 6 ns max

5. The AVEC, IPL0, IPL1, and IPL2 signals are intercepted by a 74LS157. The pertinent specs are:

   input high current .02 m.a.
   input low current -.4 m.a.
   propagation delay 9 ns TYP, 14 ns max

6. The bus arbitration signals BR, BG, and BGACK are gated with a 74AS32. The pertinent specs are:

   BR, BG input high current .02 m.a.
   input low current -.5 m.a.
   BGACK output hi current -2 m.a.
   output lo current 20 m.a.
   propagation delay 1 ns min, 6 ns max

7. The HALT signal is gated with a MOSFET. The specs are:

   input high current .02 m.a.
   input low current -.5 m.a.
   output lo current 10 m.a.
   output hi current open drain
   propagation delay 3 ns min, 10 ns max

8. The RESET pin is driven by an open emitter bipolar transistor in a wire or configuration. When the hardware reset command is given to the 68020 probe, it drives this pin low and any target circuit attached to this pin will also be affected.
9. BR, BG, and BGACK operate in the target system even when PROBE has stopped emulation in the target system unless masked by using the PROBE Hardware commands.
10. Memory reference cycles conducted in a PROBE MAP RAM array are duplicated in the target system.

APPENDIX K
TECHNICAL REPORTS

This section provides technical information, potential problem areas, and bug reports for the PROBE. It will be updated periodically if you send in your registration card.

This appendix contains the following Technical Reports:

WHAT HAPPENS WHEN THE TARGET SYSTEM HANGS

If the target system hangs because DACK's are not returned to the processor, PROBE cannot regain control until:

1. <ESC> is typed
2. The PROBE watchdog timer times out

For both of these cases, the PROBE forces a Bus Error to the 68020 in the target in order to regain control. This pushes information into the target system stack. If the target stack pointer is not valid, a double bus fault may occur and the target will continue to hang. The following message is displayed:

Could not STOP execution of target processor. Reset? Y/N

If you choose to reset the target then register values will be lost.

If the target stack is valid then PROBE can regain control and the following message is displayed:

Execution stopped by Bus Error in target system. Reset? Y/N

If you do not reset, the registers remain unchanged.

MYSTERIOUS BREAKPOINT 7 INSTRUCTION

Sometimes when the single step command hangs up, and you type <ESC> to regain control of the system, you find that the next instruction to be single stepped is a Breakpoint 7. You may type <ESC> again to get out of the single step command entirely, then unassemble the code at the same spot. You may then notice the Breakpoint 7 is still there, even if the program is in Eprom memory in your target.

What probably happened is that a Bus error occurred during the single step operation. PROBE single steps by executing a Go command with an HWExecute breakpoint set on the next instruction which the 68020 could execute (with possible branches this could be at one of 3 locations so PROBE sets 3 HWExecute breakpoints). If, however, a Bus Error occurs between the Go and the breakpoint, the step will not occur if the interrupt handler in the target does not

return to the next instruction to be stepped. You regain control by typing <ESC> as explained in the previous tech note. This may leave PROBE in an indeterminent state.

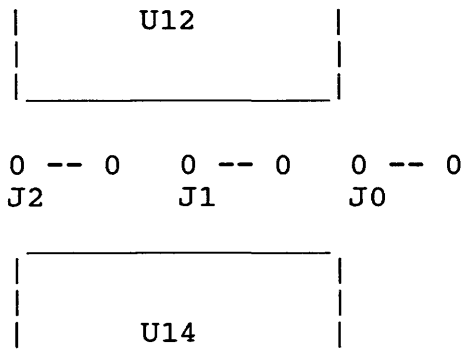
PROBE implements the HWExecute breakpoint by forcing a Breakpoint 7 instruction into the 68020 with hardware logic in the POD (it does not put the Bk 7 in memory). When PROBE goes to an indeterminent state, the PROBE logic may still be decoding the address of the instruction where it tried to put the Breakpoint 7. The POD will then always force the Breakpoint 7 instruction into the 68020 whenever this address is referenced for any PROBE command. When this occurs, the only solution is to do a Hardware Reset command.

S/W BREAK POINT AND H/W EXECUTION BREAK POINT USAGE (single step)

The 68020 probe normally uses bp # 7 as the bp that is set in user ram, for a s/w bp, and is forced on the bus for either a h/w execution bp or when single stepping.

