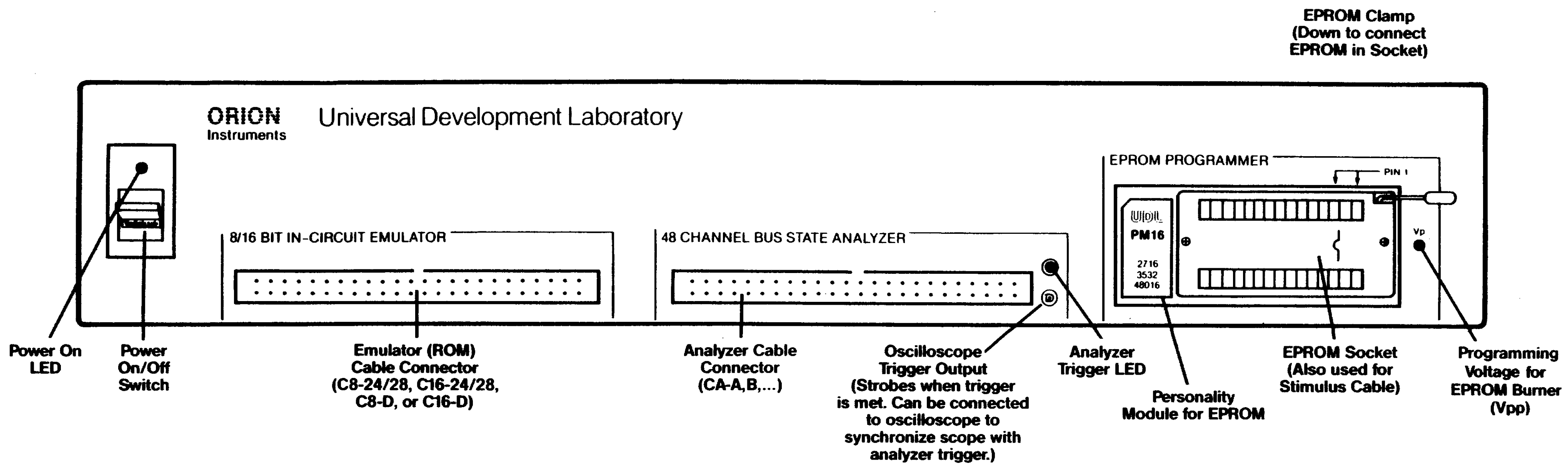
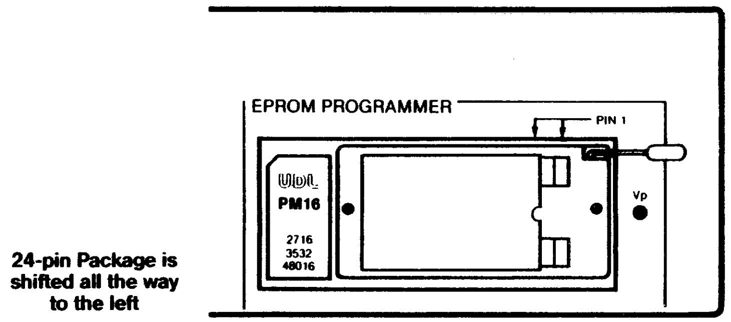
ORION
Instruments

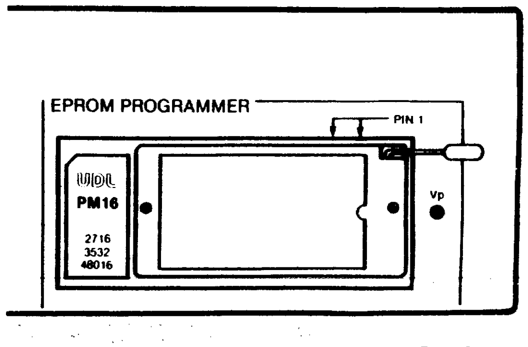
UniLab II™

Volume One
User's Guide

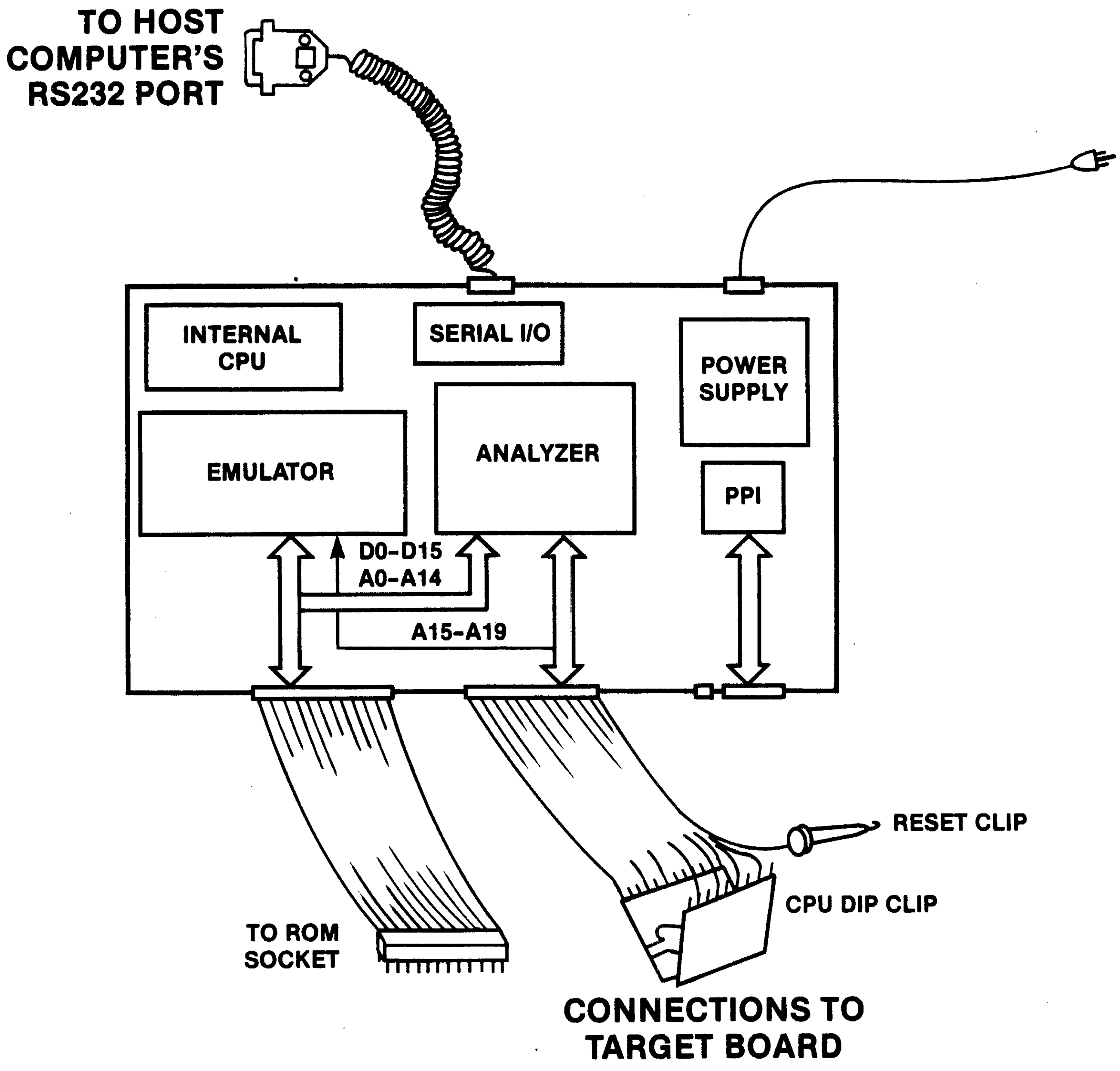

EPROM Clamp (Down to connect EPROM in Socket)
ORION Instruments
Universal Development Laboratory
EPROM PROGRAMMER
PIN 1
UDL
PM16
2716
3532
48016
Vp
8/16 BIT IN-CIRCUIT EMULATOR
48 CHANNEL BUS STATE ANALYZER
Power On LED
Power On/Off Switch
Emulator (ROM) Cable Connector (C8-24/28, C16-24/28, C8-D, or C16-D)
Analyzer Cable Connector (CA-A,B,...)
Oscilloscope Trigger Output (Strobes when trigger is met. Can be connected to oscilloscope to synchronize scope with analyzer trigger.)
Analyzer Trigger LED
Personality Module for EPROM
EPROM Socket (Also used for Stimulus Cable)
Programming Voltage for EPROM Burner (Vpp)



24 Pin EPROM in Programming Socket



28 Pin EPROM in Programming Socket



TO HOST COMPUTER'S RS232 PORT
INTERNAL CPU
SERIAL I/O
POWER SUPPLY
EMULATOR
ANALYZER
PPI
D0-D15
A0-A14
A15-A19
RESET CLIP
CPU DIP CLIP
TO ROM SOCKET
CONNECTIONS TO TARGET BOARD

Table of Contents
UniLab Manual

VOLUME ONE
UniLab User's Guide

VOLUME TWO
UniLab Reference Manual

The information in this reference manual applies to both the UniLab and the OptiLab.

Introduction: Getting Acquainted

Contents

The UniLab Method

Welcome to a new world of development systems.

The **UniLab II**^{tm} will change the way you develop software for your microprocessor projects. The UniLab does away with most of the guesswork and frustration associated with hunting for bugs in your code.

What Is the UniLab?

The UniLab (Universal Development Laboratory) is a personal microprocessor development system.

In one box, the UniLab includes all of the instruments needed for the development of microprocessor-based products. It transforms your personal computer into a complete workstation for prototyping, testing, and debugging.

The UniLab monitors the address, data and control signals on your microprocessor circuit board. This lets you see how the system responds to your programmed instructions.

You use the UniLab to tell your processor what you want it to do, and at the same time watch what it really does.

How the UniLab Works

The UniLab's emulation ROM feeds instructions to the processor while the bus state analyzer watches the effect of those instructions. When you use the UniLab, you watch your processor executing your code.

Your microprocessor stays in control of the board-- but the UniLab is in control of your processor.

How You Work with the UniLab

You conduct a dialogue with the UniLab. The topic of conversation is the system you are testing. You describe some condition that appears on the bus, and the UniLab replies with a display of the program execution.

Non-Intrusive Analysis

The UniLab can watch your target board's bus without interfering with your processor. Unlike other hardware test and diagnostic systems, the UniLab does not alter the flow of your program. This means that your processor continues to run after the UniLab has captured a trace of the program activity.

The UniLab can also stop your processor at a breakpoint, or with a hardware interrupt signal.

A New Way to Solve Your Problems

The UniLab is a step ahead of traditional techniques.

Older methods work best when you know the cause of the problem at the beginning. With those methods you could only look at the program's execution by stopping at specific code addresses. Since you usually don't start out knowing the cause of your problem, the older methods required that you spend a lot of time guessing.

With the UniLab, you describe the symptom of your bug. Then you can watch what the program does both before and after the symptom occurs. The symptom that you describe is called the "trigger specification." For example, you might start out by asking the UniLab to show you what happened just before your stack overflowed, or you might want to see each write into a particular string variable.

You start by asking general questions, and quickly zero in as the UniLab helps you track down the specific problem.

If you are used to single-stepping, or setting multiple breakpoints, you will appreciate the new, more powerful method of observing program flow with the UniLab's combination of emulator and bus state analyzer.

Breakpoints and Single-stepping

The UniLab's unique approach to problem solving emphasizes the actions of your processor. After all, what you really care about is getting results, not the contents of Register DX at step 235.

But sometimes, to get the job done, you do need to know the "internal state" of the processor. The UniLab's processor-specific DEBUG software lets you set breakpoints, look at and alter both internal registers and target RAM, and single-step through code.

To do this, the UniLab needs some of your target processor's resources. Usually all the UniLab needs is a working stack and one to four bytes of ROM. We call the required bytes of ROM "the reserved area." This area can be moved with the command **=OVERLAY.**

On many Disassembler/DEBUG packages the DEBUG needs to use a software interrupt vector on the target processor.

Hardware Interrupt feature

The DEBUG software, in combination with the UniLab hardware, provides an additional feature: immediate DEBUG control. The UniLab can interrupt the target processor and perform any of the DEBUG features at any time, without the need to set a breakpoint.

The UniLab system provides this feature by using an additional target system resource: either a non-maskable interrupt or the interrupt request. This feature can be easily disabled if desired.

Simulating Inputs

You use the Stimulus Generator to simulate inputs to your target board. You control the stimulus generator from the keyboard, and eliminate the need for the usual input switches on prototype boards.

Programming EPROMs and EEPROMs

When you have completed the design and testing of your software, you use the UniLab's built-in EPROM programmer to burn your code into an EPROM.

Automated production tester

Once you have a product ready to ship, you can use the UniLab macro capability to write test macros. A test macro can load routines into emulation memory, run them, and compare the traces to known good traces that were saved earlier. The UniLab can thus verify all the functions of your product.

From first prototype to final test

Since the UniLab runs properly when your target system has bus shorts, it can even help diagnose hardware problems (see the first several entries in the **TroubleShooting** Chapter).

So as you develop your product from the prototype phase to the production floor, the UniLab II will help you every step of the way.

Guide to the Documentation

Document conventions: special words

Orion's Emulation Module^tm^

An Emulation Module is a small board that plugs into the target system in place of the microprocessor. It makes the connection between the UniLab and the target system quick and easy.

Orion's MicroTargets^tm^

A MicroTarget is a minimally configured, expandable control board that you can use to verify your setup and to develop your software before your prototype hardware is ready.

Orion's DDBs

DDB refers to the Disassembler/DEBUG software packages that support specific processors.

DEBUG

DEBUG refers to the features of the DDB package that provide breakpoint displays, hardware interrupt, and access to target RAM and internal registers.

DEBUG control

You have DEBUG control when the processor is under the control of the UniLab idle hardware.

Trigger spec

The trigger specification (trigger spec) tells the UniLab when to freeze the trace buffer and send it to the host system for display. The trace buffer contains a record of the bus activity of your target system.

The minimal trigger spec describes only a set of bus cycles (the trigger cycle). The trigger spec can be as narrow as a single address, or so broad that every cycle matches. The trigger spec usually includes a delay as well-- a count of how many cycles to capture after the trigger cycle.

More advanced trigger specs produce filtered traces (for example, a trace that contains only the write cycles), or can include a sequence of qualifiers which must be seen before the trigger cycle is sought.

Orion's PPA

The Program Performance Analyzer^tm^ (PPA) produces tables and bar graphs of program activity. These displays help you debug and optimize your software.

Document conventions: typeface conventions

Commands always appear in **UPPER CASE, BOLD FACE** type. The UniLab software accepts commands in any mixture of upper and lower case.

The UniLab recognizes only hexadecimal numbers, unless you use either **D#** or **B#** in front of the number, for decimal and binary, respectively.

A description of a parameter is shown in lower case type inside <pointy brackets>. An example of a parameter is shown in the same type as the command.

The names of function and cursor keys are shown in **Bold Face** with initial capitals. For example: press function key nine (**F9**) to start the target program and capture a trace, press the **Home** key to see a trace display from the top.

On-line documentation

The word **HELP** is the key to on-line help.

Type **HELP** by itself to see a screenful of information on all the on-line help resources.

Type **HELP** followed by the name of a command to get the on-line glossary entry. Chapter Seven of the Reference Manual is a printed version of the information in the on-line glossary.

Printed documents

The User Manual and the associated Reference Manual are general purpose documents. The features they describe apply to every UniLab Disassembler/DEBUG (DDB) software package.

Your DDB software package is shipped with an additional document, a Target Application Note which contains information on package-specific features.

The UniLab Programmer's Guide, available from Orion, will be of value mainly to experienced users who wish to write sophisticated macros.

A growing list of Engineering Technical Notes (ETNs) published by Orion's Applications Engineering Department can help you get the most out of your UniLab. Many of these notes deal with microprocessor board design and the connection to the UniLab. Write to Orion to get the latest list.

Altogether, the UniLab documentation provides assistance from the installation of the software to the writing of automated test macros.

Where to go for which information

Volume One: User Manual

The User Manual contains the information needed to install and work with the UniLab software.

To get started on installation, turn to Chapter One.

Chapter Two contains a guided demonstration of the UniLab capabilities on a Z80 target system. You should look through this short chapter, after you have completed installation. Unguided sample sessions in the Target Application Notes show the same capabilities on other processors.

Chapter Three provides an exhaustive map of the menu system, a guide to the trigger specification commands, a description of the use of command tail and batch files, and a thorough tutorial on the special features.

Chapter Four describes the optional Program Performance Analyzer.

Chapter Five describes the on-line help.

Volume Two: Reference Manual

Consult the Reference Manual for in-depth information on the UniLab.

Chapter Six has detailed information on the UniLab software commands and displays. Turn to this chapter as needed, after consulting Chapter Three and the Target Application Note.

Chapter Seven is a printed version of the information in the on-line glossary.

Chapter Eight is a guide to troubleshooting. Turn here for help when you have problems with your target system or with the UniLab.

The appendices contain useful information on DDBs, macros, cross-assemblers and compilers, EPROMs and system messages.

Target Application Notes

These Notes are guides to the UniLab trace and DEBUG features on each specific processor. A Target Application Note is shipped with the DDB software. Each Note includes:

1) a list of the processors supported by that package,
2) a diagram of the connection between the UniLab and the target microprocessor(s),
3) the instruction set recognized by the line-by-line assembler,
4) a guide to the DEBUG capabilities and an overview of the specific features of the disassembler/DEBUG package,
5) a sample session which includes both trace and breakpoint displays, and
6) a glossary of the DEBUG commands supported by that DDB package.

ORION Programmer's Guide

A guide to macros and some of the internal words of the UniLab. Includes sample macros and definitions of some of the internal words available in the MACRO system. Can be ordered from Orion.

This document includes:

file and editor commands,
direct access to the UniLab trace buffer,
UniLab string package, and
examples of automated test routines.

Engineering Technical Notes

Short notes on microprocessor control board design issues and solutions. Write to Orion to get the latest list of notes or to order a specific note.

Chapter One: Installing The UniLab

Contents

Equipment Requirements

All the discussions that follow assume that your host computer is:

a PC, or XT, or AT compatible machine,
with at least 320K of RAM,
with a standard serial port,
running DOS version 2.0 or higher.

Unless you have this equipment, you will not be able to run the UniLab software. If you know that your system meets these requirements, skip to the next page.

Available RAM

During "boot up" most PC compatible machines tell you how much RAM they have. If you have any doubts, you can use the DOS **CHKDSK** utility (CHecK DiSK). This utility was designed to examine your disk-- but it also reports the status of your RAM; after it reports the status of your disk. To invoke it, type **CHKDSK** at the DOS prompt.

If you do not have enough RAM to run the UniLab software, you will need to purchase either additional RAM chips or a memory expansion board for your computer.

Serial port

To connect to the UniLab, your computer must have a 25-pin serial port connection, or, for ATs, a 9-pin connection and a short 9-to-25 adapter cable.

Serial port boards are inexpensive and widely available.

DOS Version

Type **VER** at a DOS prompt (**A>** or **B>** or **C>**) to find out what version you are using.

Your operating system will respond with a short message that includes the version number, such as:

```
XXX Personal Computer DOS Version  3.10
```

If the version is lower than 2.0, you will not be able to run the UniLab software until you get a more recent version of DOS. If your operating system does not recognize the command **VER**, then either you have a very old version of DOS, or you have some other operating system.

Processor-specific Configuration

As part of the installation of your UniLab you will need to connect the Orion hardware to your target system. Orion offers a line of Personality Paks (PPAKs), which include all the connections, equipment, accessories and software you will need.

Personality Paks

The Orion Personality Pak includes additional documentation, which supplements the User's Manual.

In some cases, the PPAK includes installation information which replaces the instructions in step three of the process described in this chapter.

Some Personality Paks include Emulation Modules^{tm} and MicroTargets^{tm}. An Emulation Module plugs into your target board, in place of the microprocessor. It simplifies the task of connecting the UniLab to your target system.

Orion MicroTargets are expandable target boards, which allow you to start running your code on known good hardware before your own target system is running.

Consult your Orion Sales Representative for current product availability and pricing.

16-bit Systems

If your target system has a 16-bit data bus, you follow the installation procedure described in this chapter, but you must use a 16-bit emulator cable, as illustrated on page 1-??. These cables have two separate ROM plugs, one of which must be plugged into an odd byte ROM socket and the other into an even byte ROM socket.

Either the odd or the even byte can be the most significant byte of the 16-bit word, depending upon whether your processor follows the Intel or Motorola model of memory organization.

Basic Information...

...About the UniLab Hardware and Software

Hardware controls

You control the UniLab with a program that runs on your personal computer. The instrument itself has only two physical switches--

the on/off switch on the front panel,
and a baud rate selector inside the instrument.

When you turn on the UniLab, the light above the switch goes on. Then the Unilab is ready to accept an initialization command from the host computer.

The instrument is shipped with the baud rate already set properly. If you want to change the baud rate, see the Special Note on baud rate at the end of this chapter.

Distribution diskette

All of your interactions with the UniLab will be done through a program called **ULxx.EXE**, which runs on your host computer under the DOS operating system. Your Disassembler/DEBUG (DDB) software has already been installed as part of this file. To distinguish among DDBs, each **.EXE** file has a distinctive name, based on the processor name, such as **UL48**, **UL88**, etc.

The **.VIR**, **.OVL** files included on your distribution diskette contain additional executable code. The **.OPR** files are necessary to run an operator system, the **.MCR** files to run a macro system. See the Special Note on Operator and Macro Systems at the end of this chapter.

The **UNILAB.SCR** file contains some message text.

You need all of these files for the UniLab system to operate. If you have more than one Disassembler/DEBUG package, they can both reside on the same directory without any problem, as long as they have the same release number (that is, 3.30 or 3.12).

Your distribution diskette also contains **AUTOEXEC.BAT**, **CONFIG.SYS** and **INSTALL.BAT** files, which are described in section two of the installation procedure, and **.BIN** and **.TRC** files, described in appendix J.

Glossary diskette

A separate diskette contains the on-line glossary and it's supporting files.

...About Installation

Quick instructions, for the experienced. . . .

If you can get the installation done with a minimum of guidance, you will want to follow the **Quick Step by Step.**

And detailed instructions, to help you learn as you install

The **Detailed Step by Step** contains thorough instructions and longer explanations.

The best of both worlds

The parallel organization of the two sets of instructions allows you to dip into the detailed procedure while following the quick procedure. Entries in the **Quick Step by Step** refer you to the appropriate pages in the **Detailed** writeup.

For example, most people will want to read the **Detailed** description of how to connect the **RES-** wire to their target board.

Overview of the installation process

You go through a four-step installation process the first time you get the UniLab ready for work:

1. Connect the UniLab to the serial port on your computer.
2. Install the UniLab software.
3. Connect your UniLab to the target system.
4. Test and verify your connection

Once the UniLab has been connected and verified, you will never need to go through this process again.

...About Test and Verification of Your Setup

Test procedure

After you have connected the UniLab to your target board and installed the software, you should verify the connection using the simple test program included with your Disassembler/DEBUG (DDB) software. The verification procedure confirms that you have correctly connected the UniLab to a functional target board.

Load the test program into emulation memory with the command **LTARG.** You should get a trace of that program, then use both breakpoints (**RB**) and the hardware interrupt (**NMI**) to establish DEBUG control.

For a complete guide to verification, turn to section four of the **Detailed** installation. Below is an introduction to the verification procedure.

Introduction to verification

You can choose between two methods of verifying your trace display. You can compare your trace with the printout which appears in the separate Target Application Note included with your DDB software. Or you can use a UniLab command to compare your trace with the one stored in a file on your distribution diskette.

You might need to alter the **LTARG** program to reflect the memory map of your target board-- especially the location of the stack pointer. If you do so, your trace will be slightly different from the standard one.

The breakpoint display you will compare with the printout in the Orion documentation.

If both the trace compare and the breakpoint display are accurate you can be certain that you have connected everything properly, and that you have a working target board.

...About Additional Documentation

Connection information

With either a Personality Pak or an Emulation Module, you receive specific connection instructions, which may replace some of the information in this chapter.

You will find a connection diagram in the Target Application Note for your processor. Appendix C is a table of the connections for all microprocessors.

If you need to build a customized cable, turn to Appendix D.

Information on UniLab software

For an brief introduction to the general capabilities of the UniLab, turn to the next chapter, **Guided Demo,** and to the command reference card.

For the authoritative guide to the processor specific features of your DDB software, turn to your processor-specific Target Application Note. That document also makes note of any configuration commands that you must include in your target program for the DEBUG features.

Chapter three, **Operation,** gives you an overview of the UniLab's features. The chapter after that, **Performance Analysis,** covers the tools for analyzing and optimizing your program.

As you use the UniLab software, you will appreciate the on-line help facilities. These are covered in the last chapter in volume one, **On-Line Help.**

UniLab Reference Manual

Turn to volume two, the UniLab Reference Manual, for more information. The **Operations In Detail** chapter thoroughly covers the use of the features which were introduced in volume one.

The **Command Reference** chapter gives you a definition, explanation and example for every UniLab command. The on-line version of this information, available by typing
HELP <command name>, is updated more frequently than the printed Reference Manual.

...About TroubleShooting

Difficulties with UniLab software

The very first resource to turn to is the on-line help facilities. Chapter Five covers the on-line help thoroughly. Press **F1** to get either the general help screen or a context dependent help screen. Use **HELP** <command> to get the on-line glossary entry for a particular command.

If you do not understand an error or status message, you will want to consult Appendix I.

Trouble with target system

The **TroubleShooting** chapter in the Reference Manual starts with a list of symptoms. The **Solutions in Depth** section of that chapter then helps you solve your problems.

Additional Useful Information

Who is hosting this party? What all the names mean

The UniLab receives all commands from your personal computer-- the host. The UniLab software resides on the host. A little bit of code resides in the UniLab's ROM. The UniLab receives instructions from the host, and sends information to it, over a serial port.

The UniLab, in turn, controls the target board. The target board is the microprocessor control system that you are developing-- or the one that Orion sent you as part of the Personality Pak. In either case, the UniLab's emulation memory contains the program that runs the target board.

How they all talk to one another: UniLab & host

The host talks to the UniLab through the RS-232 interface at 19,200 baud. We achieve this high speed by talking directly to your serial communications chip.

The serial port is rated at only 9600 baud, the highest speed you can achieve when using DOS calls for serial communications. The higher speed used by the UniLab does not harm your serial port in any way. If you wish to change the baud rate, turn to the special note on baud rate at the end of this chapter.

After the UniLab gets an instruction, it often performs actions without needing to talk to the host again. When the UniLab needs to send information back to the host it uses the same RS-232 interface.

But the speed of most UniLab operations does not depend on the speed of the serial interface. The rate of serial data transfer will make you wait only when you load or save large programs.

Serial port of AT

The UniLab plugs into a standard 25 pin serial port, not the 9 pin port of the AT. If you have an AT or AT compatible you must put a 9 to 25 pin adapter on the serial port of your computer.

How they all talk to one another: UniLab & target

The UniLab communicates with the target board through two fifty-pin parallel connectors on the front of the UniLab, which carry signals back and forth as sketched out below.

UniLab inputs

The bus state analyzer of the UniLab has 48 input bits. The addresses take up to twenty bits, the data takes eight or sixteen, and the control lines take another four. Between eight and twenty lines are left over for miscellaneous uses, to be chosen by you.

The UniLab also has four clock inputs from the target board: **K2-**, **K1-**, **RD-**, and **WR-**. The UniLab uses these inputs to determine whether the processor is working, and to clock information into the trace buffer.

UniLab outputs

The emulator of the UniLab looks at the twenty bit address and the bus control signals, and responds with eight- or 16-bit data when the processor tries to fetch from an emulated address. Emulation ROM may respond to a read signal as well, depending on the bus architecture of the microprocessor. Your microprocessor will not be able to write into emulation memory.

The UniLab also sends a RESET signal out on the wire labeled **RES-**. This wire often requires special connection.

When you use certain DEBUG routines, a signal is sent out on the wire labeled **NMI-** (Non-Maskable Interrupt), to either the NMI pin of the processor, or to the IRQ (Interrupt ReQuest) pin if the processor lacks an NMI. This wire sometimes requires special handling.

Quick Step-by-Step

1. **Connect the UniLab to Host** (page 1-13)
   Connect the UniLab to your host computer, and turn it on.

2. **Software Installation** (page 1-16)
   On hard disk systems:
   Use the command **INSTALL** to move the UniLab software onto your hard disk. Change or create a CONFIG.SYS file in your hard disk's root directory, so that it contains the settings in the sample CONFIG.SYS file on your distribution diskette. Copy the glossary files from the second diskette to the directory **C:\ORION.**
   On floppy disk systems:
   Copy all the files from your UniLab distribution diskette to a "bootable" diskette.
   On both hard and floppy disk systems:
   Reboot your system and start up the UniLab program..

3. **Connect the UniLab to Target Board** (page 1-25)
   You can keep the UniLab turned on while you connect it to the target board, but we recommend that you turn off the power supply to the target. You must turn off the power to your system if it runs on anything but +5 voltage.

   Remove any ROMs to be emulated from their sockets on your board, and then put ROM plug of the emulator cable into a vacant socket. Connect the other end of the cable to the emulator socket of the UniLab.

   Put the DIP clip onto your microprocessor and connect the wires from the emulator cable and the analyzer cable to the correct processor pins. See the Target Application Note for your DDB, or use the **PINOUT** command. Connect the other end of the analyzer cable to the analyzer socket of the UniLab.

   Connect the RES- wire to the proper place in the reset circuit. See pages 1-34 to 1-36 for details.
   Turn on the power supply to your target board.

4. **Verify Your Setup** (page 1-40)
   Follow the verification procedure described on page 42, or the minimal test described here: get into the UniLab program and press **F10** to enter the menu mode. Select the "LOAD OR SAVE A PROGRAM" sub-menu with **F2**, then choose the "LOAD A SAMPLE PROGRAM" option, **F4**. Then return you to the Main Menu with **F10**, and select the "USE THE ANALYZER" sub-menu, **F4**. Select the "RESET AND TRACE FIRST CYCLES," **F1**.

   After you've verified the connection, you can turn to page 2-1 to learn more about the UniLab.
   When you want to exit, type in the command **BYE**.

Detailed Step by Step

Use this section to get more information than you can find in the very sparse **Quick Step-by-Step.** The numbered headings follow the outline of the **Quick** section, but here you get several pages for each numbered heading.

Each numbered task has been broken down into several subtasks. Some troubleshooting help appears here, but if you really run into difficulties you should consult the **TroubleShooting** chapter in the Reference Manual.

1. Connect the UniLab to Host

Overview

The first thing you will want to do is connect the UniLab to your personal computer.

Once it has been connected, you will never need to disconnect it, unless you have to use the serial port for some other instrument or device. The UniLab will not interfere with the proper functioning of your other personal computer software.

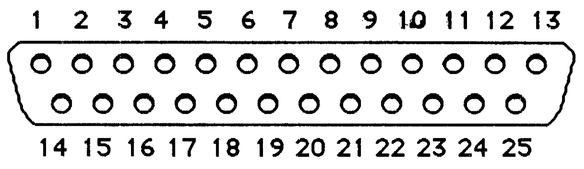
You could go directly to step two and install your software without the UniLab attached to the host computer. However, you would not be able to use the program, since the first thing it does is send a message to the UniLab and wait for a reply.

When you do start up your software, the UniLab must be properly connected and turned on-- otherwise the software will freeze up (hit **CTRL-BREAK** to unfreeze). So we recommend that you first connect the UniLab, then install your software.

Find the correct port

The UniLab usually talks with the host through communications port one (COM1)-- the usual default serial communications port. In general COM1 will either be the only serial port on your system or be one of two ports. Look for the male DB-25 connector on the back or side of your computer. The female DB-25 connector is for a parallel printer.

DB-25 Connector



Serial port of AT

If you have an AT or AT compatible, with a 9 pin serial port, you will need a standard 9 to 25 pin adapter.

Connect the cable

After you've found the serial port, plug in the RS-232 cable from the UniLab.

Turn on the UniLab

Then plug the UniLab into an electrical outlet, and turn it on. Your UniLab is now connected and ready for the software.

Trouble?

Can't find any port

Look at the sides, the back, and the bottom of the computer. Some even have them hidden behind removable panels (especially some of the popular lap-tops). Some computers are sold without a serial port as standard equipment.

If you do not have a serial port, you will have to acquire and install a serial port board before you can continue.

Found more than one port

You might have more than one RS-232 standard serial port. If they both are female (that is, they both accept the host end of the UniLab-to-host cable), then you have several choices. You could check the manual for your computer, or look at its internal connections-- but there is a much simpler approach:

> Choose one of the ports, and plug the UniLab into it. Later in this process, after you have the software installed, start it up. If it doesn't freeze-up, then you have the UniLab connected to the correct port.
>
> If it does freeze-up immediately after displaying "Initializing UniLab," then use **CTRL-BREAK** (tap the break key while holding down the control key) to break out of the freeze. Plug the UniLab into the other port, and type **INIT.** The program should initialize the UniLab without freezing up.
>
> If the program freezes with the UniLab plugged into either port, consult the **TroubleShooting** chapter.

2. Software Installation

- Install the software
  - On a hard disk
  - On a floppy disk computer
- Reboot your computer
- Start up the UniLab program
- Trouble?

Install the software

If you have a hard disk on your computer, you probably will want to use the INSTALL.BAT file to install the UniLab software on your hard disk. You can, if you wish, perform the installation tasks yourself.

If you have only floppy drives on your computer, you will need to copy the UniLab software onto a "bootable" DOS diskette.

On a hard disk

Explanation

The INSTALL batch file makes a directory called **\ORION** on your hard disk and copies to that directory all the files on the distribution diskette.

INSTALL.BAT also adds two lines to your AUTOEXEC.BAT file:

```
set ORION=C:\ORION
set GLOSSARY=C:\ORION
```

which set up two "environment strings" that the UniLab software requires. The first string tells the UniLab program where to look for various overlay files, the second tells the program where to look for the on-line glossary files.

You will need to copy the files on the glossary disk by yourself, or the commands **WORDS** and **HELP** <command> will not work.

You also must create or alter your CONFIG.SYS file so that DOS allows the UniLab software to have 16 files open and use 10 buffers. A sample CONFIG.SYS file is included on your distribution diskette. The CONFIG.SYS file must reside in your root directory (**C:\**). It should contain these two lines:

```
files=16
buffers=10
```

The new settings in CONFIG.SYS do not take effect until you reboot your computer. If these system variables are already set to higher values, that is fine. However, if "files" is set too low then some UniLab features will not work and if "buffers" is too small then many UniLab commands will run slowly.

Procedure

Put the master UniLab diskette into floppy drive A. Execute the INSTALL batch file by typing

A:INSTALL

You will also need to copy the files from the second diskette to your **C:\ORION** directory and create or alter the CONFIG.SYS file in your root directory (**C:\**).

On a floppy disk computer

Explanation

Before you can use all the features of the UniLab software you need to set up two "environment strings" and two system variables. The sample AUTOEXEC.BAT and CONFIG.SYS files on your distribution diskette will, when put onto a bootable diskette, set up your floppy disk system.

Make a new "bootable diskette" and use it to boot up your system, so that the environment gets set up properly for your new software. You can copy onto the bootable diskette either the entire UniLab system or just the CONFIG.SYS and AUTOEXEC.BAT files.

The AUTOEXEC.BAT included on the distribution diskette sets the environment string "ORION" to the correct value for running the UniLab software from floppy disk drive A.

```
set ORION=A:\
set GLOSSARY=B:\
```

The first string tells the UniLab program where to look for various overlay files, the second tells the program where to look for the on-line glossary file.

If you wanted to run the UniLab software from drive B you would have to change the AUTOEXEC.BAT file to say:

```
set ORION=B:\
```

The CONFIG.SYS file included on the distribution diskette tells DOS to allow any piece of software to have a maximum of 16 files open and use 10 buffers. It contains these two lines:

```
files=16
buffers=10
```

The new settings in CONFIG.SYS do not take effect until you reboot your computer. If these system variables are already set to higher values, that is fine. However, if "files" is set too low then some UniLab features will not work and if "buffers" is too small then many UniLab commands will run slowly.

On a floppy disk computer (continued)

Procedure

Put your DOS master diskette in drive A. Put a new blank diskette in drive B and format it as a "system" diskette with the DOS command:

```
FORMAT /S B:
```

After you have formatted the diskette, take your DOS master out of drive A, and replace it with the UniLab distribution diskette. Copy all the UniLab files to the newly formatted diskette with the DOS command:

```
COPY A:*.* B:
```

Or, copy only the AUTOEXEC.BAT and CONFIG.SYS files onto the newly formatted diskette. Each time you use your UniLab software, you should make certain that the computer has been booted up from a diskette with the correct CONFIG.SYS file on it.

Reboot your computer

<u>Explanation</u>

The settings of the system constants "files" and "buffers" can <u>only</u> be changed by rebooting the system.

The lines in AUTOEXEC.BAT are not as vital-- you can set or change the value of the variable "ORION" at any time, by typing in from your keyboard:

SET ORION=<path name>

where <path name> is any valid DOS path description, such as **C:\ORION** or **C:\ASM\DDB**. You can change the value of "GLOSSARY" in the same way. Of course, you will want to change the setting of these two variables only if you actually move the UniLab files to a different directory.

The **GLOSSARY** variable tells the program where to look for the on-line glossary and its associated files. The UniLab software needs the glossary only when you use either **HELP** or **WORDS**. On a floppy drive system, you have to put the glossary diskette in drive B: when you want to use the on-line glossary. With a hard disk, you will want to copy the files from the glossary diskette to the **\ORION** directory on drive **C:**.

<u>Reboot Procedure</u>

With a hard disk computer:

Hold down the **CTRL** and **ALT** keys, and tap the **DEL** key.

Or, turn the power off and back on again. On some computers you must wait half a minute before turning the power back on.

With a floppy disk system:

First put your new bootable diskette in drive A.

Then you can reboot the same way that you do with the hard disk (see above).

Start up the UniLab program

The diskette you received contains eight or more files. Though you need all of the **.SCR, .VIR, .OVL, .OPR** and **.MCR** files for the software to run properly, you only call one file by name-- the command file, which ends in **.EXE**.

Use the DOS command **DIR** to get a listing of all your UniLab files. You will see one with **.EXE** at the end of its name.

Hard disk:	**DIR C:\ORION**
Floppy disk:	**DIR A:**

Start up the program by typing in the name of the command file.

First actions

The first thing the program does is spend a second or two loading itself from disk. Then it will display the welcoming help screen, reproduced on the next page. If your software package supports several related processors, you will be presented with a menu of processors during this startup sequence.

After it has displayed that screen, it sends an initialization message to the UniLab and waits for a reply from the UniLab. When the program receives a reply from the UniLab it displays the message "Initialized," then determines and displays the size of your UniLab's emulation memory (8K, 32K, 64K, or 128K).

You, and the program, are now ready to get started.

Welcoming help screen:

```
                      UniLab              Copyright 198X
                        II                Orion Instruments
                   Version X.XXX          Redwood City, CA


 XXXXX disassembler installed - with DEBUG.
        ________________________________________________________________
          HELP is available on-line by entering HELP or F1.
          Enter  HELP command  to see the definition of "command."
          Type  WORDS command  to see a list of commands.
          ____________________________________________________________
          Use the function key F10 for MENU mode operation and quick
            access to most common commands.
          More help is available on the Ctrl-F1 key.
          Press Ctrl-F10 for display of cursor key functions.
          ____________________________________________________________
          Type MESSAGE for current messages.
Initializing UniLab...
  Initialized   32K Emulation Memory
```

NOTE: Type in **MESSAGE** to get information about the most recent additions and updates.

Trouble?

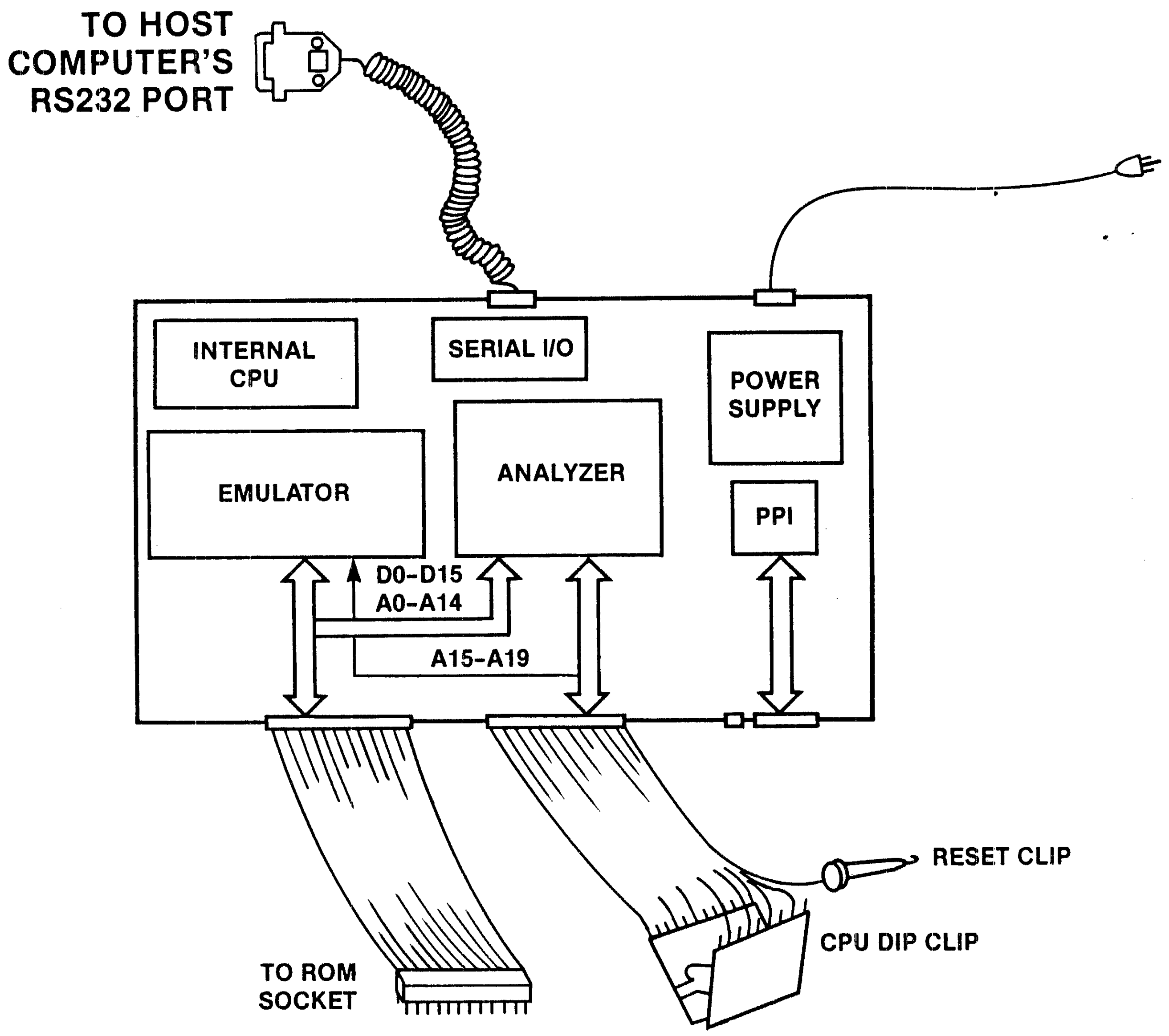
If the UniLab does not respond, you will see the program freeze-up after printing the "**Initializing UniLab...**" message on the screen. If the response from the UniLab was somehow garbled, you will see a "**RS-232 error #**xx" message. See below and in the **TroubleShooting** chapter.

If the program does not get a response from the UniLab, you will have to press the **CONTROL** and **BREAK** keys at the same time.

You probably have the UniLab plugged into the wrong port. Plug the RS-232 cable into the alternative port, and use the UniLab command **INIT** to send the initializing message to the instrument again. If the program again freezes, consult the **TroubleShooting** chapter.

If you get an "**RS-232 Error #**xx" then you probably have a background task running, such as a printer spooler, or have a "bus contention problem" on the serial ports of your computer. Both problems can be quickly solved. See the **TroubleShooting** chapter.

Insert picture of UniLab from frontispiece of Manual.



CONNECTIONS TO TARGET BOARD

3. Connect the UniLab to the Target Board

Overview

The UniLab controls and monitors the target board through the cables that connect to the two fifty-pin connectors on the front of the UniLab, between the power switch on the left side and the EPROM programmer socket on the right.

The emulator cable plugs into the socket on the left. This cable carries the data signals from the UniLab to the board, and the address signals from the board back to the UniLab.

The analyzer cable plugs into the socket on the right, and carries control signals back and forth. The analyzer cable also picks up some of the address signals.

Personality Paks and Emulation Modules

If you have a Personality Pak or an Emulation Module, you should consult the documentation which accompanies those products for any additional information about the connecting the UniLab to your target system.

All about cables

Make certain that you have the proper cable for your microprocessor. Most analyzer cables support several different processors. The cables are labeled with a letter, which should match the letter on the pinout diagram in the Target Application Note for your processor.

You can also get a cabling diagram on the screen with the **PINOUT** command. This on-line diagram usually shows only the main processor supported by the software package.

When you connect to your target board, you will be hooking up to your board at three different places:

1) at a ROM socket (with the ROM connector plug),
2) at the microprocessor (with the DIP Clip),
3) and at the reset circuit (with the RES- wire).

With some processors, you will also need to make a fourth connection for the NMI- wire from the UniLab.

ROM Connector

Most of the wires from the outlet on the left side, labeled **8/16 BIT IN-CIRCUIT EMULATOR,** go to a ROM connector. This connector plugs into the target board, occupying a ROM socket. Here the UniLab picks up data signals and address inputs. You must orient the plug properly.

DIP Clip

The remaining wires from the **EMULATOR** cable go to the DIP clip, along with most of the wires from the right side outlet--labeled **48 CHANNEL BUS STATE ANALYZER.** The DIP clip physically clips onto the microprocessor.

At the processor, the UniLab picks up and asserts control signals. It also picks up the microprocessor clock signals and, often, the upper bits of the address.

Reset wire and the NMI wire complications

These two wires carry output signals from the UniLab to the target processor. The reset signal causes your processor to start executing the target program from the beginning. The NMI signal tells your processor to jump to a hardware interrupt vector.

Both these wires would be connected directly to the pins of your processor, in the best of all possible worlds. However, the real world is not always that simple.

Complication explanation

These two UniLab output signals are "open collector" (resistor-transistor logic-- RTL), which means that the signal is not strong enough to pull down the output of a logic element (transistor-transistor logic-- TTL).

These two signals are inputs to a "logic element" -- your processor. However, these input pins of your processor are often driven by the (TTL) outputs of other chips (logic elements) on your target board.

If the pin of your processor that you are trying to connect the UniLab's wire to is driven by an external logic gate, you might have to:

> disconnect your processor's pin from the circuit that drives it,
> connect the pin to a "pullup resistor,"
> and then connect the UniLab wire directly to your processor's pin;

OR

> disconnect from the circuit the pin that drives your microprocessor's pin,
> and then connect the pullup resistor and the UniLab wire to the processor pin;

OR

> connect the UniLab wire to the input of the chip whose output drives your processor's pin.

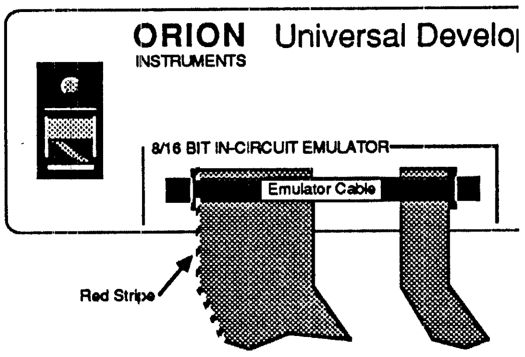
OR

Depending upon the unique configuration of your target board, you might have to do something else.

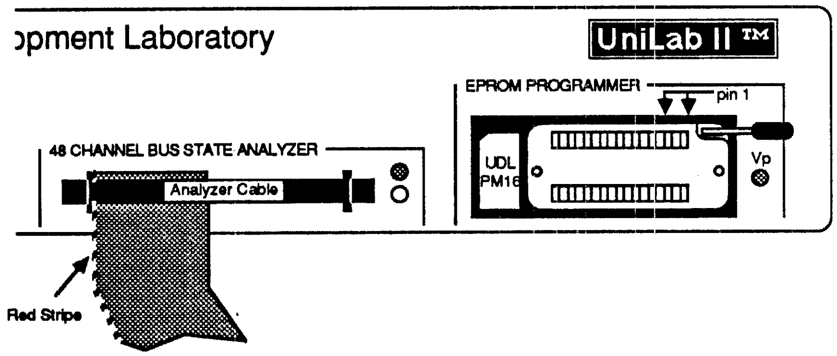
Plug cables into UniLab connectors

Plug the 50-pin cable labeled "Emulator" into the left socket on the UniLab, plug the "Analyzer" cable into the right socket.

Both connectors must be plugged in with the plastic key on the upper surface, and the red edge of the cable to the left.



Emulator Cable



Analyzer Cable

Take PROMs off board

The UniLab emulates the Read Only Memory (ROM) of the target board. To avoid bus contention problems you should remove the ROMs that you will emulate. Note that you can run a program partly out of emulation ROM and partly out of chips.

To avoid damage to components, turn off the power to your target system before removing chips.

Exception: run program from PROM

You will keep the ROM chips on your target board when you want to watch the execution of a program running from the chip. Before you can execute a program completely from chips, make certain that you disable the emulation ROM with **EMCLR**. .

EMCLR will also disable all the DEBUG features. Later, after you re-enable emulation you will want to re-enable the DEBUG features as well. Use the Mode Panel (press **F8**) to re-enable option **SWI VECTOR** on mode panel three.