The other 7 bp's can be optionally selected by changing a pal on the 68020 pod and installing a jumper on the 68020 buffer assy. The pal is socketed and is identified as chgadrxn; where x = the pal revision level and n = 0 to 6 identifies the bp level decoded (no n indicates the normal level 7). The jumpers can be located on the bottom of the buffer assy between U 12 and U 14 as shown below. No jumper is the normal level 7 condition.

BP#	J2	J1	J0
0	IN	IN	IN
1	IN	IN	OUT
2	IN	OUT	IN
3	IN	OUT	OUT
4	OUT	IN	IN
5	OUT	IN	OUT
6	OUT	OUT	IN
7	OUT	OUT	OUT



GENERATING A C LIST FILE WITH LINE NUMBERS

This utility is provided as a convenience. It is not necessary for use by PROBE software. The CLIST program found on the PROBE diskette accepts an input "C" source file and produces a listing file. It also expands tabs into spaces from the source file so that the file may be printed on any line printer.

FORMAT: CLIST [sourcefile [,destinationfile [,spaces per tab]]] [options]

If the files are not listed, then the user is prompted for the file names. Source lines are transferred to the **destinationfile** in the format specified later. All tabs are expanded to spaces with tab stops every specified number of columns. The two **options** which can be specified are i and c:
-i<include drive>
I specifies the drive to be searched for all include files. example-ia (no intervening spaces)
-c
C specifies that comments do not nest. An */ ends all comments currently in effect, no matter how many /* have occurred.

This is the LISTING FORMAT which is produced:

C ***** 00000. xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

C — Is a comment indicator. The 'C' is placed in this column if the first character in the source line is considered to be inside a comment.

***** — Is the include nesting level from the include files which are currently being used. Each include file which includes another file will add one more * to this field. There are a maximum of 5 *.

00000 — Is the line number of the line in the current file. Each <CR> in the file increments the line count. The line count starts at 1 in each file.

xxxxx — Is the line of source code.

EXAMPLES: Produce a listing in ftocm.lst with tabs set every 8 spaces. Include files are on the default drive and comments nest.

```
clist ftocm.c, ftocm.lst, 8
```

Produce a listing directly to the line printer with tabs set every 4 spaces. Include files are on the default drive and comments nest.

```
clist ftocm.c, lpt1:, 4
```

Produce a listing directly to the line printer with tabs set every 4 spaces. In addition, all include files exist on drive C, and comments do not nest.

```
clist ftocm.c, lpt1:, 4 -ic -c
```

You are prompted for the destination file and for the number of spaces per tab. All include files exist on drive A, and comments nest.

```
clist ftocm.c -ia
```

TIMEOUT BREAKPOINTS WITH POP UP WINDOWS

If you pop up a Watch Window during emulation when a timeout breakpoint is set, the time PROBE spends displaying the window is not included in the breakpoint. Therefore, even if the time exceeds

the timeout value, PROBE may not cause the breakpoint since it disables the timer while updating the Window.

PROBE PERFORMANCE FOR BLOCK OPERATIONS

Here are some reference numbers for block operations in the Xfer command. They were derived in the following environment:

> Block operation on 64k bytes of memory
> 68020 operating at 20 mhz

Block Save	32 sec
Compare block	33 sec
Find string	22 sec
Move block	29 sec
Set block	15 sec

APPENDIX L
MORE ON CONF020

The CONF020.EXE program is used to perform a confidence test on the 68020 pod for the Probe /2 and Probe /3 products. This test will run in one of two modes:

DEMO BOARD: The 68020 buffer assembly is connected to the ATRON DEMO board. This mode will test target system access as well as performing internal diagnostics.

INTERNAL ONLY: The 68020 is connected to something other than the ATRON DEMO board. Only the internal diagnostics are performed; no cycles are run in the target system. If the target system can support the access cycles and conditions as described here, then the DEMO BOARD MODE may be used in the target system:

1. Cache always enabled from target system (CDIS/ == 1)
2. Bus error never occurs in target system (BERR/ == 1).
3. RAM must exist from 00000000 to 0002FFFF (192K).
4. Memory accesses performed in the target as shown in Table 1-1 (memory does not necessarily need to exist at these locations but a DACK signal must be returned):