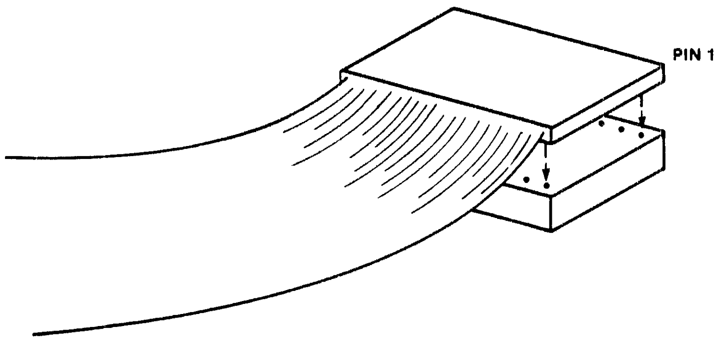
Alternative to running the program from ROM

When you want to run the program that resides in a ROM, you could instead read the program from the chip into the UniLab's emulation memory, using the **PROM READER MENU.**

Put ROM cable in ROM socket

The ROM plug on the emulator cable goes into any ROM socket in the target system. A single connection allows you to emulate several ROMs, but all ROMs that are to be emulated must be removed from their sockets.

Put the ROM plug into a vacant ROM socket, preferably the one the microprocessor reads from on reset.

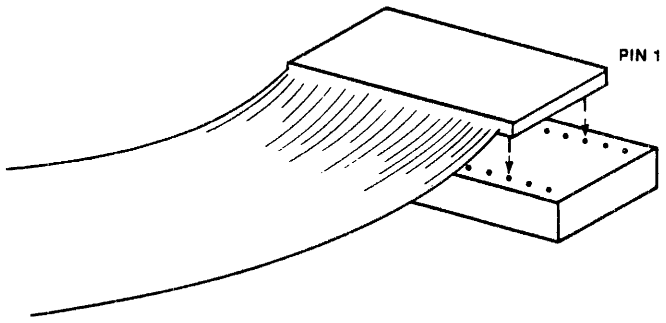


24 Pin ROM Plug in 24 Pin Socket

24-pin ROM plugs go into 24-pin ROM sockets. And 28-pin plugs (not pictured) are suitable for 28-pin sockets.

24-pin Plug in 28-pin Socket

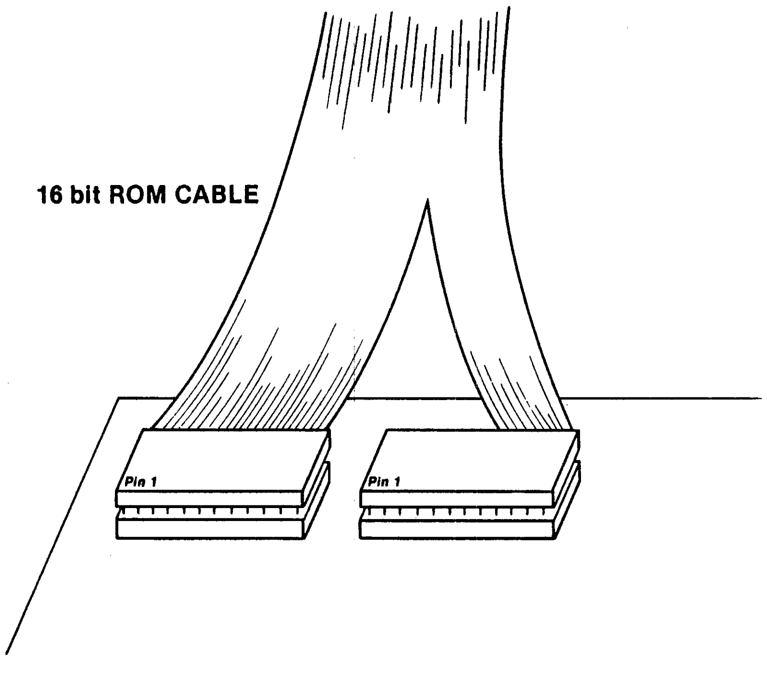
A 24-pin ROM plug can also go into a 28-pin socket, if pin one of the cable goes into pin three of the ROM it replaces. That will leave four unfilled positions on the socket, 1 & 2 and 27 & 28.



24 Pin ROM Plug in 28 Pin Socket

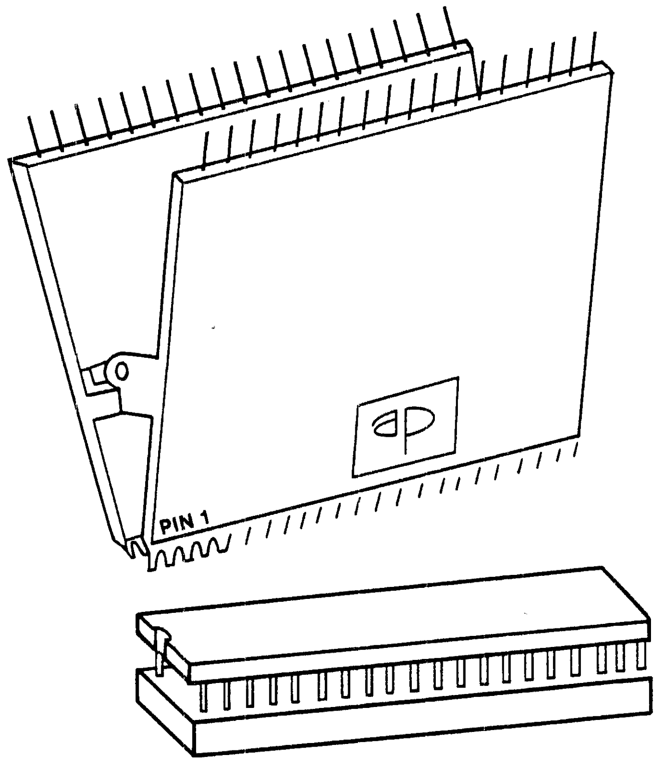
16-bit ROM Plugs

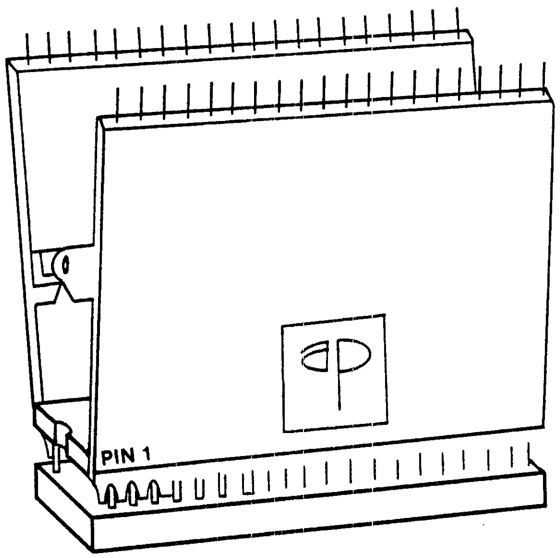
The 16-bit ROM cables are a pair of ROM plugs on a single cable. One goes into an odd byte plug, the other goes into an even byte plug.



Put DIP clip onto microprocessor

With the power to the target system still turned off, put the DIP clip onto the microprocessor, being sure to orient pin one of the DIP clip with pin one of the microprocessor.

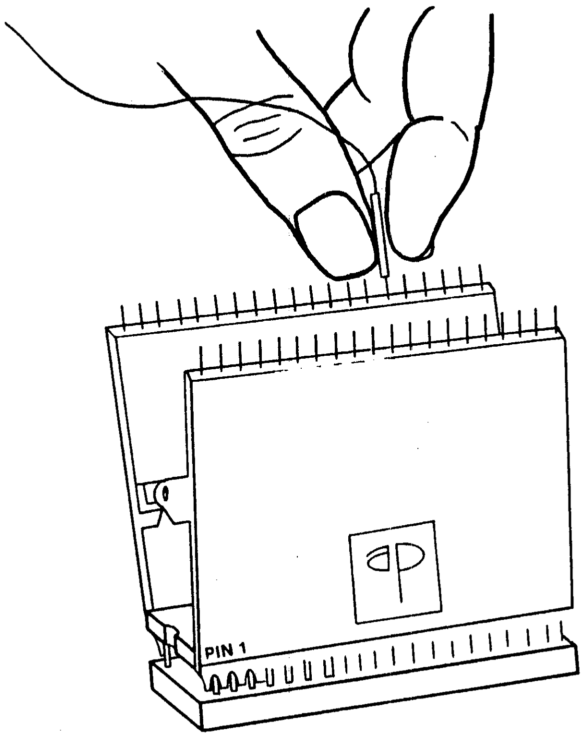




Attach proper wires to the clip

Connect the proper wires from the emulator cable and from the analyzer cable to the pins on the 40-pin DIP clip provided. The connection diagram is in the Target Application Note for your processor-- or type **PINOUT** to get an on-line diagram.

Double-check your wiring to the DIP clip, since this is a vital link in the connection.



Making Connections from Analyzer and Emulator Cables

Attach the RES- wire

The RES- wire carries the reset strobe from the UniLab to your target board. This signal causes the target board to start the target program from the beginning.

Connection to Simple Circuits

If you have a simple RC (Resistance-Capacitance) network attached to the reset pin of your microprocessor, then you can connect the RES- wire directly to the reset pin, and move on to connecting the NMI- wire.

Common Complication

However, many boards have a logic element in the reset circuit, an SSI or MSI chip that drives the reset pin, and gets driven in turn by a simple RC network.

As noted in the overview a few pages back, an "open collector" (RTL) circuit in the UniLab generates the signal on the RES- wire, so you cannot use it to "pull down" the output of a logic gate (a TTL or "totem pole" output).

Common Solution

The UniLab's RES- output can pull down the <u>input</u> of the logic gate. By controlling the logic gate, the UniLab controls the RESET pin of your microprocessor.

In general, you will find it easiest to clip the RES- wire onto the positive side of the capacitor in the RC circuit. See the diagrams on following pages that show typical reset circuit connections for an INTEL processor and for the Z80.

If you have trouble finding the capacitor, try asking the board designer, if you can find him. It might be easier to find and read the schematics. If neither personnel nor diagrams can be found, then you might have to trace the circuit.

Less Common Solution

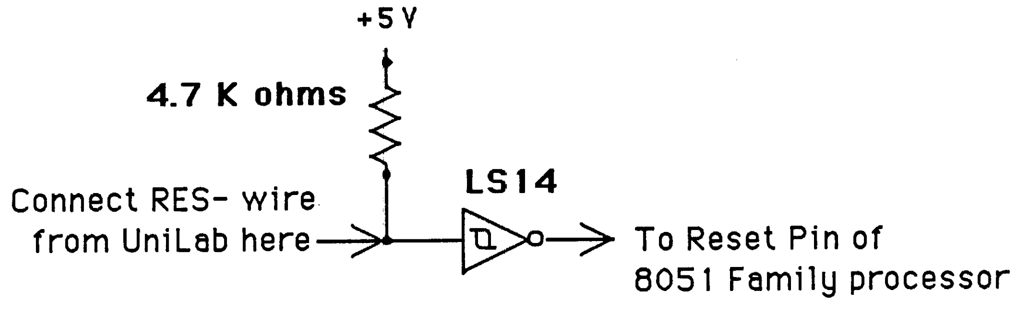
Sometimes the common solution will not work, and you will have to alter your reset circuit. You might have to remove the chip that drives the reset circuit, and connect a pullup resistor in its place.

8051 Family Complication

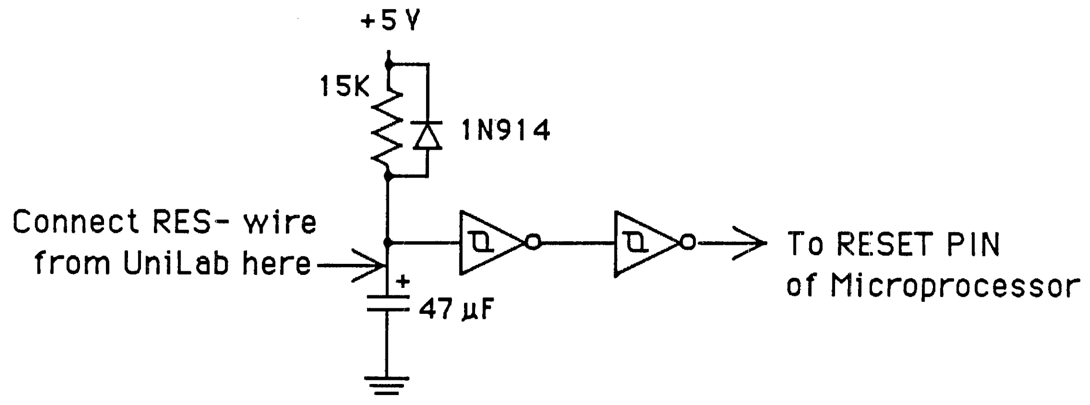
If your processor is in the **8051 family,** the RESET pin of your processor requires a positive going signal. You will have to feed the UniLab's negative going signal through a special inverting circuit, such as the one below.

The members of the **8051 family:**

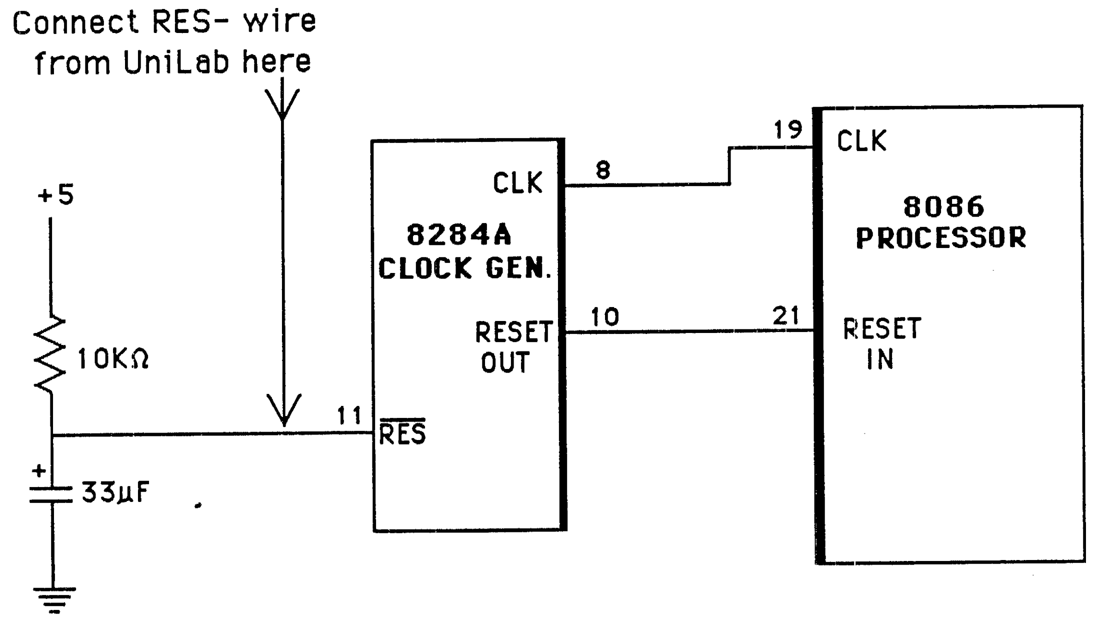
8051, 8052, 8031, 8032 and the piggyback **8051.**


+5 V
4.7 K ohms
LS14
Connect RES- wire from UniLab here
To Reset Pin of 8051 Family processor

Typical Reset Circuit necessary for 8051 family processor


+5 V
15K
1N914
Connect RES- wire from UniLab here
47 µF
To RESET PIN of Microprocessor

Typical "power on reset circuit" for Z80 microprocessor, showing connection of RES- line from Unilab



Typical "power on reset circuit" for Intel microprocessor, showing connection of RES- line from Unilab

Attach NMI- wire

The NMI- signal asserted by the UniLab causes the target microprocessor to vector to a hardware interrupt location. The UniLab software uses the hardware interrupt for several DEBUG features:

NMI,
RI <trig spec> **SI**, and
"silent" DEBUG control for automatic RAM access.

The Non-Maskable Interrupt signal is connected to either the NMI or IRQ circuit of your board.

<u>Simple Pullup Circuits</u>

If you have a simple pullup resistor (a resistor running from the pin to the power supply voltage) attached to the Non Maskable pin of your microprocessor, then you can connect the NMI- wire from the UniLab directly to the microprocessor pin and move on to step 4 of the installation-- checkout your setup.

<u>Common Complication</u>

However, some boards have a logic element in the circuit, an SSI or MSI chip that drives the NMI pin. Sometimes an output of the processor causes the NMI pin to be activated.

Since an "open collector" circuit in the UniLab generates the signal on the NMI- wire, you cannot use it to "pull down" the output of a logic gate.

<u>Hardware Solution</u>

If you encounter this problem, the easiest solution is to temporarily alter your target board:

isolate the NMI pin from the circuit that drives it,
connect a resistor of from 1K to 10K ohms between the pin and your power supply voltage (pullup resistor),
connect the wire from the UniLab directly to the processor pin.

To isolate the NMI pin from the circuit that drives it you can either:

> bend the pin of your processor out of the socket on your target board
>
> OR
>
> build a stacked socket arrangement by cutting the appropriate pin off of a "soldertail socket" and plugging the processor into this new socket. Then put this stacked arrangement into the socket on your target board.

Software Solution

If your target board makes use of the NMI pin of your processor, you might choose to do without the UniLab features that require this resource.

Use the mode panel (press **F8**) to disable the **NMI VECTOR** option on mode panel three, and then use the command **SAVE-SYS** to save the newly configured software.

Other uses of the NMI output

You can use the **INT** command to produce a low-going transition on the NMI- wire when the UniLab goes into trigger search state.

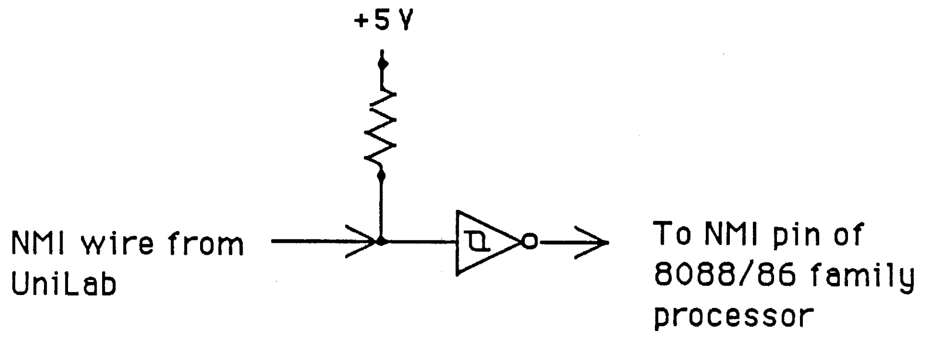
If you need to shut down some peripheral equipment when an error condition occurs, you can disable the UniLab NMI routines and write your own interrupt routine. See the entry on **INT** in the on-line glossary, or in the **Command Reference** chapter of the Reference Manual.

8088 and 8086 Family Complication

Most NMI pins are "active low"-- that is, the pin should normally be held at high voltage, and the program gets interrupted when the pin is pulled to low voltage.

However, Intel chose the opposite convention for their 8088 and 8086 family of processors. The NMI pin on these processors is active high.

This means that you must feed the signal coming from the UniLab through an inverting circuit. One choice is to use a 74LS14, as shown below.



Typical NMI circuit needed for 8088/8086 family processors

<u>4.</u> <u>Verify Your Setup</u>

Overview

Now that you have your target board properly connected, you should give it a small shakedown cruise.

You will load in a sample program and run it while it resides in UniLab emulation ROM.

This will only take a few minutes, and will provide you with a broad idea of what the UniLab can do, at the same time as you test out your installation.

This section assumes that you have purchased a disassembler/DEBUG packages.

If, instead, you purchased the "generic" UniLab software, then you will not have any of the disassembly and DEBUG features. And, a sample program is not included in your software.

<u>Where next?</u>

For more instruction, follow along with the **Guided Demo** in the next chapter. See also the **Operations** chapter.

Load a sample program

Start up the UniLab program if you have not already done so. press function key 10 **(F10)** to get the main menu.

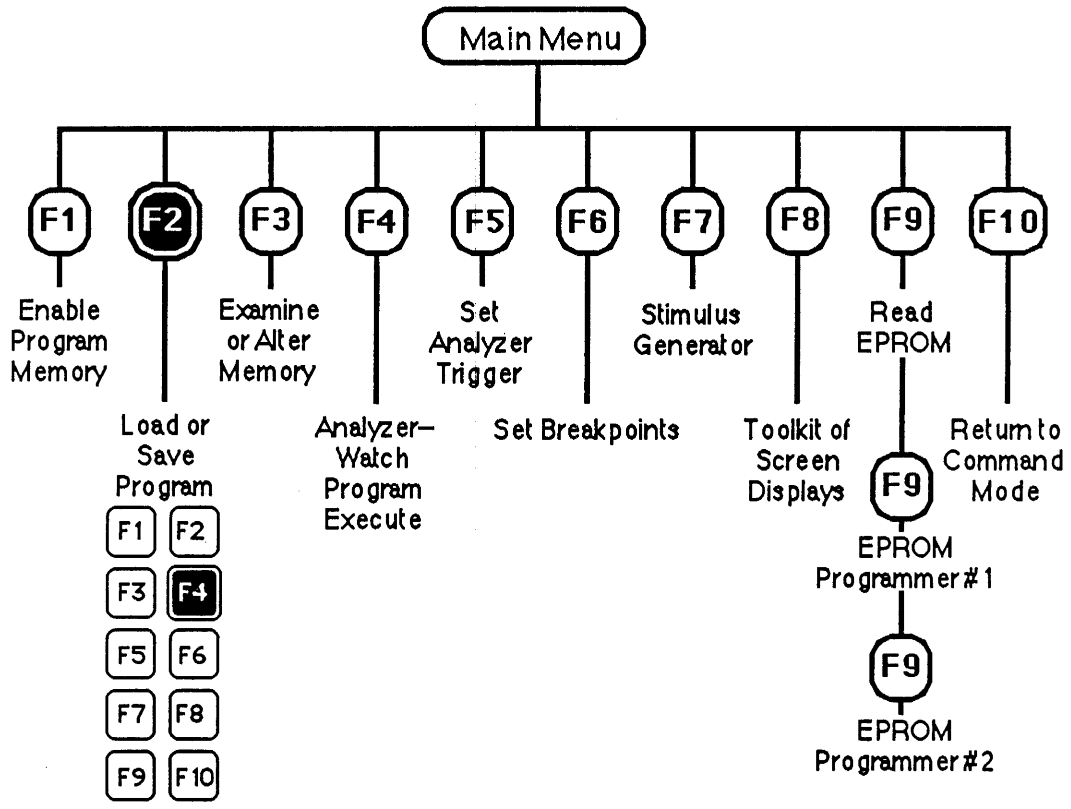
From the main menu of the UniLab program, press

F2

to select the "LOAD OR SAVE A PROGRAM" submenu, then press

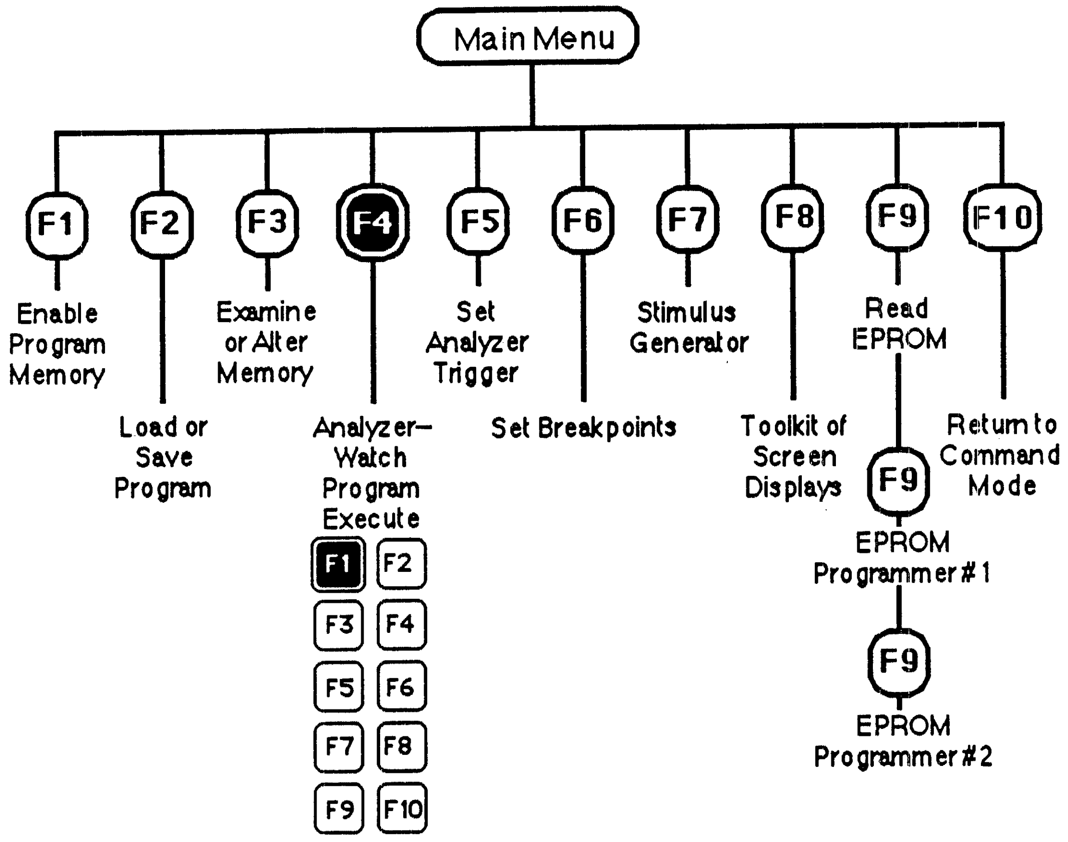
F4

to select "LOAD A SAMPLE PROGRAM." This will enable memory and load the simple demo program for your target processor.



Run the program

To watch the program executing, first press **F10** to return to the main menu and use **F4** to select the "USE THE ANALYZER" sub-menu. Then, press **F1** to select "RESET AND TRACE FIRST CYCLES."



You should get a trace display that agrees with the one in the Target Application Note for your processor. See the next page to find out how to have the computer compare your trace to the sample trace produced by Orion.

<u>Trouble?</u>

If you get a "NO ANALYZER CLOCK" message, or find that your trace buffer is filled with garbage, or have any other problem, then consult the **TroubleShooting** chapter.

Compare to sample trace

After you have generated a trace with the simple target program, compare your trace to the standard trace. Either look at the printout in the Disassembler/DEBUG writeup for your processor, or use the UniLab command

AA TCOMP TESTxxxx**.TRC**

to compare your trace to the sample trace included on the distribution diskette. The sample trace, stored as an encoded file called **TEST**xxxx**.TRC** on your distribution diskette, is not available for all processors. See appendix J, or get a directory listing of your distribution diskette.

If **TCOMP** detects a difference between your trace and the known good trace stored as a file, then it will show you part of the good trace followed by the first differing line of your trace.

If your trace is okay, then you can be confident that you have connected your UniLab properly and that your target board is working. You should skip to the page after next, with the Play Around heading.

If you have a bad trace, whether you detect it visually or with **TCOMP**, see the next page.

Visual Inspection

You should be especially sensitive to four aspects of the trace when you examine it:

1) The very first address-- if it is not right, then you've already found the problem, and shouldn't bother to look any further.
2) The very first item in the data column-- if it is wrong, then you probably have bad data lines on your target board.
3) The value popped off the stack-- if it is not the same as the value pushed, then you probably need to patch the value of the stack pointer in the test program.
4) Other addresses and data-- you can have a problem even though the first part of the program looks okay. For example, you have a grounded address line on your target board, bit 6 of the address. You won't notice this until bit 6 is supposed to go high (40 hex), and doesn't.

Bad Trace?

Typically you will find one of three things wrong with your trace if the fault lies in your connection to the UniLab:

1) Very first address wrong-- you should check: RES- wire and address wires.
2) Control column incorrect-- you should check wires C4 through C7.
3) Bad data popped off stack-- see below.

Bad Data from stack?

Be especially aware of the push and pop instructions early on in the program. Is the same value getting read as is being written? If the answer is no, then it might be that your stack pointer is not pointing at RAM.

Check the Stack Pointer-- Processors with Stack in External RAM

The simple test programs generally set the stack pointer within the first few steps. The program sets the stack pointer so that it points at the RAM on the Orion MicroTarget^{tm} board. If you do not have RAM at that region of the memory map, then the program will read garbage data when it tries to pop values off the stack.

When this happens, you will either get bad data at the breakpoints (all FFFF's, for example) or you will not be able to get DEBUG control at all.

Look at the program as it executes, or look at the listing of it in the DEBUG notes. Is the stack pointer pointing to RAM on your target board?

If the stack pointer needs a different value, use the optional on-line assembler, **ASM**, or use the UniLab command <word> <addr> **MM!**. You can use **MM!** to poke a new 16-bit word into the instruction that initializes the stack pointer. You can easily patch the program, so that the 16-bit address of the stack pointer points to RAM. For example, **FFFE 10 MM!** will put the value FFFE into bytes 10 and 11 of emulation ROM.

Stack Pointer Note

If you change the address of the stack, **TCOMP** will indicate a difference between your trace and the standard trace. You will have to visually inspect the trace to determine whether you have good data.

Once everything is properly connected, you can use **TSAVE** to save a trace for future testing.

Play around a little

This concludes the installation section.

Your target system should now work normally with the emulated ROM. Consult the **TroubleShooting** chapter if you have any problems.

Play with the menu system a bit, to get an idea of the capabilities of the UniLab. Turn to the Menu section of Chapter Three for a map of the menus. The menu mode makes a good interactive learning tool-- before each command that it executes, it echoes to the screen the words that you would type in from command mode.

Experiment a bit with the instrument. For guidance, turn to the next chapter, the **Guided Demo.** When you feel ready to use the command mode, press **F10** once to get into the Main menu, and then a second time to get to command mode.

How to exit

When you want to leave the program, type **BYE** on a line by itself, followed by a carriage return.

<u>Where to go Next</u>

With trouble

If you have trouble getting your system working with the UniLab, follow the suggestions in the **TroubleShooting** chapter of the UniLab reference manual.

With a functioning system

You want to either learn more about the UniLab, or to start using it immediately.

To gain the most familiarity with the instrument you can follow this sequence of four steps:

1) Go through the **Guided Demo** chapter first. You can just read through it, but will learn more by following the steps on your computer.

2) Use the UniLab for a while in MENU mode, to gain familiarity with basic commands.

3) Look at the **Operations** chapter, especially the second section.

4) Use the UniLab on tasks, with the help of the Command Reference card.

Of course, you can skip steps, and even go directly to step four.

Special Note: Operator and Macro Systems

Your software is shipped configured as an operator system. This system recognizes only:

> 1) the commands described in the Command Reference chapter of the UniLab Reference manual (use the command **WORDS** to see the list),
> 2) hexadecimal numbers,
> 3) four mathematical operators (+, -, *, and /), and
> 4) the commands **U.** and **.** (a period), which pop a number off the top of the host system stack and display it. **U.** displays the number as an unsigned integer, while **.** displays the number as a signed integer.

The UniLab uses postfix operators-- type in an operator after the operands. These operators will consume two operands, and leave the result on the host stack. Use **U.** or **.** to (destructively) display the result.

You will not need any other commands, or access to internal variables and constants, unless you need either to write macros, or to customize your software.

If you need more power than the operator system has, use the UniLab command **MACRO** to create a macro system.

You can return to the standard operator system with the command **OPERATOR.**

MAKE-OPERATOR creates a new operator system that recognizes the macros you defined in your macro system. The current macro system will be saved, then an operator system created and saved. You can specify different names for the macro and operator systems. After the files have been saved, **MAKE-OPERATOR** will cause the UniLab program to terminate, returning you to DOS.

The file "MAKE," from your distribution diskette, must be in your ORION directory when you use **MAKE-OPERATOR.**

Use the on-line glossary for more information on these commands.

Special Note: Display Characteristic Commands

Color Monitor

If you have a color monitor, you will want to let the UniLab software know, by entering the command **COLOR.** You can change the default colors with the menu-driven command **SET-COLOR.** See the entries in the Command Reference chapter for more information. After you issue the command **COLOR,** you will want to use **SAVE-SYS** to save the newly configured program.

Screen Flicker

If your monitor flickers when you use the **PgUp** key, you will want to issue the command **CLEAR,** and then use **SAVE-SYS** to save the newly configured program. You can turn this alteration off with **CLEAR'.**

Special Note: Alter the Baud Rate

When you want to change the baud rate of serial communications between the UniLab and the host computer, you must

> toggle a switch on the main board of the UniLab,
> and use a command to change the software.

First

The very first thing you should do is reconsider your decision to change the baud rate. It is rarely, if ever, necessary, and serves only to slow down the software.

But if you decide to change from 19200 baud to 9600 baud, you should follow the instructions in the **TroubleShooting** chapter of volume II.

Chapter Two: Guided Demonstration

Contents

Overview

This Guided Demo uses the menus of the UniLab software to:

1. Enable emulation memory
2. Load a program into memory
3. Look at the program in memory
4. Get a trace of the program executing
5. Set a breakpoint in the program.

You get your program into emulation ROM in steps one and two, double-check your preparations in optional step three, and then work with the program in steps four and five.

The demonstration generates several traces of program execution, and sets a breakpoint to establish DEBUG control over the target processor. It uses a Z80 processor and a very simple program for all the examples. However, you will follow the same procedure when analyzing <u>any</u> program on <u>any</u> processor.

Menus and commands

The heading on each page tells you which command you could use, instead of the menu choice. Also, whenever you execute a menu option the UniLab software tells you what command causes the same effect as your menu choice.

How to follow the demonstration

You can use your UniLab system to follow the steps of this demonstration. You can follow exactly the sequence of function keys pressed in this chapter.

However, unless you have the Z80 DDB package, you will have to enter different values and file names in response to the menu prompts.

The trace and breakpoint displays will differ from processor to processor. For processor-specific trace and breakpoint displays, turn to the Target Application Note for your DDB software.

Related Documentation

Before using this chapter, you should have your UniLab system installed and verified, using the instructions in Chapter One.

After going through this chapter, you will know how to use the basic features of your Disassembler/DEBUG software on your own program. You can gain more expertise as you need it, with the help of the printed documentation and the on-line help facilities.

See the **Operation** chapter for information on the commands and the special features, as well as a complete guide to the menu system. The separate Target Application Note contains information on the processor-specific features of your DDB software.

Call Up the UniLab Software

When you call up the UniLab program, there will be a brief pause while the software is loaded from disk. The first actions the program takes:

1. display the copyright notice,
2. either tell you which processor is supported by this software package,
   OR,
   if your DDB software supports more than one processor, present you with a menu of "processors supported" and wait for you to choose one,
3. display the opening screen,
4. initialize the UniLab.

After it initializes the UniLab, the software will execute any commands that you included on the DOS command line. You will find that useful later, since you can set up a batch file to always load a program and a symbol filé when you enter the UniLab software. See the DOS and UniLab section of the Chapter Three for more information.

```
                    UniLab                 Copyright 198X
                      II                   Orion Instruments
                 Version X.XXX             Redwood City, CA


 XXXXX disassembler installed - with DEBUG.

_______________________________________________________________
          HELP is available on-line by entering HELP or F1.
          Enter  HELP command  to see the definition of "command".
          Type  WORDS command  to see a list of commands.
          ______________________________________________________
          Use the function key F10 for MENU mode operation and quick
            access to most common commands.
          More help is available on the Ctrl-F1 key.
          Press Ctrl-F10 for display of cursor key functions.
          ______________________________________________________
          Type MESSAGE for current messages.
Initializing UniLab...
  Initialized   32K Emulation Memory
```

Get the Main Menu

In command mode, press function key 10 (**F10**) to get the main menu.

```
                         UniLab MAIN MENU

                 F1    ENABLE PROGRAM MEMORY
                 F2    LOAD OR SAVE A PROGRAM
                 F3    EXAMINE OR CHANGE PROGRAM MEMORY
                 F4    ANALYZER -- WATCH PROGRAM EXECUTE
                 F5    ADVANCED ANALYZER TRIGGER
                 F6    SET BREAKPOINTS AND SINGLE STEP PROGRAM
                 F7    USE THE STIMULUS GENERATOR
                 F8    TOOLKIT ROUTINES
                 F9    READ OR PROGRAM A PROM
                F10    EXIT TO COMMAND MODE
```

Before you load a program into the UniLab emulation ROM, you must enable a range of program memory. This tells the UniLab hardware which ROM addresses it is to emulate.

You will want to know the memory map of your target board, so that you do not cause a bus contention problem by mapping emulation ROM into the same address space as target RAM. However, certain processors, such as the 8051, do allow RAM and ROM to occupy the same address at the same time.

If you use the command **LTARG** instead, you will not need to enable memory first-- that command enables the range it needs before poking a test program into memory.

The Five-Step Demonstration

1. Enable a segment of memory

The command is: **EMENABLE**

In the MAIN menu, press function key 1 (**F1**) to get to the ENABLE PROGRAM MEMORY menu.

In the ENABLE menu, press **F2** to enable a range of memory. You will be prompted for the start and end address of the range to emulate. After you enter the parameters, you will see a report on the current setting of emulation memory.

With the Z80, we will be loading a program into the lowest 2K of memory, addresses 0 through 7FF (the UniLab expects all numbers in hexadecimal). This range includes the reset address of the Z80 processor: 0000.

```
                         ENABLE PROGRAM MEMORY MENU

                          F1   DISPLAY CURRENT STATUS OF EMULATION MEMORY
                          F2   ENABLE A RANGE OF EMULATION MEMORY
                          F3   ADD ANOTHER RANGE OF MEMORY
                          F4   SET A16-A19 MEMORY SEGMENT BITS
                          F5   DISABLE ALL EMULATION MEMORY
                         F10  RETURN TO MAIN MENU

Enter starting address of emulation memory (on 2K boundary):0
Enter ending address of emulation memory (rounded to 2K blocks):7FF
 The command is: 0 7FF EMENABLE

 Emulator Memory Enable Status:
                 7 =EMSEG
        0 TO  7FF EMENABLE
```

Emulator memory enable status

The memory status report tells you not only the setting of **EMENABLE**, but also the value of **=EMSEG**, which has already been set to the correct default value for your processor.

The **=EMSEG** value will differ from DDB to DDB. It corresponds to the most significant four bits (A16-A19) of the 20-bit address input to the UniLab. The UniLab emulation ROM will only respond on the bus when the upper four bits of the address input match the **=EMSEG** value and the lower 16 bits match an emulated address. See the on-line glossary entry, or Section Two of the **Operations In Detail** chapter for more information on **=EMSEG**.

2. Load a program into memory

Binary file load The command is: **BINLOA**

After you enable the proper range of memory, press **F10** to get back to the MAIN menu.

In the MAIN menu press **F2** to get the LOAD OR SAVE A PROGRAM menu.

```
                UniLab MAIN MENU

          F2   LOAD OR SAVE A PROGRAM
```

In the LOAD menu press **F2** to load a binary format file. You will be prompted for the start and end addresses for this memory load. The UniLab software will stop at the end of the file, or at the "end address," whichever comes first. When it is done loading, the UniLab software tells you the last address that it wrote into.

Load the short and simple program included on the distribution diskette. The program for the Z80 gets loaded in starting at address 0000. This sample program is very short (32 bytes), so the UniLab only loads up to address 31.

Some disassembler packages support several different processors, and so will have several different **.BIN** files on the diskette. Consult Appendix J if you are not certain which file to load in, or what address to start loading at. Or, instead, you can choose **F4** on this menu, which will both enable memory and load the test program for you.

```
                          LOAD OR SAVE PROGRAM MENU

                         F1   LOAD INTEL HEX FILE
                         F2   LOAD BINARY OBJECT FILE
                         F3   SAVE A BLOCK OF MEMORY TO DISK FILE
                         F4   LOAD A SAMPLE PROGRAM
                        F10  RETURN TO MAIN MENU

Enter the Starting address:0
Enter the Ending address:7FF
 The command is: 0 7FF BINLOAD

File Name? --- a:testZ80.bin end = 31
```

3. Examine the program in emulation memory

Memory Dump The command is: **MDUMP**

After you load the program, press **F10** to return to the MAIN menu.

In the MAIN menu, press **F3** to get the EXAMINE OR CHANGE PROGRAM MEMORY menu.

```
                    UniLab MAIN MENU

                F3   EXAMINE OR CHANGE PROGRAM MEMORY
```

In the EXAMINE MEMORY menu, press **F8** to "dump" a range of memory. You will be prompted for the start and end address of the range to dump.

This simple command for examining memory shows you only the hexadecimal contents and ASCII interpretation of each byte. Since it works on 10-byte chunks of memory, you get the full range 0 through 2F when you ask for 0 to 2B.

```
                        EXAMINE OR CHANGE PROGRAM MEMORY MENU

                        F1   EXAMINE AND ALTER MEMORY
                        F2   DISASSEMBLE FROM MEMORY
                        F3   CHANGE ONE BYTE
                        F4   CHANGE ONE WORD
                        F5   FILL A BLOCK OF MEMORY WITH ONE VALUE
                        F6   MOVE A BLOCK OF MEMORY
                        F7   COMPARE TWO BLOCKS OF MEMORY
                        F8   EXAMINE A RANGE MEMORY
                      F10  RETURN TO MAIN MENU

 Enter the Starting address:0
 Enter the Ending address:2B

  The command is: 0 2B MDUMP
  0    31 00 19 3E 12 01 56 34  11 9A 78 21 DE BC C5 C1   1..>..V4..x!....
 10    3C 3C 3C 3C 3C 3C 3C 3C  3C 3C 3C 3C 3C 3C 3C 3C   <<<<<<<<<<<<<<<<
 20    3C 3C 3C 3C 3C 3C 3C 3C  3C C3 05 00 8E 84 F7 A0   <<<<<<<<<.......
```

When you work on program memory, you will more often use the command that disassembles from memory, rather than just dumping memory. See the next page.

3. Examine the program in emulation memory

Disassemble the program The command is: **DM**

In the EXAMINE MEMORY menu press **F2** to disassemble from memory. You will be prompted for the start address and the number of lines to disassemble.

Here we disassemble 0A lines (10 decimal) so that the disassembled display fits on the screen.

When you are in command mode, you can also use the command **DN**, which requires only a starting address, and disassembles into a full length window on the right-hand side of the screen.

```
                    EXAMINE OR CHANGE PROGRAM MEMORY MENU

                  F1    EXAMINE AND ALTER MEMORY
                  F2    DISASSEMBLE FROM MEMORY
                  F3    CHANGE ONE BYTE
                  F4    CHANGE ONE WORD
                  F5    FILL A BLOCK OF MEMORY WITH ONE VALUE
                  F6    MOVE A BLOCK OF MEMORY
                  F7    COMPARE TWO BLOCKS OF MEMORY
                  F8    EXAMINE A RANGE MEMORY
                 F10  RETURN TO MAIN MENU



Enter the Starting address:0
Enter the number of lines to disassemble (default=5):A

 The command is: 0 10 DM
0000  310019   LD SP,1900
0003  3E12     LD A,12
0005  015634   LD BC,3456
0008  119A78   LD DE,789A
000B  21DEBC   LD HL,BCDE
000E  C5       PUSH BC
000F  C1       POP BC
0010  3C       INC A
0011  3C       INC A
0012  3C       INC A
```

You may have already noticed that the simple program in the **.BIN** file is identical to the program poked into memory by the command **LTARG.**

4. Use the Analyzer

Though it was helpful to see a disassembly of the program from memory, the value of the UniLab comes from the ability to watch your microprocessor system as it executes the program.

In the EXAMINE MEMORY menu, press **F10** to return to the MAIN menu.

In the MAIN menu, press **F4** to get the ANALYZER menu.

UniLab MAIN MENU

F4 ANALYZER -- WATCH PROGRAM EXECUTE

4. Use the Analyzer

Reset the microprocessor, and watch the first cycles

The command is: **STARTUP**

In the ANALYZER menu, press **F1** to reset your microprocessor and display a trace buffer of the first cycles of the program execution.

Only the first few lines of the trace are shown, but you can look at more of the trace by using the **Down Arrow** and the **PgDn** keys on the cursor/numeric key pad. The **Home** key shows the trace from the top. See the Special Features section of the Chapter Three for more information on the use of these keys.

```
                         ANALYZER MENU

                 F1   RESET AND TRACE FIRST CYCLES
                 F2   TRACE IMMEDIATELY
                 F3   TRACE FROM A SPECIFIC ADDRESS
                 F4   COUNT CYCLES BETWEEN TWO ADDRESSES
                 F5   SAMPLE THE BUS CONTINUOUSLY
                 F6   SAMPLE ADDRESS ACTIVITY
                F10  RETURN TO MAIN MENU

The command is: STARTUP  resetting
cy# CONT ADR DATA
  0  B7 0000 310019   LD SP,1900
  3  B7 0003 3E12     LD A,12
  5  B7 0005 015634   LD BC,3456
  8  B7 0008 119A78   LD DE,789A
  B  B7 000B 21DEBC   LD HL,BCDE
  E  B7 000E C5       PUSH BC
  F  D7 18FF 34 write
 10  D7 18FE 56 write
 11  B7 000F C1       POP BC
 12  F7 18FE 56 read
 13  F7 18FF 34 read
 14  B7 0010 3C       INC A
 15  B7 0011 3C       INC A
```

Whenever you start the analyzer, the UniLab will receive a trigger specification from the host computer, which tells it, among other things, when to freeze the trace buffer. As soon as the trace buffer has been frozen, it is transferred to the host for display. Your microprocessor, however, continues to execute code.

See Section One of the **Operations In Detail** chapter to find out more about interpreting the trace.

4. Use the Analyzer

Sample the bus The command is: **SAMP**

Though you are looking at a trace of the first cycles of the program, the program continues to run.

In the ANALYZER menu, press **F5** to sample about one bus cycle every second. These are random selections from the microprocessor bus. Press any key to stop the display.

```
                         ANALYZER MENU

              F1   RESET AND TRACE FIRST CYCLES
              F2   TRACE IMMEDIATELY
              F3   TRACE FROM A SPECIFIC ADDRESS
              F4   COUNT CYCLES BETWEEN TWO ADDRESSES
              F5   SAMPLE THE BUS CONTINUOUSLY
              F6   SAMPLE ADDRESS ACTIVITY
             F10  RETURN TO MAIN MENU

 The command is: SAMP

F7 000C DE read
B7 001C 3C       INC A
B7 001C 3C       INC A
B7 001F 3C       INC A
F7 002B 00 read
B7 001D 3C       INC A
F7 0007 34 read
```

This use of **SAMP** confirms what was already obvious from the **STARTUP** trace-- the program spends most of its time executing "INC A" instructions.

Since **SAMP** gathers isolated bus states, the data going to the trace disassembler is isolated bytes (or words, with 16-bit processors). This means that often the disassembler sees only a portion of a multi-byte instruction.

For example, the very first cycle captured by **SAMP** in the transcript above is interpreted as a read of DE. However, the program never reads a value of DE-- except when it reads the immediate value to load into the HL register, in the fifth line of the program (see previous page).

The lesson: while using **SAMP** either turn off the disassembler (with **DASM'**), or leave it on while realizing that it can be "fooled."

4. Use the Analyzer

Set a trigger on an address The command is: <address> **AS**

The preset triggers of the UniLab are helpful, but once you become familiar with the UniLab system you will usually prefer set up your own trigger specification.

In the ANALYZER menu, press **F3** to set up a trigger on a single address. You will be prompted for the address.

For the Z80, specify 29, the address of the jump instruction.

```
                              ANALYZER MENU

                     F1   RESET AND TRACE FIRST CYCLES
                     F2   TRACE IMMEDIATELY
                     F3   TRACE FROM A SPECIFIC ADDRESS
                     F4   COUNT CYCLES BETWEEN TWO ADDRESSES
                     F5   SAMPLE THE BUS CONTINUOUSLY
                     F6   SAMPLE ADDRESS ACTIVITY
                    F10  RETURN TO MAIN MENU

Enter the Trigger address:29
 The command is: 29 AS

  -5  B7 0024 3C        INC A
  -4  B7 0025 3C        INC A
  -3  B7 0026 3C        INC A
  -2  B7 0027 3C        INC A
  -1  B7 0028 3C        INC A
   0  B7 0029 C30500    JP 5
   3  B7 0005 015634    LD BC,3456
   6  B7 0008 119A78    LD DE,789A
   9  B7 000B 21DEBC    LD HL,BCDE
   C  B7 000E C5        PUSH BC
   D  D7 18FF 34 write
   E  D7 18FE 56 write
```

The trigger address is always labeled as cycle 0, and the rest of the bus cycles are labeled relative to the trigger.

As always, the program shows you only a screenful of the trace. You can use the **PgDn, Down Arrow** and **Home** keys to look through the trace, as mentioned two pages back.

5. Use the DEBUG
Set a breakpoint to Establish Debug Control

The command is: **RESET** <address> **RB**

When you are done with the ANALYZER menu, press **F10** to get back to the MAIN menu.

In the MAIN menu, press **F6** to choose the DEBUG -- SET BREAKPOINTS AND SINGLE STEP PROGRAM menu.

UniLab MAIN MENU

F6 SET BREAKPOINTS AND SINGLE STEP PROGRAM

When you want to use any of the DEBUG features, you must establish debug control-- which causes special hardware in the UniLab to take control of your microprocessor. There are several ways to establish DEBUG control. In this demonstration, we establish DEBUG control by setting a breakpoint.

In the DEBUG menu, press **F1** to set a breakpoint. You will be prompted for the program address where you wish to set the breakpoint. The example shows a breakpoint set at address **27**, two cycles before the JUMP instruction.

For more information on DEBUG control, consult the on-line glossary entries for specific commands, or read the DEBUG section of the **Operations In Detail** chapter. The Target Application Note for your DDB software contains a sample trace and DEBUG session.

DEBUG MENU

```
F1   SET A BREAKPOINT TO ESTABLISH DEBUG CONTROL
F2   RESUME EXECUTION TO A BREAKPOINT
F3   EXECUTE THE NEXT STEP (WON'T FOLLOW JUMPS & BRANCHES)
F4   SINGLE STEP or IMMEDIATE DEBUG CONTROL
F5   GO TO AN ADDRESS WITH A BREAKPOINT SET
F6   GO TO AN ADDRESS AND EXIT THE DEBUG
F10  RETURN TO MAIN MENU

Enter the breakpoint address in emulation memory:27
The command is: RESET 27 RB  resetting

AF=2928 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900
0027  3C       INC A                          (next step)
```

---

5. Use the DEBUG

Set another breakpoint The command is: <address> **RB**

Once you have established DEBUG control, you can use any of the other DEBUG features.