Accesses in target for confidence test Demo Board Mode

Address	Access Size	Access type	DACK required
00000000	Byte, Word, Long	Read/Write	any DACK
00000001	Byte, Word, Long	Read/Write	any DACK
00000002	Byte, Word, Long	Read/Write	any DACK
00000003	Byte, Word, Long	Read/Write	any DACK
000004xx	Byte, Word, Long	Execute/Write	any DACK
00010000	Byte	Read/Write	any DACK
00010xxx	Byte,Long	Read/Write	any DACK
00020000	Byte,Long	Read/Write	32 bit DACK
00020001	Byte	Write	32 bit DACK
00020002	Byte,Long	Read/Write	32 bit DACK
00020003	Byte	Write	32 bit DACK
00040000	Byte	Read/Write	any DACK
00080000	Byte	Read/Write	any DACK
00100000	Byte	Read/Write	any DACK
00200000	Byte	Read/Write	any DACK
00400000	Byte	Read/Write	any DACK
00800000	Byte	Read/Write	any DACK
1000000	Byte	Read/Write	any DACK
02000000	Byte	Read/Write	any DACK
03000000	Byte	Read/Write	any DACK
04000000	Byte	Read/Write	any DACK
08000000	Byte	Read/Write	any DACK
10000000	Byte	Read/Write	any DACK
20000000	Byte	Read/Write	any DACK
40000000	Byte	Read/Write	any DACK
55555554	Long	Read	any DACK
55555555	Byte	Read/Write	any DACK
80000000	Byte	Read/Write	any DACK
AAAAAAAA	Byte,Long	Read/Write	any DACK

INDEX

Other Entries

A

B

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

ATRON REPAIR SERVICE POLICY

Atron will provide service repair of the 68020 PROBE on the following basis.

PROBE within 1 year hardware warranty period

Atron will send a new board to customer and customer will return failed board to Atron. Atron will pay for UPS surface freight to customer. (Customer pays for upgraded freight service.) Customer pays for return freight of failed board to Atron. A PO # will be required in advance of sending new board to customer for the price of a new system and is automatically canceled upon arrival of the failed board to Atron.

PROBE outside 1 year hardware warranty period

Atron will send a new board to customer and customer will return failed board to Atron. Customer pays for freight service in both directions. A PO # will be required in advance of sending new board to customer for the price of a new system. Upon receipt of the returned board, Atron will invoice customer in the amount of Atron's then fixed repair cost for the board. If the failed board is not received by Atron within 15 days after sending customer a new board, the PO is due and payable.

LIMITED WARRANTY

Atron Corporation warrants this product to be in good working order for a period of 1 year from the date of purchase from Atron or an authorized Atron dealer. Should this product fail to be in good working order at any time during this warranty period, Atron will, at its option repair or replace this product at no additional charge except as set forth below. Repair parts and replacement products will be furnished on an exchange basis and will be either reconditioned or new. All replaced parts and products become the property of Atron. This limited warranty does not include service to repair damage to the product resulting from accident, disaster, misuse, abuse, or non-Atron modifications of the product.

Limited warranty service may be obtained by delivering the product during the warranty period to Atron. If this product is delivered by mail, you agree to insure the product or assume the risk of loss or damage in transit, to prepay shipping charges to Atron and to insure the product is adequately packed.

ALL WARRANTIES FOR THIS PRODUCT, WHETHER EXPRESS OR IMPLIED, ARE LIMITED IN DURATION TO A PERIOD OF 90 DAYS FROM THE DATE OF PURCHASE, AND NO WARRANTIES, WHETHER EXPRESS OR IMPLIED, WILL APPLY AFTER THIS PERIOD.

ATRON HEREBY DISCLAIMS ALL OTHER EXPRESS AND IMPLIED WARRANTIES FOR THIS PRODUCT INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

IF THIS PRODUCT IS NOT IN GOOD WORKING ORDER AS WARRANTED ABOVE, YOUR SOLE REMEDY SHALL BE REPAIR OR REPLACEMENT AS PROVIDED ABOVE. IN NO EVENT WILL ATRON BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH PRODUCT.

SOFTWARE LICENSE AGREEMENT

All Atron software is protected by both United States Copyright Law and International Treaty provisions. Therefore, you must treat this software just like a book with the following exception: Atron Corp authorizes you to make archival copies of the software for the sole purpose of backing up your software and protecting your investment from loss.

This means that this software may be used by any number of people and may be freely moved from one computer location to another so long as there is no possibility of it being used at one location while it is being used at another - just like a book.