In the DEBUG menu, press **F2** to set a breakpoint and then release the processor. You will be prompted for the address. When the program reaches the new breakpoint, you will again see the breakpoint display.

Here we begin with the processor stopped just before address 27 and set the next breakpoint to address 5. This allows us to see the state of the processor immediately after it executes the JUMP instruction.

```
                              DEBUG MENU

             F1    SET A BREAKPOINT TO ESTABLISH DEBUG CONTROL
             F2    RESUME EXECUTION TO A BREAKPOINT
             F3    EXECUTE THE NEXT STEP (WON'T FOLLOW JUMPS & BRANCHES)
             F4    SINGLE STEP or IMMEDIATE DEBUG CONTROL
             F5    GO TO AN ADDRESS WITH A BREAKPOINT SET
             F6    GO TO AN ADDRESS AND EXIT THE DEBUG
             F10   RETURN TO MAIN MENU

AF=2928 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900
0027  3C      INC A                             (next step)


Enter the breakpoint address in emulation memory: 05
The command is: 05 RB
AF=2B28 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900
0005  015634   LD BC,3456                       (next step)
```

Breakpoints and DEBUG control

When you set a breakpoint to establish debug control, the target microprocessor gets reset after the UniLab system sets up the software breakpoint. When you resume to a breakpoint, the processor continues execution from the current breakpoint.

5. Use the DEBUG
Single-step through code The Command is: **NMI**

The UniLab single-steps either by issuing a hardware interrupt (to the NMI or IRQ pin of the processor) or by setting a breakpoint just after the instruction currently pointed to by the program counter.

In the DEBUG menu, press **F3** to single-step with the hardware interrupt. You can also use **F4** to single step, but that option will not follow branches, calls, etc.

```
                    DEBUG MENU

        F1   SET A BREAKPOINT TO ESTABLISH DEBUG CONTROL
        F2   RESUME EXECUTION TO A BREAKPOINT
        F3   EXECUTE THE NEXT STEP (WON'T FOLLOW JUMPS & BRANCHES)
        F4   SINGLE STEP or IMMEDIATE DEBUG CONTROL
        F5   GO TO AN ADDRESS WITH A BREAKPOINT SET
        F6   GO TO AN ADDRESS AND EXIT THE DEBUG
        F10  RETURN TO MAIN MENU

AF=2928 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900
0027  3C       INC A                           (next step)

 The command is: NMI
AF=2A28 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900
0028  3C       INC A                           (next step)

 The command is: NMI
AF=2B28 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900
0029  C30500   JP 5                            (next step)

 The command is: NMI
AF=2B28 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900
0005  015634   LD BC,3456                      (next step)
```

Here we begin with the processor stopped at address 27, and then press **F4** three times, to step up to and then through the jump instruction.

You can also use the **NMI** command to establish DEBUG control in the first place, instead of **RESET** <addr> **RB.** In fact, if you accidentally set a breakpoint on an address that is never executed, then you can press any key, which will generate a hardware interrupt.

End Note

Summary

This demo has gone through the process of loading a program, looking at it, and running it. It generated two traces of the program, then set breakpoints and single-stepped through a jump instruction.

How to leave

If you now want to exit from the UniLab program, type **BYE**.

Preview: advanced trigger and trace

Every bus cycle, the UniLab reads in 6 bytes of information. In this Guided Demo we set a trigger specification on only 2 bytes-- the 16 bits of the address input with the **AS** command.

The trigger can be a much more sophisticated than that. You can also:

1) set triggers on any combination of the six bytes that the UniLab reads from the target board each bus cycle,
2) use qualifiers to delay the search for the trigger until after some other bus activity occurs,
3) and produce filtered traces that can include or exclude the bus cycles you describe.

These capabilities are discussed in the Command section of Chapter Three.

Preview: more DEBUG

As was hinted in the last section of this demonstration, there is more than one way to establish DEBUG control. And once you have control, there are a number of things you are able to do, besides setting breakpoints and single-stepping through emulation ROM:

1) alter any writable register,
2) examine and alter target system RAM,
3) read and write to ports,
4) set multiple breakpoints,
5) set breakpoints and single-step through RAM,
6) exit DEBUG control.

These are discussed in the Target Application Note for your DDB software, as well as in the DEBUG section of the Chapter Six.

Preview: command mode

As you use the menus, you will gain familiarity with UniLab methods and commands. Soon you will feel comfortable enough to operate in command mode most of the time.

In command mode you have access to many helpful features that you cannot use in menu mode. For example, you can split the screen, and display different data in the top and bottom. Or, you can invoke the "mode panel" and toggle features on and off.

For more information, consult the Commands and Special Features sections of Chapter Three.

Next

With the background this chapter has given you, you should be able to get to work with the basic features of your software.

You can gain more expertise as you need it, with the help of the printed documentation and the on-line help facilities.

Chapter Three: Operations

Contents

Overview

Chapter description

This four section chapter provides an overview of the UniLab features.

Section One is an exhaustive guide to the menu system.

Section Two introduces the UniLab command language, then focuses on how to build a trigger specification. It includes flow charts which illustrate the trigger commands. Many examples are worked through and explained.

The brief third section shows how to use DOS batch files and the UniLab command tail. This knowledge can reduce repetition and tedium while you test and alter your code.

The last section demonstrates special screen-oriented features of the UniLab, and explains the features assigned to function keys and cursor keys in command mode.

When to use: menus, commands and special features

Menus

The menus help to guide your activities when you first work with the UniLab. And they help you learn the command language, by displaying the command which corresponds to your menu selection. Most UniLab commands can be used while in the menu system. This allows you to try out a command as soon as you learn it. After a short time, you will be ready to work without menus.

Commands

Use the command mode when you have gained passing familiarity with some of the UniLab's functions. In command mode you have access to the full power of the UniLab trace, trigger and DEBUG features, and can take advantage of the special features that are not available in menu mode.

Special features

You can start using the special features as soon as you leave menu mode. The wealth of special features includes:

the split screen, to show several types of information at once,

the screen history, to recover commands and displays,

the command line editor, to edit and re-issue commands,

the mode panel, to toggle features on and off, etc.

Related chapters

When you require more information than this chapter provides, turn to the **Operations In Detail** chapter of the Reference Manual for in-depth information about every aspect of the instrument.

For information on the Program Performance Analyzer (PPA) consult Chapter Four.

If you have not yet installed and verified your UniLab system, turn to Chapter One, **Installation.**

1. The Menu Mode

The menu system allows you to do work with the UniLab without any previous experience-- and at the same time gain familiarity with the commands. You enter (and leave) the menu system with function key 10 (**F10**).

You get the Program Performance Analyzer (PPA) menu by pressing **ALT-F10**. See the next chapter for more information on the PPA.

After you have gained familiarity with the UniLab program, you will spend most of your time working with the command language, which allows you to control all aspects of the instrument from a single context.

Choose from menus

You choose from the menus with the press of a function key. Whenever you make a menu choice, the UniLab program will:

ask you for any needed parameters,
tell you what command you would use,
and then execute the command.

The arrangement of options

Within many of the menus, the options appear in roughly the order that you will need them. For example, the ADVANCED ANALYZER menu shows the trigger spec commands in order of increasing complexity.

Commands and special features in menu mode

You can use many UniLab commands from within the menu mode.

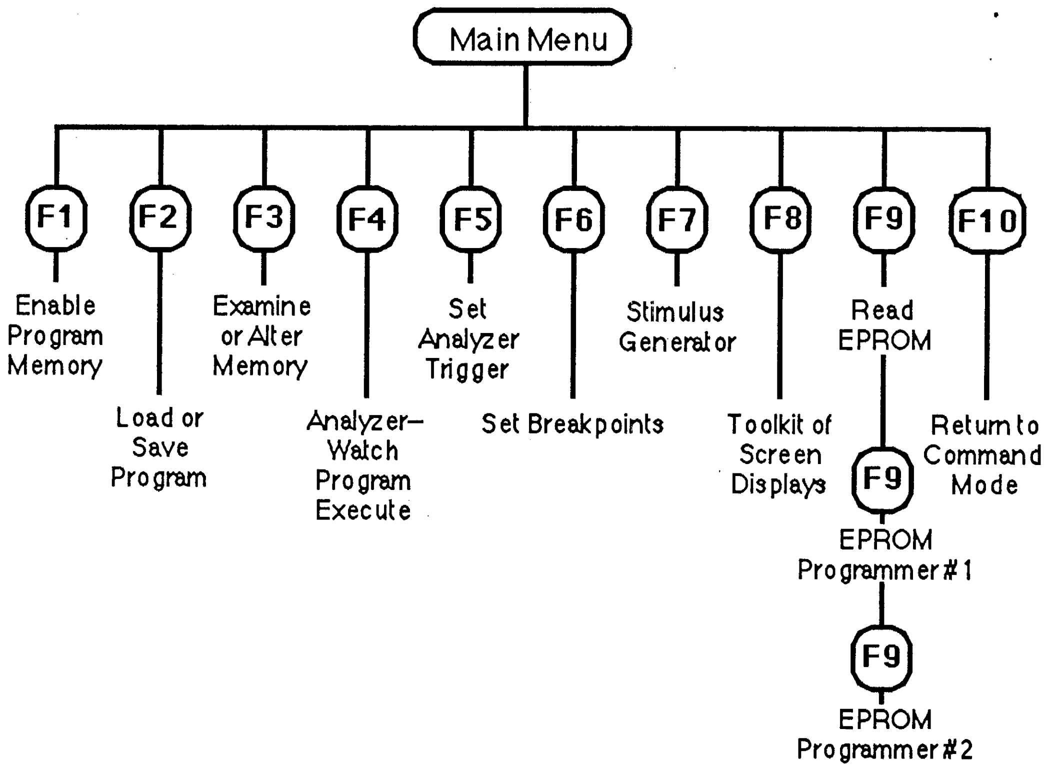
However, the mode panels, windows and most of the other special functions are not available.

Map to the menu system

Each of the following pages shows one of the menus, and briefly explains the menu choices.

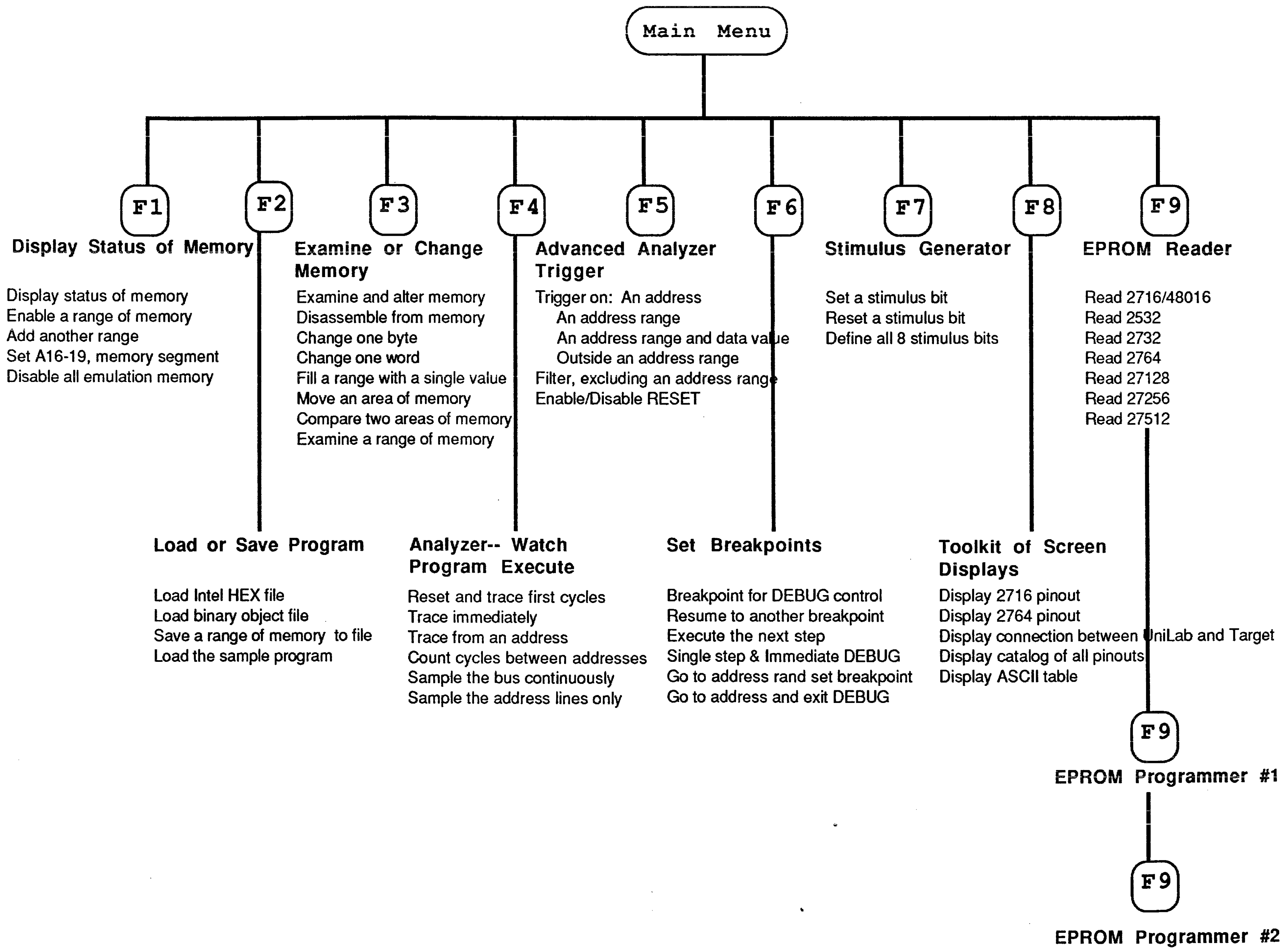
The chart below shows you the straightforward arrangement of the menus. The only complication, and that a mild one, is that you must travel through the EPROM reader menu to get to the two EPROM programmer menus. Even with this inconvenience, you can get any EPROM programmed with five key strokes, in the very worst case.

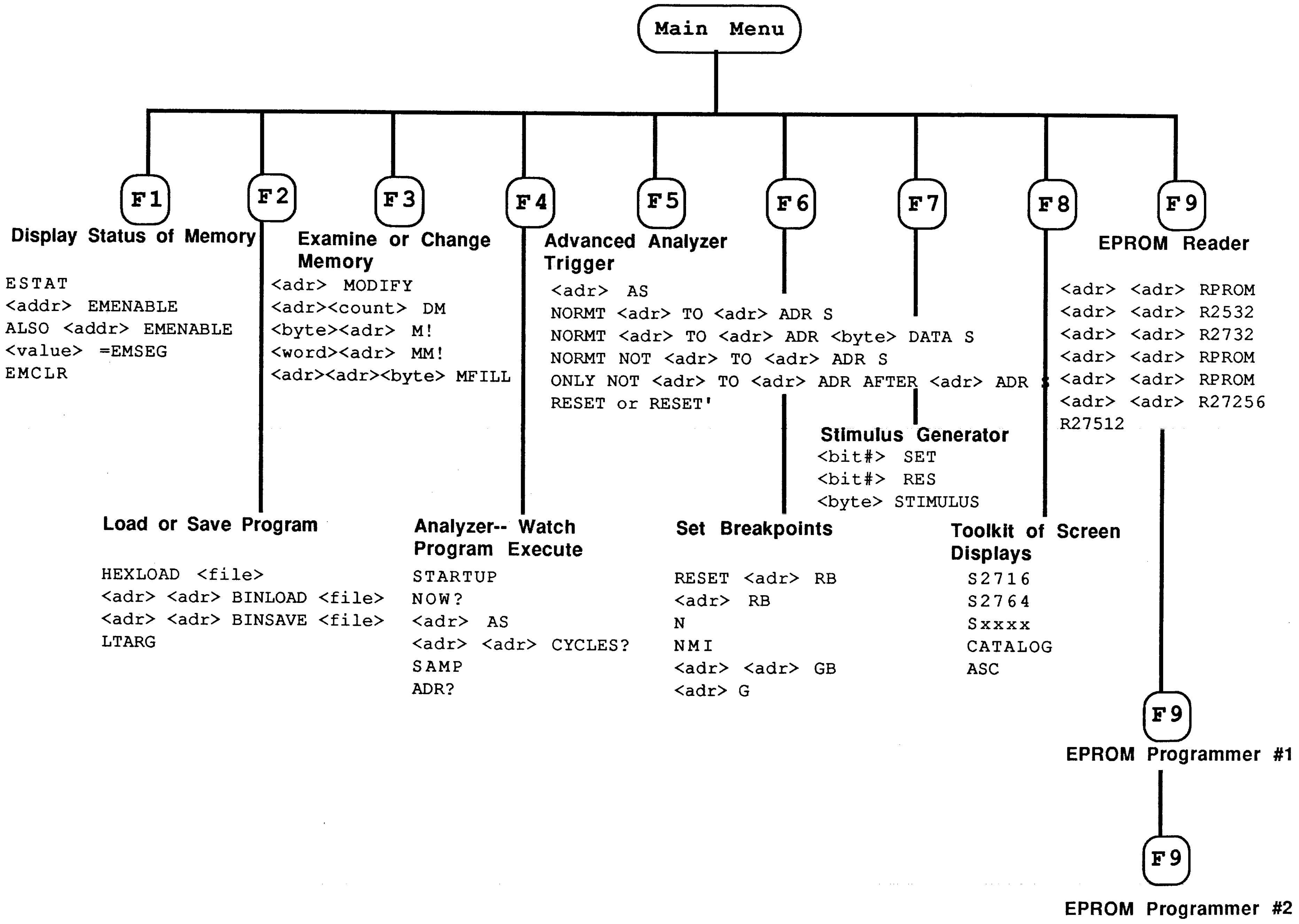
The charts on the next two pages illustrate the functions available from each submenu. The first chart shows the descriptions of the options, the second lists the equivalent commands.



In the Main menu, press **F10** to exit from menu mode. Press **F10** again to return to menu mode.

From any of the sub-menus, press **F10** to return to the Main menu.


Main Menu
F1
Display Status of Memory
Display status of memory
Enable a range of memory
Add another range
Set A16-19, memory segment
Disable all emulation memory
F2
Load or Save Program
Load Intel HEX file
Load binary object file
Save a range of memory to file
Load the sample program
F3
Examine or Change Memory
Examine and alter memory
Disassemble from memory
Change one byte
Change one word
Fill a range with a single value
Move an area of memory
Compare two areas of memory
Examine a range of memory
F4
Analyzer-- Watch Program Execute
Reset and trace first cycles
Trace immediately
Trace from an address
Count cycles between addresses
Sample the bus continuously
Sample the address lines only
F5
Advanced Analyzer Trigger
Trigger on: An address
An address range
An address range and data value
Outside an address range
Filter, excluding an address range
Enable/Disable RESET
F6
Set Breakpoints
Breakpoint for DEBUG control
Resume to another breakpoint
Execute the next step
Single step & Immediate DEBUG
Go to address rand set breakpoint
Go to address and exit DEBUG
F7
Stimulus Generator
Set a stimulus bit
Reset a stimulus bit
Define all 8 stimulus bits
F8
Toolkit of Screen Displays
Display 2716 pinout
Display 2764 pinout
Display connection between UniLab and Target
Display catalog of all pinouts
Display ASCII table
F9
EPROM Reader
Read 2716/48016
Read 2532
Read 2732
Read 2764
Read 27128
Read 27256
Read 27512
F9
EPROM Programmer #1
F9
EPROM Programmer #2



F1 Display Status of Memory

```
ESTAT
<addr>  EMENABLE
ALSO <addr> EMENABLE
<value>  =EMSEG
EMCLR
```

F2 Load or Save Program

```
HEXLOAD <file>
<adr> <adr> BINLOAD <file>
<adr> <adr> BINSAVE <file>
LTARG
```

F3 Examine or Change Memory

```
<adr>  MODIFY
<adr><count>  DM
<byte><adr>  M!
<word><adr>  MM!
<adr><adr><byte> MFILL
```

F4 Analyzer-- Watch Program Execute

```
STARTUP
NOW?
<adr>  AS
<adr> <adr> CYCLES?
SAMP
ADR?
```

F5 Advanced Analyzer Trigger

```
<adr>  AS
NORMT <adr> TO <adr> ADR S
NORMT <adr> TO <adr> ADR <byte> DATA S
NORMT NOT <adr> TO <adr> ADR S
ONLY NOT <adr> TO <adr> ADR AFTER <adr> ADR S
RESET or RESET'
```

F6 Set Breakpoints

```
RESET <adr> RB
<adr>  RB
N
NMI
<adr> <adr> GB
<adr> G
```

F7 Stimulus Generator

```
<bit#>  SET
<bit#>  RES
<byte> STIMULUS
```

F8 Toolkit of Screen Displays

```
S2716
S2764
Sxxxx
CATALOG
ASC
```

F9 EPROM Reader

```
<adr>  <adr>  RPROM
<adr>  <adr>  R2532
<adr>  <adr>  R2732
<adr>  <adr>  RPROM
<adr>  <adr>  RPROM
<adr>  <adr>  R27256
R27512
```

F9 EPROM Programmer #1

F9 EPROM Programmer #2

The UniLab system will present you with the main menu whenever you enter menu mode, which you do either with function key 10, or with the command **MENU** :

```
                 UniLab MAIN MENU

                F1   ENABLE PROGRAM MEMORY
                F2   LOAD OR SAVE A PROGRAM
                F3   EXAMINE OR CHANGE PROGRAM MEMORY
                F4   ANALYZER-- WATCH PROGRAM EXECUTE
                F5   ADVANCED ANALYZER TRIGGER
                F6   SET BREAKPOINTS AND SINGLE STEP PROGRAM
                F7   USE THE STIMULUS GENERATOR
                F8   TOOLKIT ROUTINES
                F9   READ OR PROGRAM A PROM
               F10   EXIT TO COMMAND MODE
```

Explanation

None of the entries on this menu actually correspond to UniLab commands. You use this menu to choose the appropriate sub-menu. The options appear in the order that you will need them:

To work on your target software in emulation ROM, you need first to enable memory (**F1**), and then load the program into emulation memory (**F2**).

After that, you will probably want either to look at the program in memory (**F3**) or to watch it execute (**F4**). You will need both of these capabilities during your work with the target system.

Most of your time with the UniLab will be spent setting trigger specifications and examining the traces that result. This is how you track down bugs. The menu supplies the most common trigger commands (**F5**). You will need to use commands to make use of the full power of the UniLab trigger logic.

After you track the bug down to a small section of code, you might want to look at the internal state of the processor (**F6**) while it executes that portion of the program-- though this is not necessary for most test and diagnostic work. But if you want to look at the internal state, you must first establish DEBUG control. You can then take advantage of all the DEBUG features, including the ability to single-step.

When you are done with testing and altering your code, you can burn the tested program into an EPROM (**F9**).

In the Main Menu, press **F1** to get the first sub-menu:

```
                    ENABLE PROGRAM MEMORY MENU

                    F1   DISPLAY CURRENT STATUS OF EMULATION MEMORY
                    F2   ENABLE A RANGE OF EMULATION MEMORY
                    F3   ADD ANOTHER RANGE OF MEMORY
                    F4   SET A16-A19 MEMORY SEGMENT BITS
                    F5   DISABLE ALL EMULATION MEMORY
                   F10  RETURN TO MAIN MENU
```

Explanation

Use this menu to tell the UniLab which ROM addresses it is to emulate.

The UniLab can load programs into any area of emulation memory that has been enabled. Check the current status of emulation memory with the first menu choice (**F1** or **ESTAT**) before loading in a program. The UniLab can load a program into target RAM, after DEBUG control has been established.

Enable an area of emulation memory with the second menu choice (**F2** or <start addr> <end addr> **EMENABLE**). This will clear out any previous enable settings. The command **SAVE-SYS** <filename> will save the current settings of UniLab software. Save the software after you enable the range of memory that you need for your project, and you will not need to enable memory again.

To enable another range of memory without clearing out the previous setting, pick the third item on the menu (**F3** or **ALSO** <start addr> <end addr> **EMENABLE**).

The upper four bits of the address are properly set up by each Disassembler/DEBUG software package. In the unlikely event that you need to change that setting, use menu choice four (**F4** or <hex digit> **=EMSEG**). The change does not have any effect until the next **EMENABLE** command. See the first section of the **Operations In Detail** chapter in the UniLab Reference Manual if you need more information about the upper four bits of the address.

To disable all emulation memory use the fifth menu choice (**F5** or **EMCLR**). Normally this is done only when you want to run a program from a ROM chip on the microprocessor board. This command also disables the DEBUG features.

As in all sub-menus, F10 returns you to the MAIN MENU.

Press **F2** from the MAIN MENU to get the second sub-menu:

```
                    LOAD OR SAVE PROGRAM MENU

                     F1   LOAD INTEL HEX FILE
                     F2   LOAD BINARY OBJECT FILE
                     F3   SAVE A RANGE OF MEMORY TO DISK FILE
                     F4   LOAD A SAMPLE PROGRAM
                    F10  RETURN TO MAIN MENU
```

Explanation

Use this sub-menu to load programs into emulation memory, or to save memory as a binary file.

An INTEL hex format file contains within it the address that every byte of code will go to. When you load one of these files, you specify only the name of the file, since you cannot choose what addresses it will load into (**F1** or **HEXLOAD** <file name>).

When you load a binary object file, you must specify where in emulation ROM to start and end
(**F2** or <from addr> <to addr> **BINLOAD** <file name>). The program will load until either it reaches the <to addr>, or it reaches the end of file, whichever comes first. If you wish, you can load only a portion of a file.

Save a range of memory to a disk file with the third menu choice
(**F3** or <from addr> <to addr> **BINSAVE** <file name>).

Load and run the simple test program as part of the verification of your installation (**F4** or **LTARG**). This choice changes the enable status of emulation ROM.

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press **F3** from the MAIN MENU to get the third sub-menu:

```
                    EXAMINE OR CHANGE PROGRAM MEMORY MENU

                   F1   EXAMINE AND ALTER MEMORY
                   F2   DISASSEMBLE FROM MEMORY
                   F3   CHANGE ONE BYTE
                   F4   CHANGE ONE WORD
                   F5   FILL A RANGE OF MEMORY WITH ONE VALUE
                   F6   MOVE AN AREA OF MEMORY
                   F7   COMPARE TWO AREAS OF MEMORY
                   F8   EXAMINE A RANGE OF MEMORY
                  F10  RETURN TO MAIN MENU
```

Explanation

Use this sub-menu to examine, alter, move and compare either emulation ROM or target RAM. All memory operations work on RAM, after you have established DEBUG control; or, with the **NMI**-dependent automatic DEBUG control feature, any time you have a program running .

Call the screen oriented display to see the hexadecimal and ASCII representation of memory, and alter it if you wish (**F1** or <addr> **MODIFY**).

Use the disassembler to see what instructions are stored in memory (**F2** or <from addr> <# of instructions> **DM**).

Alter memory either with choice three to change a byte (**F3** or <byte> <address> **M!**), or with choice four to change a 16-bit word (**F4** or <word> <address> **MM!**).

Use menu choice five to fill a block of memory with a single byte (**F5** or <from addr> <to addr> <byte value> **MFILL**).

Move blocks of code from one place to another with choice six (**F6** or <start addr source> <end addr source> <start addr dest> **MMOVE**).

Use choice seven to compare two areas of memory, and get a display of where they differ (**F7** or <from addr> <to addr> <comp addr> **MCOMP**).

Choice eight shows a hexadecimal and ASCII dump of any range of memory (**F8** or <from addr> <to addr> **MDUMP**).

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press **F4** from the MAIN MENU to get the fourth sub-menu:

```
                    ANALYZER MENU

                      F1   RESET AND TRACE FIRST CYCLES
                      F2   TRACE IMMEDIATELY
                      F3   TRACE FROM A SPECIFIC ADDRESS
                      F4   COUNT CYCLES BETWEEN TWO ADDRESSES
                      F5   SAMPLE THE BUS CONTINUOUSLY
                      F6   SAMPLE ADDRESS ACTIVITY
                     F10  RETURN TO MAIN MENU
```

Explanation

This submenu provides six "pre-configured" analyzer triggers, that will capture and display program execution.

Use the first menu choice to look at the first cycles that your program executes (**F1** or **STARTUP**). This is a good first step when you start testing a new piece of software for your microprocessor board,

Choice two captures a "snapshot" trace buffer of the current state of the target system (**F2** or **NOW?**). This command demonstrates an important principle: your processor does not stop when you capture a trace. While you are examining the "frozen" trace buffer, the processor continues to execute code.

The simplest-- and most frequently used-- trigger specification tells the UniLab to show what happens after the program reaches the address you specify (**F3** or <addr> **AS**).

Use choice four to count the number of bus cycles that occur between addresses (**F4** or <first addr> <second addr> **CYCLES?**).

Choice five gives a rather rough grained view of your program by displaying one bus cycle per second (**F5** or **SAMP**).

An even rougher grained view is available. You can look at sample of only the address bus (**F6** or **ADR?**).

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press **F5** from the MAIN MENU to get the fifth sub-menu:

```
                  ANALYZER TRIGGER MENU

                 F1   TRIGGER ON AN ADDRESS
                 F2   TRIGGER ON A RANGE OF ADDRESSES
                 F3   TRIGGER ON A RANGE OF ADDRESSES AND A DATA VALUE
                 F4   TRIGGER OUTSIDE A RANGE OF ADDRESSES
                 F5   FILTER EXCLUDING A RANGE OF ADDRESSES AFTER AN
                    ADDRESS
                 F6   TURN RESET OFF OR ON (reset is now on)
                F10   RETURN TO MAIN MENU
```

Explanation

This sub-menu provides several of the more common triggers that can be built with the UniLab command language.

The first choice is a repeat from the fourth menu. It sets a trigger on an address (**F1** or <addr> **AS**).

Use the second choice to trigger when any member of some <u>range</u> of addresses appears on the bus (**F2** or <addr> **TO** <addr> **ADR S**). You can also set a trigger on several ranges of addresses, but not with a menu selection.

Choice three does allow you to trigger only when both a given range of addresses and a given data value appear on the bus (**F3** or **NORMT** <addr> **TO** <addr> **ADR** <byte> **DATA S**).

Use choice four to trigger when the processor goes outside a given range of memory (**F4** or **NOT** <addr> **TO** <addr> ADR **S**). With most DDB software packages, you can limit the trigger event still further with one of the commands **WRITE, READ**, or **FETCH**. Then you can set up the UniLab to trigger when the processor tries to <u>fetch</u> from outside of ROM, or <u>write</u> outside of RAM.

Choice five will capture a trace that starts to record when one address appears and excludes some other range of addresses from the trace (**F5** or **ONLY NOT** <addr> **TO** <addr> **ADR AFTER** <trigger addr> **ADR S**).

With RESET enabled, the UniLab resets your target processor when the analyzer is started-- that is, whenever an **S** is issued. With RESET disabled, the analyzer will search the program "in progress" for the trigger whenever an **S** is issued. Toggle RESET on and off with choice six (**F6** or **RESET** and **RESET'**).

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press **F6** from the MAIN MENU to get the sixth sub-menu:

```
                    DEBUG MENU

                   F1    SET A BREAKPOINT TO ESTABLISH DEBUG CONTROL
                   F2    RESUME EXECUTION TO A BREAKPOINT
                   F3    EXECUTE THE NEXT STEP (WON'T FOLLOW JUMPS &
                     BRANCHES)
                   F4    SINGLE STEP or IMMEDIATE DEBUG CONTROL
                   F5    GO TO AN ADDRESS WITH A BREAKPOINT SET
                   F6    GO TO AN ADDRESS AND EXIT THE DEBUG
                  F10    RETURN TO MAIN MENU
```

Explanation

This sub-menu provides DEBUG features that establish DEBUG control, display internal registers and single-step through your program. The menu selections do not provide you with all the features available in command mode.

Use the first choice to set a breakpoint on the first address of an instruction (**F1** or **RESET** <addr> **RB**). This command causes the processor to start executing the program from the beginning, and then disables RESET. When the program reaches the breakpoint and shows the internal register display, you have established DEBUG control.

With DEBUG control established, use the second choice to set a new breakpoint and free the processor from debug control, so that it can run until it reaches the new breakpoint (**F2** or <addr> **RB**). This command does not cause your program to start again.

Once at a breakpoint, step through instructions one at a time with a display of the internal registers at each step (**F3** or **N**). This command will not follow jumps, calls or branches.

Use choice four to either establish DEBUG control in the first place, or to step through your code (**F4** or **NMI**). This feature works on both "in-line" code and on jumps, calls, etc. . It uses the hardware interrupt of your processor to single-step (except with the 8088/86 DDB package).

From a breakpoint, alter the program counter and then set a breakpoint with menu choice five (**F5** or <New PC> <addr> **GB**).

Use option six to change the program counter and then release the program from DEBUG control without a breakpoint set (**F6** or <New PC> **G**).

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press F7 from the MAIN MENU to get the seventh sub-menu:

```
                STIMULUS MENU

                  F1   SET A STIMULUS BIT
                  F2   RESET A STIMULUS BIT
                  F3   DEFINE ALL 8 STIMULUS BITS
                 F10   RETURN TO MAIN MENU
```

Explanation

Use the stimulus generator to send signals out through the PROM PROGRAMMER socket. These eight signals, which you control from your keyboard, can replace the usual prototype board dip switch. This sub-menu provides all the stimulus features.

Use the first choice to set (change to high signal) a single bit of the stimulus output (**F1** or <bit number> **SET**).

Use the second choice to "reset" (change to low signal) a single bit of the stimulus output (**F2** or <bit number> **RES**).

Use the third choice to specify all eight bits at once, by setting the output to the value of two hexadecimal digits (**F3** or <byte value> **STIMULUS**).

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press **F8** from the MAIN MENU to get the eighth sub-menu:

```
                TOOLKIT MENU

                   F1   DISPLAY PINOUT OF 2716 PROM
                   F2   DISPLAY PINOUT OF 2764 PROM
                   F3   DISPLAY PINOUT OF PROCESSOR AND UniLab CABLE
                   F4   DISPLAY CATALOG OF AVAILABLE PINOUTS
                   F5   DISPLAY ASCII TABLE
                  F10   RETURN TO MAIN MENU
```

Explanation

The toolkit menu provides valuable reference screen displays.

The most valuable is the chip diagram that shows how the cables connect to your processor (**F3** or **PINOUT**).

The other choices show you the chip diagram of two PROMs (**F1** and **F2**), a catalog of commands you can use to get the chip diagrams of many processors (**F4**-- but note that these commands are not functional in menu mode), and a display showing the hexadecimal codes for every ASCII character (**F5**).

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press **F9** from the MAIN MENU to get the ninth sub-menu:

```
                 PROM READER MENU

                 F1   READ 2716/48016 - use PM16
                 F2   READ 2532 - use PM16
                 F3   READ 2732 - use PM32
                 F4   READ 2764 - use PM64
                 F5   READ 27128- use PM64    (PM56 for 27128A)
                 F6   READ 27256- use PM56
                 F7   READ 27512- use PM512
                 F9   Go to Prom Programmer Menu
                F10   RETURN TO MAIN MENU
```

Explanation

Use this sub-menu to load a program from an EPROM into the UniLab's emulation ROM. Use the ninth option to get write into an EPROM with the PROM PROGRAMMER MENU (press **F9**).

Menu Choice	Command
F1	RPROM
F2	R2532
F3	R2732
F4	RPROM
F5	RPROM
F6	R27256
F7	R27512

Each entry on the menu tells you which Personality Module (PM) you need to use while reading from the PROM. The PM nestles next to the EPROM PROGRAMMER socket, and alters control signals as necessary for each PROM.

You can perform all EPROM programming and reading from within the menu system. The commands, though available, are only necessary for programming EPROMs within macros.

See Appendix G for more information on EPROMs and commands.

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press **F9** from the PROM READER MENU to get the first PROM PROGRAMMER sub-menu:

```
                 PROM PROGRAMMING MENU #1


   F1    PROGRAM A 2716     (use PM16 personality module)
   F2    PROGRAM A 2532     (use PM16 personality module)
   F3    PROGRAM A 2732A    (use PM32 personality module)
   F4    PROGRAM A 2764A    (use PM64 personality module)
   F5    PROGRAM A 27128A   (use PM56 for A version)
   F6    PROGRAM A 27256A   (use PM56 personality module)
   F7    PROGRAM A 27512  (use PM512 personality module)
   F9    Next page of Prom Programming Menu
  F10    RETURN TO MAIN MENU
```

Explanation

the two PROM PROGRAMMING menus can program any EPROM that Orion supports.

Menu Choice	Command
F1	P2716
F2	P2532
F3	P2732A
F4	P2764
F5	P2764
F6	P27256
F7	P27512

Press **F9** to get the second page of the menu.

See Appendix G if you want more information on EPROMs and commands.

As in all sub-menus, **F10** returns you to the MAIN MENU.

Press **F9** from the first PROM PROGRAMMER sub-menu to get the second PROM PROGRAMMER sub-menu:

```
                PROM PROGRAMMING MENU #2

             F1   PROGRAM A 27C16   (use PM16 personality module)
             F2   PROGRAM A 48016   (use PM16 personality module)
             F3   PROGRAM A 27C32   (use PM16 personality module)
             F4   PROGRAM A 2764    (use PM64 personality module)
             F5   PROGRAM A 27128   (use PM64 personality module)
             F6   PROGRAM A 27256   (use PM56 personality module)
             F9   RETURN TO PROM READER MENU
            F10  RETURN TO MAIN MENU
```

Explanation

These two PROM PROGRAMMING menus provide you with all the EPROM burning tools you need.

Menu Choice	Command
F1	PD2716
F2	P48016
F3	P27C32
F4	PD2764
F5	PD2764
F6	PD27256

Press **F9** to return to the PROM READER MENU.

See Appendix G if you want more information on EPROMs and commands.

As in all sub-menus, **F10** returns you to the MAIN MENU.

2. The Command Mode

The UniLab command language has several significant advantages over the menu system:

more power,
more flexibility,
access to the mode panels,
the split screen capability,
and the other special features.

The steps you go through stay the same, whether you use commands or the menu mode:

1. enable memory,
2. load in a program,
3. set trigger specs and examine traces.

The big difference is that in command mode you make up any trigger specification you want.

The bulk of this section covers the heart of the UniLab--how to use the UniLab trigger commands to build a trigger specification.

Related information

Turn to the Target Application Note of your DDB software package for information on the processor-specific aspects of the the UniLab software.

For more help with interpreting the trace display, consult section One of the **Operations In Detail** chapter in the UniLab Reference Manual.

If, after going through the information here, you want to know more about the trigger commands, turn to section Four of that chapter.

The Command Language

In command mode, the line of text you type in is interpreted whenever you press the carriage-return (Enter) key.

Common commands are assigned to some function keys. Other function keys summon help screens. Altogether forty soft-keys are available to you on the function keys. Several more features are assigned to the numeric key pad (cursor keys).

Any command that is still on the screen can be edited and reissued. Even commands that have scrolled off the screen can be reclaimed with the history feature, then edited and reissued. These special features are covered in the Special Features section of this chapter, along with the other features assigned to cursor and function keys are covered in the Special Features section of this chapter.

Enter command mode

You are normally in command mode when you first start the program. In the menu system, go to the main menu and press function key 10 (**F10**) to re-enter the command mode.

Notation conventions

Throughout the manual, **UPPER CASE BOLDFACE** type represents commands. The UniLab program itself accepts commands in any mixture of upper and lower case.

Parameter conventions

Many commands must be preceded by one or more parameters. The numbers you enter are interpreted as hexadecimal numbers.

Most commands will give an error message if you specify too few parameters.

Commands that require a file name, such as **SYMSAVE**, **BINLOAD**, etc., will prompt you for the file name if you do not enter it.

Other number bases

Though numbers are usually entered in hexadecimal, you can also use decimal or binary if you precede the number with **D#** or **B#** respectively.

Flexible entry of parameters

Entering the parameters before the command allows unlimited flexibility in how you enter the number. For example, you can enter a symbol, the number itself, or an equation (using reverse polish notation) which uses as many numbers and symbols as you like.

Spaces

Spaces are used to separate commands. It doesn't matter how much "white space" (blank spaces)is used, as long as there is at least one space. The absence of space will cause the UniLab software to misunderstand your commands.

Multiple commands can be strung together on a single line if desired.

"Not recognized"

If the UniLab software does not understand the command or parameter that you have entered, it will respond with the message "not recognized," and a row of carats (^) pointing to the first word that it does not recognize.

Usually this is the result of a typing error. You can either retype the command, or use the command line editor to edit and re-issue the command (see the Special Features section of this chapter).

Examples

The examples in the rest of this section were mostly drawn from the Command Reference chapter of the UniLab Reference Manual.

The Simple Trigger

Most of your test and diagnosis work can be done with simple triggers. The simplest trigger specification has two parts:

a description of the bus cycle to trigger on (the trigger bus cycle), and

the number of cycles to record after the trigger cycle (the delay counter value).

The flow chart on the next page illustrates the simple trigger.

As soon as you start the UniLab's bus state analyzer, it will start looking for the trigger bus cycle, meanwhile recording every cycle.

Trigger wait status line

If the search for trigger takes more than a few hundred milliseconds, the UniLab will give you the trigger wait status.

```
TRIGGER Wait Status: DELAY Count= 00A0 PASS Count= NONE
```

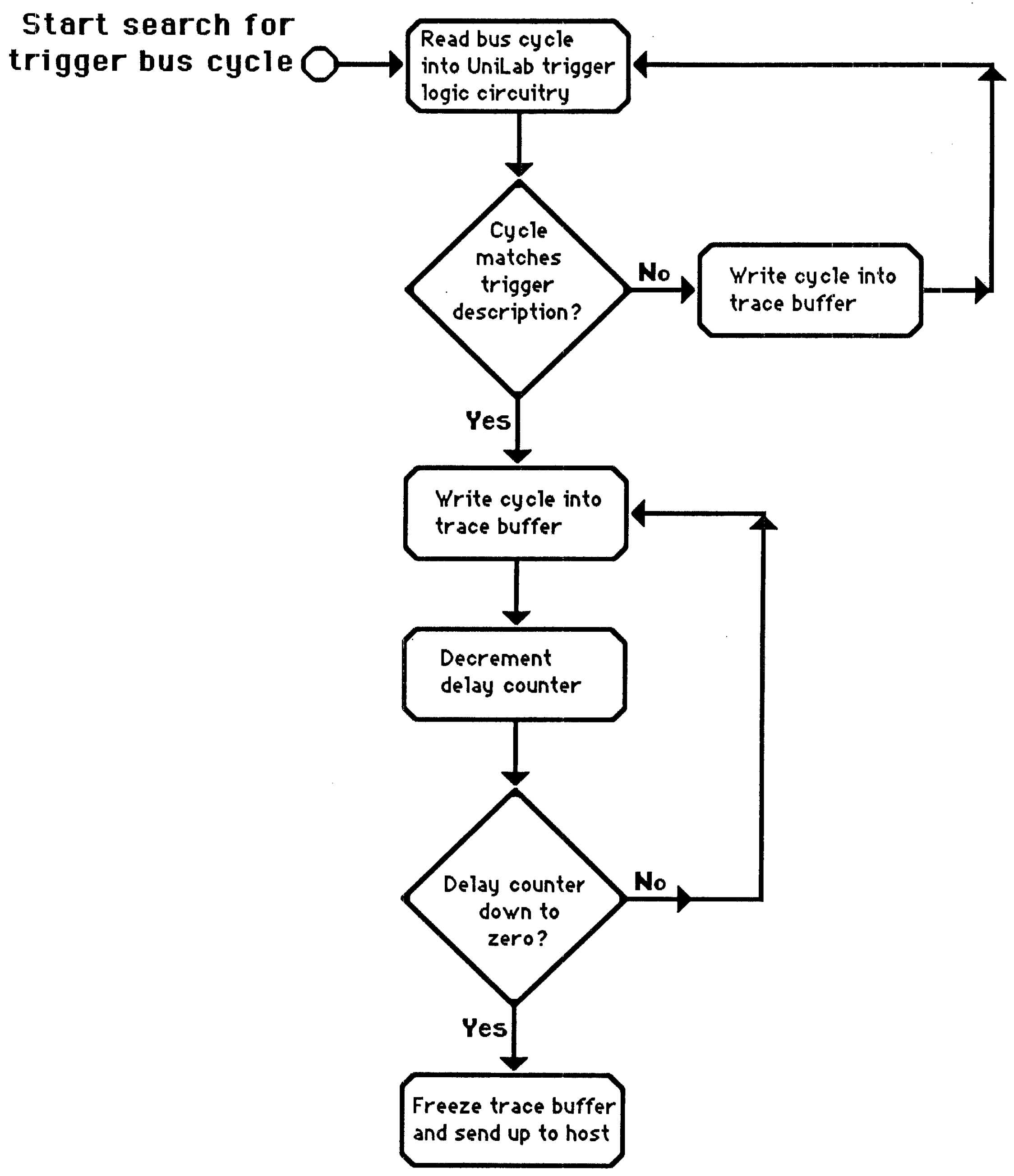
This line shows you the status of both the delay counter, explained below, and the "pass counter" which is explained in the discussion of additional qualifier capabilities.

Post-trigger actions

When the UniLab finds the trigger cycle, it continues to record bus cycles, but now counts them by decrementing the delay counter each time it records a cycle. When the requested number of cycles after the trigger have been recorded, the UniLab will freeze the trace buffer and send it up to the host.

Coming attractions

The page following the flow chart has an example of a simple trigger spec. Then come sub-sections which tell you how to specify the number of delay cycles and how to describe the bus cycle that you wish to trigger on. After the presentation of cycle description commands come three more flow charts. One illustrates a filtered trigger specification, the other two show the qualifier logic.


Start search for trigger bus cycle
Read bus cycle into UniLab trigger logic circuitry
Cycle matches trigger description?
No
Write cycle into trace buffer
Yes
Write cycle into trace buffer
Decrement delay counter
Delay counter down to zero?
No
Yes
Freeze trace buffer and send up to host

Flow chart of
Simple Trigger

Example: simple trigger

You can specify a simple trigger on address 23 with the command string:

NORMT 23 ADR

and then use the command **TSTAT** to examine the trigger status. The UniLab will not start looking for the trigger until you start it with the command **S**.

When you look at the trigger status, you will see a display like this:

TSTAT

```
Analyzer Trigger Status
23 ADR
A0 DCYCLES   0 QUALIFIERS
```

The "23 ADR" is the description of the trigger bus cycle. A0 is the number of cycles to record in the trace buffer after the trigger (the delay count).

So when you tell the bus state analyzer to start, by typing **S**, the UniLab will look for a bus cycle which has an address of 23, and have its delay counter set to A0. Since you have not specified any other limitations, the UniLab does not care what value is on the data lines, control lines, or miscellaneous inputs.

Once the UniLab sees a bus cycle with address 23, it will start decrementing the delay counter for every cycle it clocks in. When A0 (decimal 160) additional cycles have been clocked in, the UniLab will freeze the trace buffer and send it up to the host, where it will be displayed.

The trigger cycle is always labeled as cycle zero. With **NORMT**, the trace will contain 9 cycles before the trigger and A0 cycles after it.

Specify number of delay cycles

In the example above, the delay counter is set to A0. That value was set by the command **NORMT**.

The three commmands **NORMT, NORMM** and **NORMB**, serve a double purpose. They each:

clear out the previous trigger specification, and
set the value of the delay counter.

As you might guess from the names, these three words set the value of the delay counter so that the trigger cycle will end up at the top, middle or bottom of the trace buffer.

Command	Delay	
	Hex	Decimal
NORMT	A0	160
NORMM	55	85
NORMB	04	4

These three commands will serve your needs almost all of the time.

You can set the delay counter to any value you like, though you rarely will have need to do so. When you do want to set it to a non-standard delay, use the command:

```
<number of cycles>  DCYCLES
```

after one of the **NORMx** words.

<u>Describe the trigger bus cycle</u>

The UniLab reads 48 bits from the target system every bus cycle. These are organized as six 8-bit groups.

The UniLab can trigger on anything from just one value on a single input group, up to multiple ranges of values, on <u>each</u> of the six input groups.

Coming attractions

The next three pages show how to describe a bus cycle. That background material is followed by discussions of filtered traces and qualifiers. Then this command language section ends with several pages of trigger specification examples.

The input groupings

On every target system bus cycle, the UniLab software reads six bytes of inputs from the target system's bus:

two bytes of address,
two bytes of data (if a 16-bit data bus),
a byte of control values, and
a byte of miscellaneous inputs.

You can set a trigger on any of the bytes separately or in combination, using the names that the UniLab assigns to the groupings. With processors that have an 8-bit data bus, the high byte of the data inputs can be used as an additional byte of miscellaneous inputs.

The names

Each of the groupings of inputs is referred to using the same descriptive name that labels it on the trace display:

CONT   ADR   DATA   HDATA   MISC

Each of these names labels one byte of the inputs into the UniLab, except for ADR, which labels 2 bytes. You can refer separately to the two byts of the address with LADR and HADR. With 8-bit data bus processors, HDATA column appears on the trace next to the MISC column. 16-bit processors show <u>both</u> the HDATA and the DATA byte under the DATA column.

Set a trigger on a single value

To set a trigger on a single value, you first clear out any previous definitions with one of the **NORMx** commands, and then define the new trigger with a value, followed by the name of one of the groupings:

<16-bit value> **ADR**
to trigger on a 16-bit address (A0 to A15).

<8-bit value> **CONT**
to trigger on cycle type and on A16-A19.

<8-bit value> **DATA**
to trigger on the data byte.

<8-bit value> **HDATA**
to trigger on the upper byte of a 16-bit data bus or on anything you like with an 8-bit data bus.

<8-bit value> **MISC**
to trigger on anything you like. (Usually target system inputs and outputs.)

Set a trigger on a range

Set a trigger on a range of values on any input group by using <value> **TO** <value> in place of a single value. For example:

10 TO 1A DATA

Set a trigger outside a range by preceding the range with the keyword **NOT**. For example, to trigger when a program goes outside some acceptable area of memory, you would make this part of your trigger bus cycle description:

NOT 0 TO 1000 ADR

Set trigger on multiple input groups

The UniLab maintains independent truth tables for each 8-bit input group, so you can trigger on a bus cycle that matches certain values on each of several input group.

For example, you can tell the UniLab to trigger only when it sees 23 on the address lines, 61 on the data lines, 4F on the control lines and 33 on the miscellaneous inputs:

```
NORMM 23 ADR   61 DATA   4F CONT   33 MISC
```

You can specify ranges or, as explained below, multiple values for each of the input groups.

Set trigger on any of several values on a single input group

Use the keyword **ALSO** to add a value or a range to the trigger description within an input group. For example, you can trigger on any one of three data values:

```
NORMB   12 DATA   ALSO 50 DATA   ALSO 78 DATA
```

If you leave out the ALSOs, each use of DATA would clear out the data truth table. You would end up with **78 DATA** as the whole of the trigger cycle description.

<u>The Filter Trigger</u>

Use the filter trigger to capture a trace that includes only cycles that match the trigger specification.

The filter trigger specification has two parts, very similar to the simple trigger specification:

a description of the bus cycle to record
(the trigger bus cycle), and
the number of trigger cycles to record
(the delay counter value).

The flow chart on the next page illustrates the filter trigger.

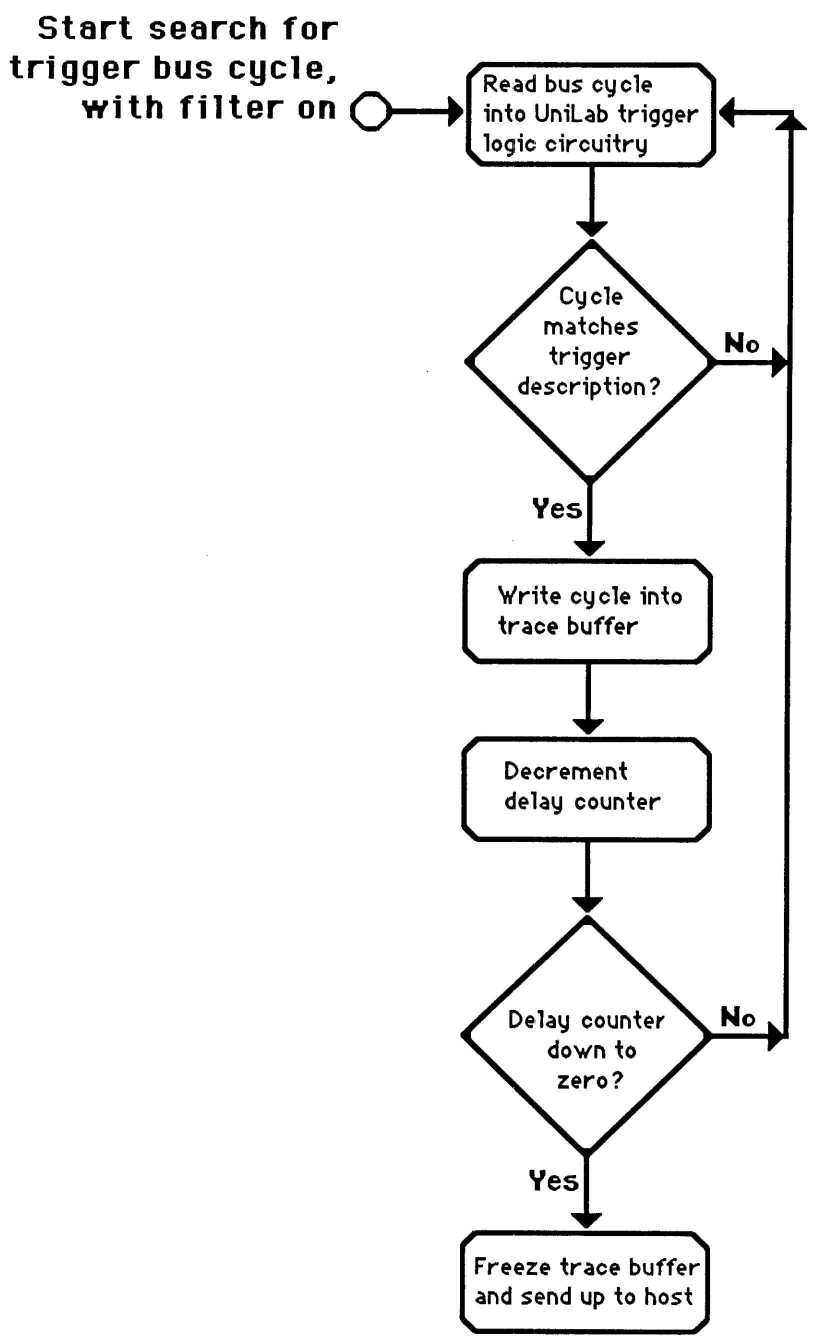
With the filter trigger, the trace buffer will only record cycles that match the trigger specification. It will always record A9 (169 decimal) occurrences that match the trigger bus cycle description.

When you filter, the bus cycles will be marked with a lower-case "f."

Specify a filter trigger

You specify a filter trigger with the same commands that you use to specify the simple trigger, except that you start with the command **ONLY** instead of one of the **NORM**x commands.

ONLY sets up the UniLab to record only those cycles that match the trigger description.


Start search for trigger bus cycle, with filter on
Read bus cycle into UniLab trigger logic circuitry
Cycle matches trigger description?
No
Yes
Write cycle into trace buffer
Decrement delay counter
Delay counter down to zero?
No
Yes
Freeze trace buffer and send up to host

Flow chart of
Trigger with filter on

Example: filter trigger

You can specify a filter trigger on address 23 with the command string:

ONLY 23 ADR

and then use the command **TSTAT** to examine the trigger status. You will see a display like this:

TSTAT

```
Analyzer Trigger Status
23 ADR
A9 DEVENTS  MISC' FILTER  0 QUALIFIERS
```

The "23 ADR" is the description of the trigger bus cycle. A9 is the number of cycles to record in the trace buffer. As you can see in the flow chart, only the bus cycles that match the trigger description will get written into the trace buffer.

When you tell the bus state analyzer to start (type **S**) the UniLab will look for a bus cycle which has an address of 23. The delay counter will be set to A9.

Since you have not specified any other limitations, the UniLab does not care what value is on the data lines, control lines, or miscellaneous inputs.

Once the UniLab sees a bus cycle with address 23, it will record that cycle in the trace buffer and decrement the delay counter. Since the delay counter is not yet zero, the Unilab will once again start looking for a bus cycle with address 23. When A9 (decimal 169) trigger cycles have been found, clocked in, and counted, the UniLab will freeze the trace buffer and send it up to the host, where it will be displayed.

You should ignore the first cycle in the trace buffer, as it is a "leftover" that does not necessarily match the trigger specification.

Additional filter capabilities and commands

You can, instead of **ONLY**, use one of the commands **1AFTER**, **2AFTER** and **3AFTER**. These commands clear out the previous trigger specification and then set up the UniLab to capture the trigger cycle and the one, two, or three cycles which follow.

These triggers capture a trace that shows what happened just after each of a series of trigger cycles. For example:

2AFTER 4500 ADR

will show you the bus cycles which access address 4500, and the two cycles which follow each access to 4500.

This would be useful if 4500 were the address of a variable, and you wanted to see which sections of your program write into it; or, if 4500 were the address of a return instruction, the next cycles would show the return address getting popped off the stack.

If you use **TSTAT** to look at the specification set up by these commands, you will note that they use "qualifiers," and set up a certain number of **PEVENTS** rather than **DEVENTS**.

The internal mechanism is only slightly different:
nothing gets recorded in the trace buffer until a cycle matches the trigger specification.
When a matching cycle is found, both that cycle and the one, two or three following are recorded.

The trigger status looks quite different, as does the flow chart.

Filter primitives

Though you will probably use only the "prepackaged" filter commands, the filter primitives are available to you. You can set up some number of **DEVENTS** or **PEVENTS** and enable the filter with **MISC' FILTER** or **CONT FILTER**. For more information on these commands, consult the glossary entries, either on-line or in Chapter Seven of the UniLab Reference Manual

The Qualifier Trigger

Use the qualifier trigger specify a sequence of bus cycles that must be seen before the UniLab starts to search for the trigger cycle. The "qualifying sequence" can be up to FFFF repetitions of up to 3 bus cycles.

The usual "one qualifier" trigger specification will have only one additional part:

> a description of the qualifier bus cycle that must show up before the UniLab starts to search for the trigger (the Q1 bus cycle).

You specify the Q1 cycle with the normal input group description words. However, you use the keyword **AFTER** between the description of the trigger and the description of the qualifier. For example:

NORMM 18FE ADR AFTER 47 CONT 20 ADR

sets up a trigger on address 18FE-- but the UniLab will not start to look for this trigger until after it sees the qualifier cycles of 47 on the control lines and 0020 on the address lines.

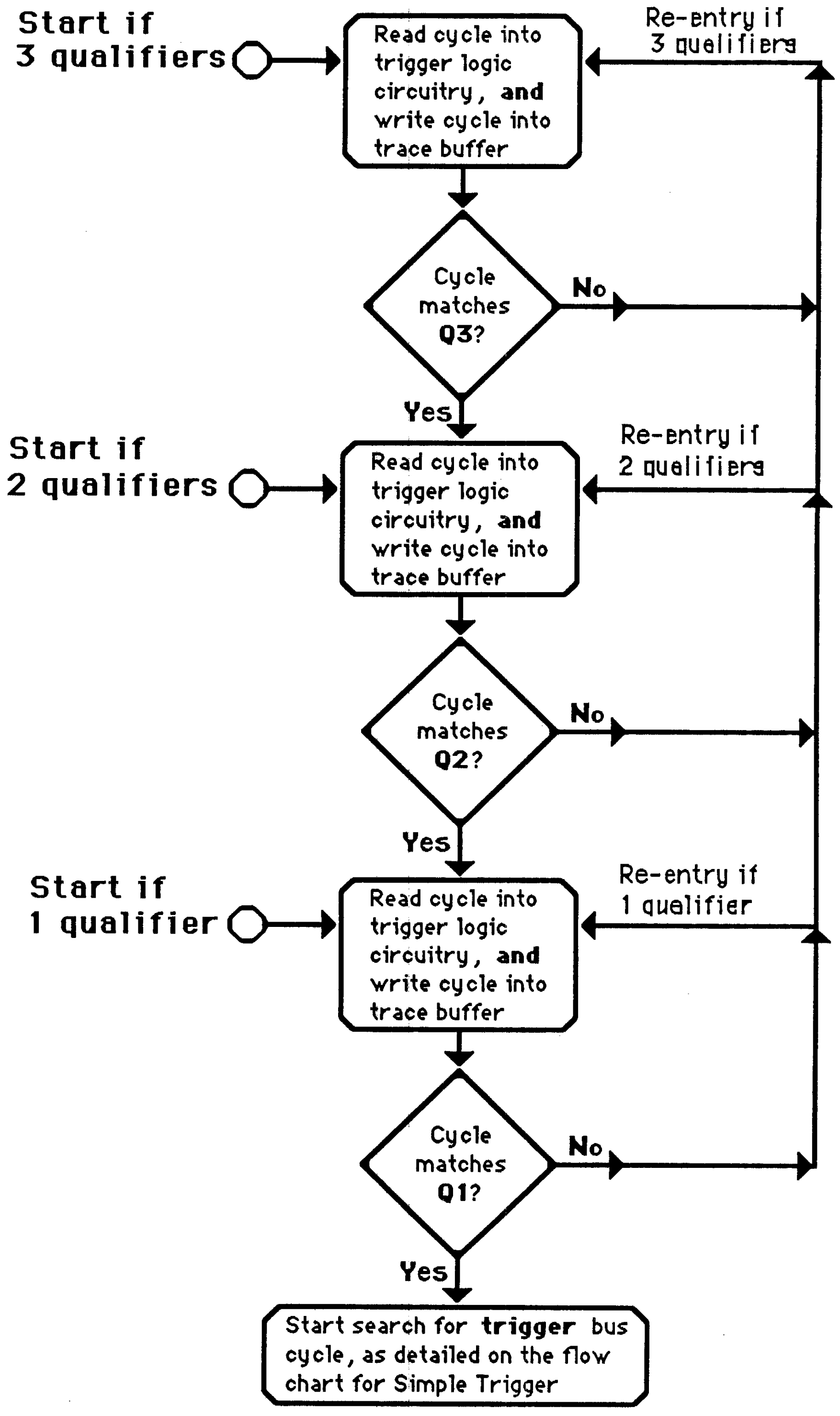
You can specify up to three qualifiers, by repeating the keyword **AFTER**. The qualifier cycles must occur one immediately after another, as indicated on the flow chart on the next page. For example, if you have three qualifiers:

NORMB 10 ADR AFTER 40 DATA AFTER 20 DATA AFTER 35 DATA

then the UniLab will start out searching for qualifier three (Q3), which is 35 on the data lines. As soon as it finds Q3, it will look, during the next bus cycle, for qualifier two. If that next bus cycle does not have 20 on the data lines, the UniLab will start to look for Q3 again.

When the UniLab does find Q3 followed by qualifier two (Q2), it will look, during the next cycle, for Q1. And if it does not find 40 on the data lines during that one cycle, it will go back to searching for Q3.

Once the UniLab finds the complete qualifier sequence, it will start to look for the trigger. The UniLab will continue to search for the trigger cycle until it is found.


Start if 3 qualifiers
Read cycle into trigger logic circuitry, and write cycle into trace buffer
Re-entry if 3 qualifiers
Cycle matches Q3?
No
Yes
Start if 2 qualifiers
Read cycle into trigger logic circuitry, and write cycle into trace buffer
Re-entry if 2 qualifiers
Cycle matches Q2?
No
Yes
Start if 1 qualifier
Read cycle into trigger logic circuitry, and write cycle into trace buffer
Re-entry if 1 qualifier
Cycle matches Q1?
No
Yes
Start search for trigger bus cycle, as detailed on the flow chart for Simple Trigger

Flow chart of Qualifiers

Example: qualifier trigger

You can specify a qualifier trigger with the command string:

```
NORMB   23 ADR   AFTER FF CONT   40 DATA
```

and then use the command **TSTAT** to examine the trigger status. The UniLab will not start looking for the trigger until you start it with the command **S**.

When you look at the trigger status, you will see a display like this:

```
TSTAT

Analyzer Trigger Status
23 ADR
AFTER  FF CONT  40 DATA
 4 DCYCLES   1 QUALIFIERS    Q1
```

The "23 ADR" is the description of the trigger bus cycle. The keyword AFTER indicates that "FF CONT 40 DATA" is a qualifier. The "1 QUALIFIERS" on the last line of the trigger status shows that there is one qualifier. The "Q1" indicates that any additional cycle description commands you enter will be part of the Qualifier One description.

When you tell the bus state analyzer to start, the UniLab will look for the qualifier sequence, and then when it finds that sequence, will go to the simple trigger flow chart.

In this example, the UniLab will start out looking for a bus cycle which has data of 40 and has FF on the control lines.

Once the UniLab sees that bus cycle, it will start to search for a bus cycle with address 23. When that trigger cycle is found, the UniLab will delay 4 cycles, freeze the trace buffer and send it up to the host to be displayed.

Additional qualifier commands

These additional commands allow you to change "context" from qualifier to trigger and back again, and to turn qualifiers off and back on.

Use the four commands **TRIG, Q1, Q2** and **Q3** to switch context from one level to another. For example, after specifying the following trigger:

```
NORMM 20 ADR AFTER 34 DATA
```

you can go back and alter the trigger cycle description with this command:

```
TRIG ALSO 30 TO 60 ADR
```

Use the command <number> **QUALIFIERS** to change the number of qualifiers that the UniLab will use when you start the analyzer. You can specify between zero and three qualifiers. For example, after the above sequence of commands, you could tell the UniLab not to use the qualifier (that is, to start out looking for the trigger cycle) :

```
0 QUALIFIERS
```

and then, later, tell it to use the qualifier again:

```
1 QUALIFIERS
```

Additional qualifier capabilities

You can use additional qualifier commands to put a delay between qualifiers and the trigger cycle or to search for the qualifier sequence multiple times.

By default, the UniLab will look for the complete qualifier sequence only once, and then will start to search for the trigger cycle after a delay of zero bus cycles. You can change either one of these defaults, with **PEVENTS** and **PCYCLES**. Only one of these two can have a non-default values.

For the sake of simplicity, these capabilities were not shown on the previous qualifier flow chart. This "Pass count" logic is detailed in the chart on the next page.

If you change the delay between qualifiers and trigger, you can only search for one sequence of qualifiers. And if you search for the qualifier sequence several times, you cannot have a delay between the qualifier sequence and the trigger cycle.

For example, along with the following specification:

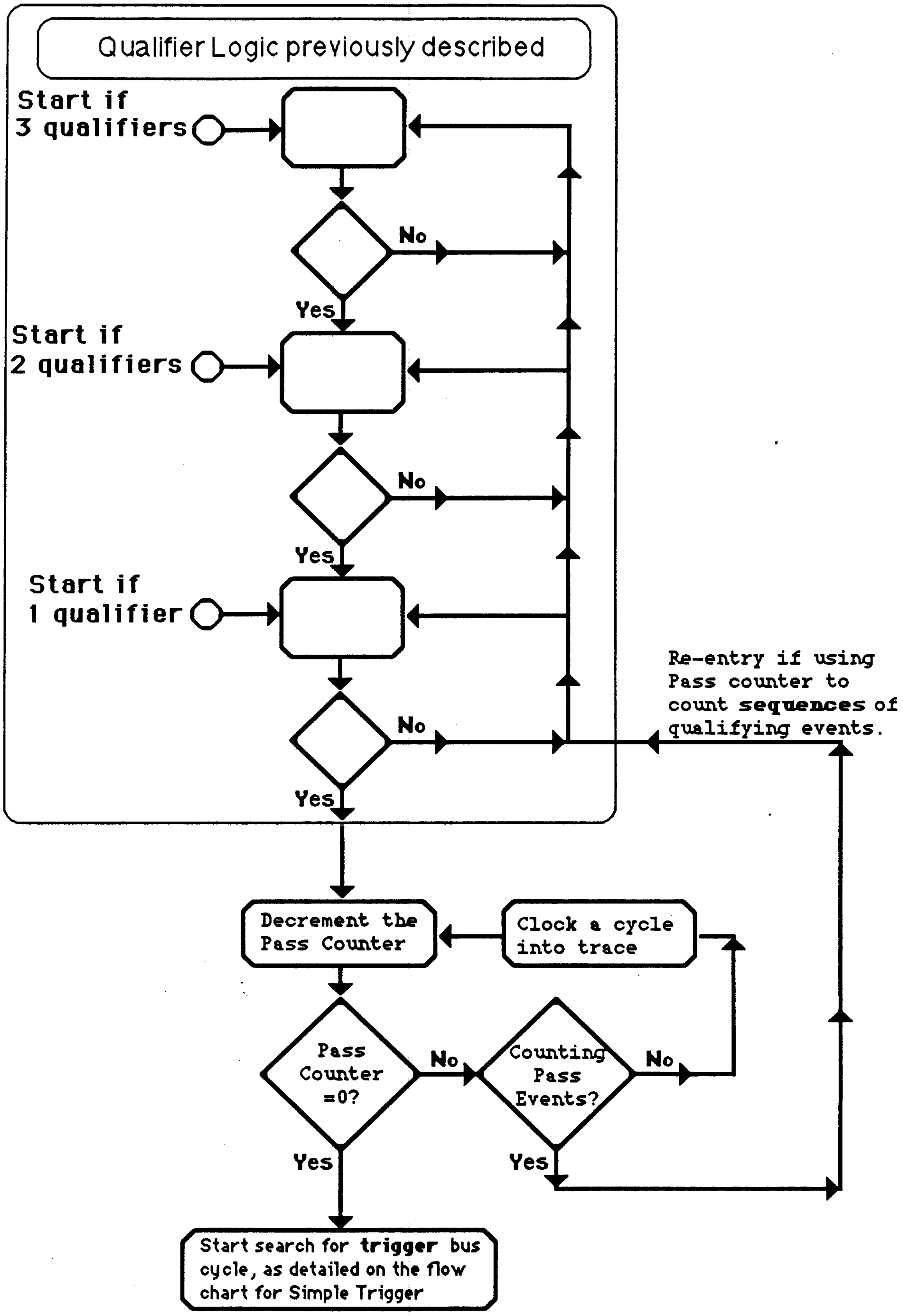
NORMB 10 ADR AFTER 34 DATA AFTER 45 DATA

you can further specify either 30 repetitions of the qualifying sequence (and the default of a zero cycle delay between qualifier sequence and trigger):

30 PEVENTS

or a 50 bus cycle delay between the qualifier sequence and the trigger (and the default one instance of the qualifier sequence):

50 PCYCLES


Qualifier Logic previously described
Start if 3 qualifiers
No
Yes
Start if 2 qualifiers
No
Yes
Start if 1 qualifier
No
Yes
Re-entry if using Pass counter to count **sequences** of qualifying events.
Decrement the Pass Counter
Clock a cycle into trace
Pass Counter =0?
No
Counting Pass Events?
No
Yes
Yes
Start search for **trigger** bus cycle, as detailed on the flow chart for Simple Trigger

Trigger Specification Examples

The remainder of this section contains trigger specification examples with explanations. These examples help you get an idea of the varied capabilities of the UniLab trigger logic.

Simple trigger examples: one input group

Trigger on a value

1205 AS
the abbreviated way to clear out the previous trigger, specify a trigger on address 1205, and start the analyzer.

NORMT 1205 ADR S
the non-abbreviated version of the above command.

NORMT 12 DATA S
clears all previous settings with NORMT and sets up a trigger for data input 12. S starts the analyzer.

Trigger on a range

NORMT 110 TO 138 ADR S
triggers when any one of a range of addresses appears on the bus.

NORMT 10 TO 21 DATA S
triggers when any one of a range of data values appears on the bus.

Trigger on a cycle type with the CONT grouping

With some processors the UniLab can distinguish between bus cycle types. The DDB packages for these processors include predefined macros which set up the CONT trigger specification to trigger on cycle types: **READ, WRITE, FETCH, I/O,** etc.

NORMT WRITE S
triggers when the processor writes to any address.

Trigger outside a range

NORMT NOT 0 TO 100 ADR S
triggers when any address outside of a range appears on the bus

Simple trigger examples: multiple input group

Trigger on an ADR and DATA combination

NORMT 1E DATA 1200 ADR S
triggers on 1200 address and 1E data.

Trigger on a DATA and HDATA combination

NORMM 23 HDATA 17 DATA S
triggers when 2317 appears on the data lines.

Trigger on a CONT and ADR combination:
fetch from outside program memory

NORMT FETCH NOT 0 TO 7FF ADR S
triggers if the program tries to fetch an instruction from outside the 0 to 7FF range. FETCH is a command with a processor-specific definition. It sets up a trigger on the CONT grouping. Not supported on some processors.

Trigger when "bad" values are associated with any of several addresses.

NORMM 10 DATA ALSO 5 DATA ALSO 3 DATA 1200 ADR S
sets the analyzer to trigger when the data is 10 or 5 or 3 and the address is 1200 .

Trigger on any member of a complicated set of addresses

NORMT 12 HADR 34 LADR ALSO 10 LADR ALSO 5 LADR
sets up the analyzer to trigger on any of the addresses 1234, 1210, or 1205.

Filter trigger examples

Save in the trace only the cycles that meet the trigger specification-- or save only those cycles and the one, two,or three that follow.

Trigger cycles only

ONLY 0100 ADR S
records only those cycles that accesses address 0100.

ONLY 10 TO 30 DATA 8FD0 ADR S
records only those cycles that access this RAM address when the data is between 10 and 30.

One following cycle each trigger

1AFTER 1200 ADR S
shows only those cycles with the address 1200 and one bus cycle following.

Two following cycles each trigger
2AFTER 1200 ADR S
shows two bus cycles following.

Two following cycles each trigger
2AFTER 1200 ADR S
shows two bus cycles following.

Filter to exclude

By including NOT in the trigger spec, you can produce a filtered trace that excludes certain cycles, rather than one that only shows certain cycles.

You can filter out a single address, or a single data value, but usually will want to filter out a range of addresses. That way you can see a trace that shows everything except some segment of code.

ONLY NOT 50 TO 100 ADR S
shows only those cycles that are not accessing the memory in the address range 50 to 100.

Qualifier trigger examples

Search for a "qualifier" sequence of bus cycles, then look for the trigger cycle.

Start searching after an address is seen

NORMT 100 ADR AFTER 535 ADR S
will trigger on address 100 anytime after address 535 is seen on the bus.

Multiple qualifiers

You can specify up to three qualifiers. When you have more than one qualifier, they must appear on the bus one immediately after another.

The trigger can occur anytime after all the qualifiers have been found.

NORMT 100 ADR AFTER 535 ADR AFTER 3F DATA S
You can add a second qualifying event-- which must occur earlier than the first. Now address 535 must be immediately preceded by data 3F hex before UniLab will look for address 100 on the bus.

Qualifiers without triggers

If you specify a qualifier but no trigger, the UniLab will trigger on the very first cycle after the qualifiers have been seen.

NORMB AFTER 1500 ADR AFTER 235 ADR S
triggers as soon as address 1500 immediately follows address 235. This would be useful when instruction at address 235. The qualifier will only be satisfied when the jump is taken. If that jump is <u>not</u> taken, the UniLab will again start looking for address 235

Examples that combine qualifiers with filters

This very useful combination allows you to set up a specification that triggers on one condition and filters on another.

Your trace will be filtered, but the trace buffer will not start to fill up until after the qualifier appears on the bus. That way you can, for example, start your trace when a certain routine is executed, and make it a filtered trace that shows only memory reads.

Show reads after some routine is reached

ONLY READ AFTER FETCH 1235 ADR S
show only reads from RAM, starting after the code at 1235 is fetched. NOTE: READ and FETCH are processor specific macros, not defined on some processors.

Exclude a loop from trace, start trace after some address

ONLY NOT 120 TO 135 ADR AFTER 750 ADR S
triggers on address 750, excludes from the trace the routine at addresses 120 through 135.

3. The UniLab in the DOS environment

The information in this brief section tells you how to use the batch file facility of DOS and the command tail feature of the UniLab software to help you get your work done more easily and with less tedium.

Batch files can automate the sequence of calls to different software packages that you must make when you:

alter source code,
produce object files,
and then call the UniLab software.

The command tail feature of the UniLab can be used in combination with a batch file to perform the same operation or series of operations every time you call the UniLab software.

This section first introduces the command tail and batch files, then shows you how to combine the two capabilities. Last, the use of macros is touched upon.

Command tail

The UniLab program allows you to include a "command tail" on the DOS command line, each time you call up the software:

```
C> ULZ80 <UniLab commands>
```

The UniLab command(s) that you include on the DOS command line will be executed after the instrument is initialized. For example, if you want to go into menu mode as soon as you enter the program, you would type in:

```
C> ULZ80 MENU
```

The command tail can be up to 80 (decimal) characters long.

Simple batch files

If you find yourself executing the same instruction every time you start up your UniLab software, you might want to include that command in a batch file, as a command tail.

Batch files are simply DOS text files that contain commands. Their names must end in .BAT. The easiest way to make one is to copy from the screen (CONsole) into a file. For example, if every time you start the UniLab software you want to load in the contents of a binary file, you could make a batch file to do that for you:

```
 C> COPY CON Uni.BAT
ULZ80   0 7FF BINLOAD MYPROG.BIN
^Z
    1 File(s) copied
C>
```

To finish the "copy" operation press CTRL-Z followed by a carriage return.

Once the batch file is made, you call it by name (leaving off the .BAT extension):

```
C> UNI
```

More sophisticated batch files

You can create batch files that call several different programs, one after another. For example, you probably go through the same cycle of procedures time after time:

1) edit your source file to correct an error.
2) assemble (and link) your program.
3) enter the UniLab program and load your newly altered program into emulation memory.

You can save yourself time by putting all this into a batch file. For example, if you are using the Norton Editor and a cross assembler for the 8096, you could make a batch file called CHANGE3.BAT

```
C> COPY CON CHANGE3.BAT
NE MODULE3.ASM
X8096 MODULE3 -ep
ERASE  TEST.BIN
LINK -c MODULE1 MODULE2 MODULE3 -o TEST.BIN -X
UL96  2080 3FFF BINLOAD TEST.BIN   STARTUP
^Z
```

After you've made this batch file, you could start the process of altering MODULE3 by typing in the command:

```
C> CHANGE3
```

The batch file first puts you into your editor, so you can alter the source file. Once you are done editing and exit from the editor, the batch file continues. You will not have to touch the keyboard again until after you get the **STARTUP** trace from the UniLab. The batch file assembles your code, erases the old version of the program, links a new version, then calls the UniLab software, loads the program into emulation memory, and starts it executing.

Meanwhile, you can do something else, and thus avoid the tedium of the mechanical steps.

Complex command tails and macros

The command tail can include as much as you want, as long as it all fits on one line. For example, you could not only load a binary file into emulation memory, you could also load in the symbol file, disassemble the program starting from address 00, and then start the program running:

```
     C> ULZ80 0 7FF BINLOAD DEMO.BIN  SYMFILE DEMO.SYM  0 DN
STARTUP
```

You would probably want to put this command tail into a batch file if you were going to use it more than once. If you make a typing mistake while entering the command tail, you might not discover it until it is too late, for example when the UniLab software tries to find a file called DEMO.BUN.

You could write a UniLab macro that does all the above and more:

```
               : LOADUP
                  0 7FF BINLOAD DEMO.BIN
                  SYMFILE DEMO.SYM
                  0 DN
                  ONLY NOT 200 TO 2A3 ADR  AFTER  FE DATA  1200
                   ADR S ;
```

and use that on the DOS command line:

```
               C> ULZ80  LOADUP
```

By using a macro, you would be able toget around the 80 character limitation on the length of your command tail. Be sure to use the UniLab command **SAVE-SYS** after you create a macro that you want to save. For more information on macros consult the UniLab Reference Manual-- see Appendix F and the entry for : (the colon) in the Command Reference Chapter.

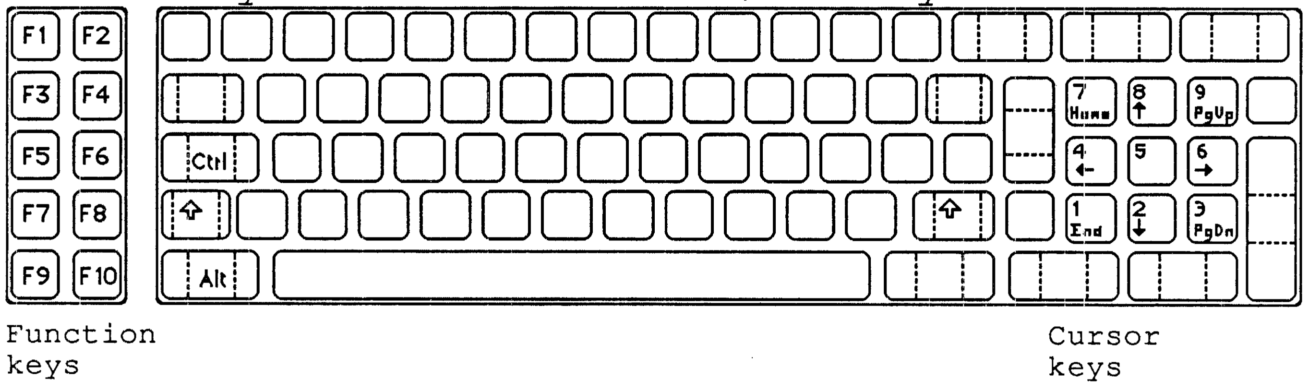
4. Special Features

This section tells you how to use the function keys and the cursor keys. These keys give you access to special features of the UniLab.

The section starts out with a brief overview of the function and cursor keys. This is followed by several pages of diagrams that summarize the features assigned to the cursor and function keys. Last comes the bulk of this section, a tutorial on how to use special keys with each of several features:

trace display,
screen history,
split screen,
command line editor, and
textfiles.

Key locations on PC/AT keyboard



Overview

Function keys

Use the function keys to call up features with a single key stroke. For example, you use **F2** (function key two) to split the screen, **F8** to enter the mode panel. The function keys are reassigned while you are in menu mode.

With the Shift, Control and Alter keys

Three keys can be used to modify the function keys. This provides a total of 40 "soft-keys." Each function key can be assigned four different commands, which you get access to by pressing the function key alone, or while holding down:

the Shift key	(labeled with a hollow arrow),
the Control key, or	(labeled with Ctrl),
the Alter key	(labeled with Alt).

The control key is also used to modify some cursor keys.

How to assign function keys

As you can see in the chart of the function keys, many have been left unassigned. You can assign these surplus keys to any command you choose (as long as that command does not require parameters).

Four commands assign a command to a function key:

```
<key number> FKEY              <command>
<key number> ALT-FKEY          <command>
<key number> SHIFT-FKEY        <command>
<key number> CTRL-FKEY <command>
```

For example, to assign the command **SAMP** to **ALT-F5**, you would type in

5 ALT-FKEY SAMP

You can assign to a function key a macro that you have defined. See Appendix F to learn how to define a macro.

Cursor keys

The cursor keys are used for movement:
"up" into screen history,
"down" through the trace buffer,
"back" to a previous command,
from one window to another windows,
within the mode panels,
back and forth through a textfile, and
down through on-line glossary entries.

The following pages will show you how to use the cursor keys Make certain that NUM LOCK (number lock) is not on when you want to use the "cursor movement" meaning of the key.

How to assign cursor keys

There are no specific keys or commands to change the way the cursor keys work-- their "meaning" changes dynamically when you invoke different features, as indicated in the diagrams on the following pages

The **PgDn** key, for example, will usually present another screenful of the trace display. After the **WORDS** command it will show you the next screenful of the list of commands.

The cursor keys also have special uses in the setting of the window size, in the use of the pop-up panels, and in the screen oriented memory command, **MODIFY.** When in these modes, no other keys will be effective, and a special prompt will inform you that the key settings have changed.

Usual settings

Though the cursor keys are reassigned as you invoke different features, they are usually assigned to three types of functions:

view the trace buffer,
review screen history, and
use the command line editor.

These default settings are summarized on the next page.

View the trace buffer with three cursor keys:

Home
Down Arrow
PgDn.

Review screen history with two cursor keys:

Up Arrow
PgUp.

And use the command line editor, to recall previous commands which you can then edit and reissue, with two regular cursor keys and four "controlled" ones:

Left Arrow
Right Arrow
Ctrl-Left Arrow
Ctrl-Right Arrow
Ctrl-PgUp
Ctrl-PgDn

Special key diagrams

The following pages use diagrams to summarize the use of the function and cursor keys in the UniLab software.

These diagrams are followed by a demonstration of the special features.

Function key assignments

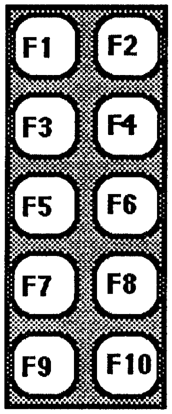
HELP with general instructions for using glossary. Also Function Key assignments.

Next Step - Execute next instruction. Will not follow jumps or branches.

Restore window split to Default sizes.

TSTAT - Display current trigger spec.

STARTUP - Issue reset pulse to target and trace first cycles of target operation.



SPLIT mode - Enter/Exit split screen mode.

NMI - Issue pulse on NMI- line to target, to gain DEBUG control or to single step through code.

MODE - Bring up pop-up mode panels for changing display or system modes.

MENU - Enter/Exit menu mode.

Function Key assignments when no other key held down

Function Key assignments when Ctrl key held down

Help for using on-line displays	F1	F2	Help for using windows
Help for Debuggers	F3	F4	Help for simple analyzer triggers
Help for Emulation memory functions	F5	F6	More help for analyzer triggers
Help for loading/ saving programs	F7	F8	Help for mode panel switches
Help for displaying/ altering memory	F9	F10	Help for trace display

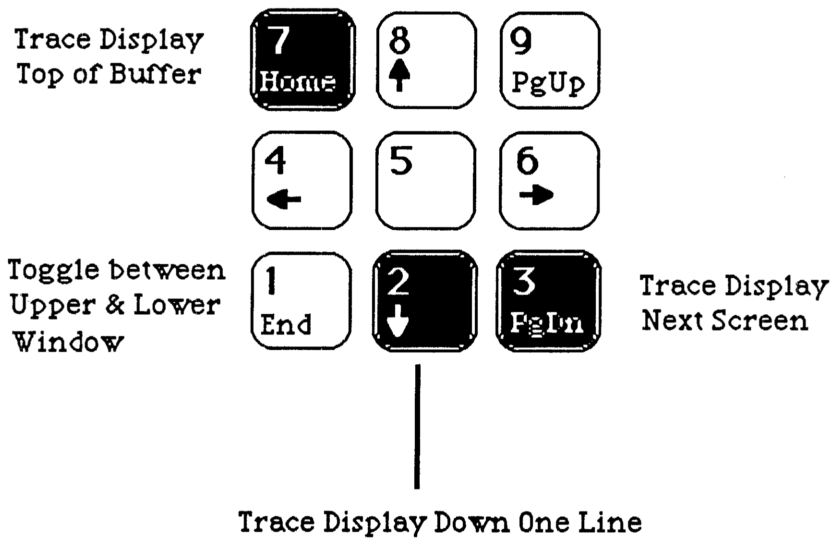
Function Key assignments when ⇧ key held down

List Function Key assignments for Shift	F1	F2	MEMO - Bring up system editor for use as custom memo pad
----	F3	F4	ASC - Show ASCII characters and hexadecimal code
----	F5	F6	----
----	F7	F8	WSIZE - Set new window split size
RES- - Pulls RES- output line low, and holds it low	F9	F10	----

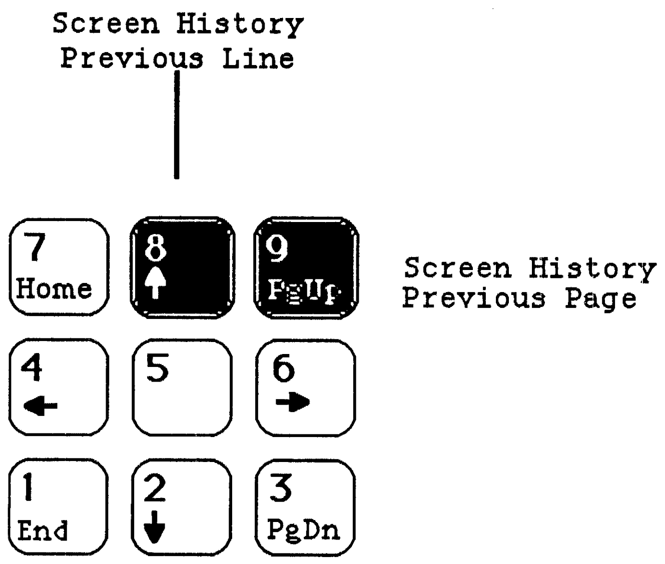
Function Key assignments when Alt key held down

List Function Key assignments for Alt	F1	F2	----
----	F3	F4	----
----	F5	F6	----
----	F7	F8	----
SSAVE - Save the screen image as a text file	F9	F10	Call up the Program Performance Analyzer Menu

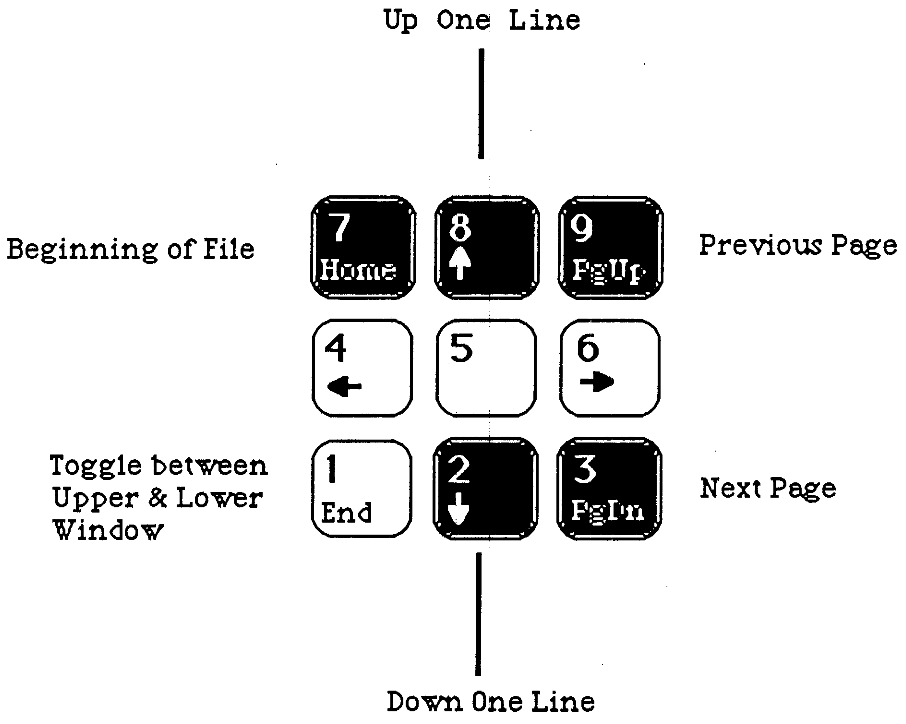
Cursor Keys and the Trace Buffer Display


Trace Display
Top of Buffer
7 Home
8 ↑
9 PgUp
4 ←
5
6 →
Toggle between Upper & Lower Window
1 End
2 ↓
3 PgDn
Trace Display Next Screen
Trace Display Down One Line

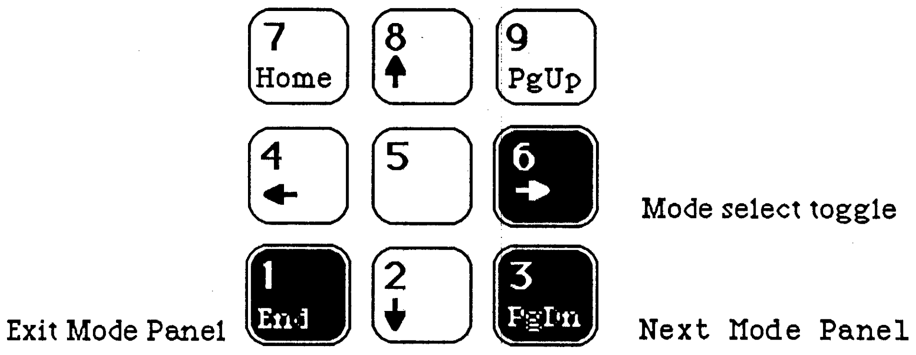
Cursor Keys and the Screen History



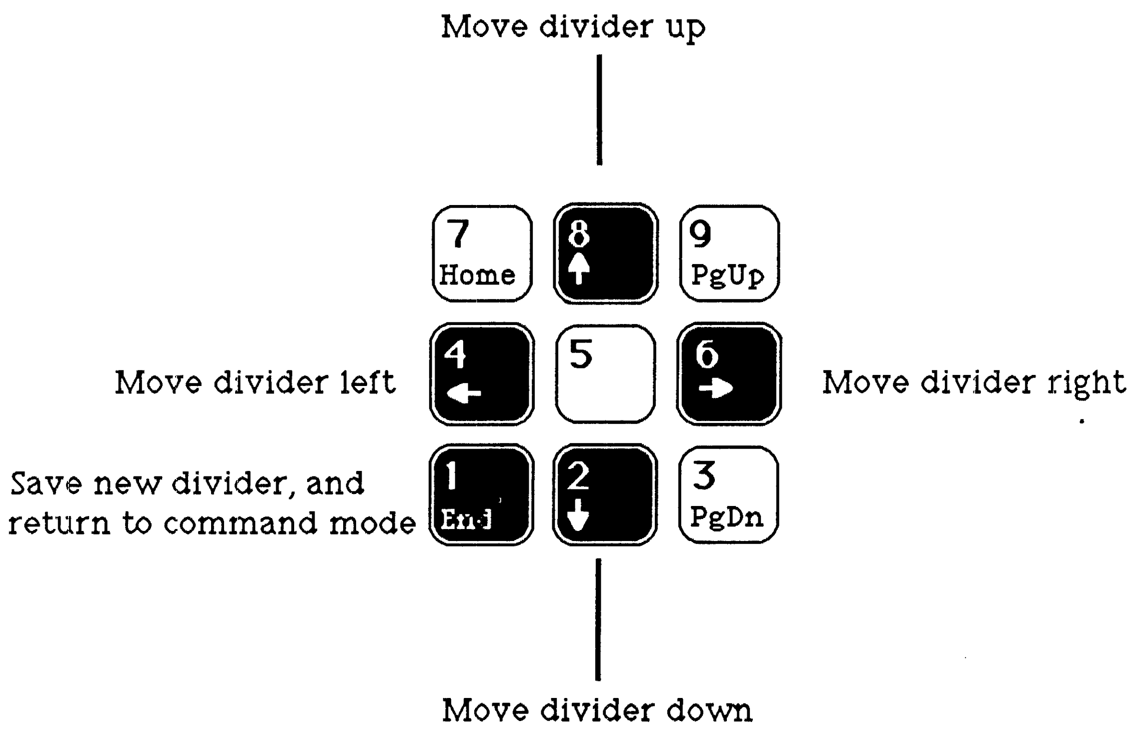
Cursor Keys and Textfiles



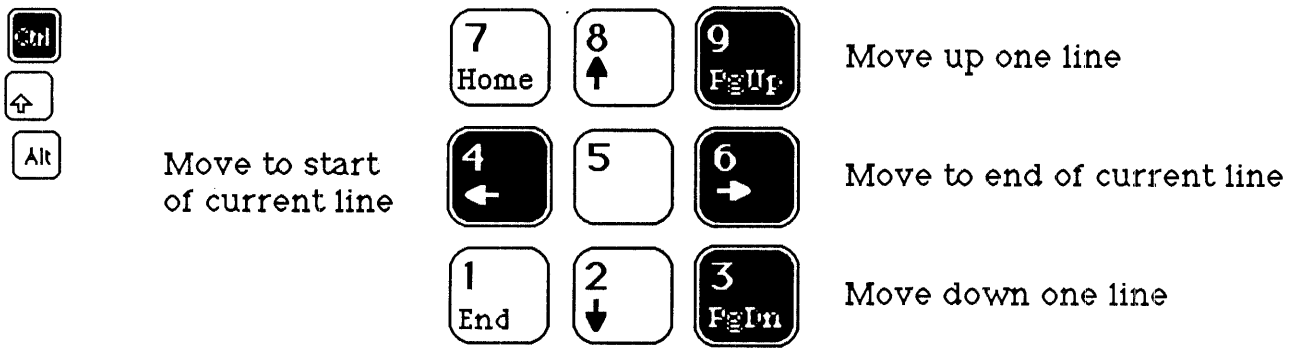
Cursor Keys and the mode panel
(enter the mode panel with **F8**)



Cursor Keys and split screen setup
(enter the setup screen with **Shift-F8**)



Cursor Keys and the command line editor
Press the cursor keys while holding down **Ctrl** key.

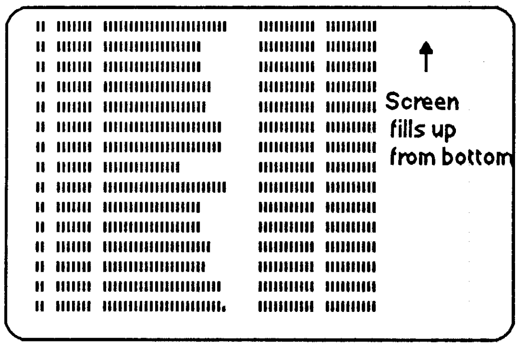


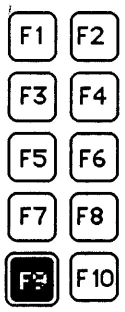
The use of Special Keys

The rest of this chapter shows examples of the use of the special keys. Each subsection shows how to use the cursor keys and the function keys to get the most out of your UniLab. Screen mockups are used to show the result of each action, starting with how to move through the trace display.

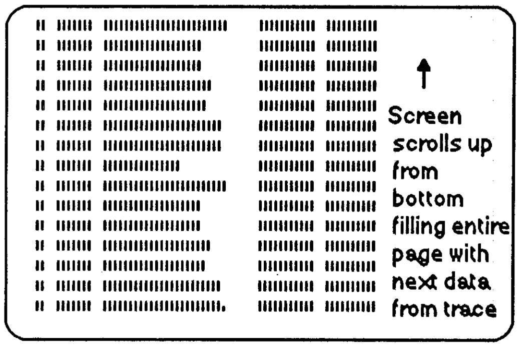
Trace display

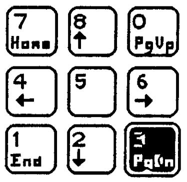
In command mode, first generate a trace, either of your own program, or of the sample target program. To get a trace of the first 170 cycles of the sample program, type **LTARG** then press **F9** (**STARTUP**).





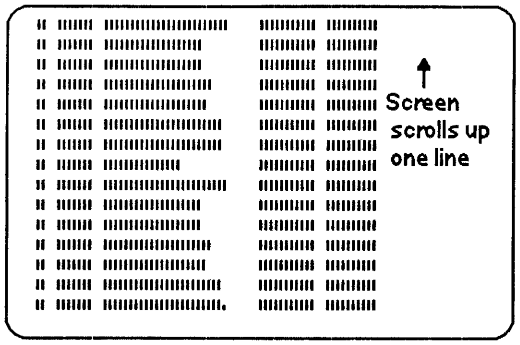
Once you have a display of the first part of the trace, press the **PgDn** cursor key to display the next screen full of data:

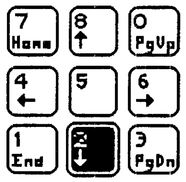




You could press **PgDn** again to see the next portion of the trace buffer. Do not press **PgDn** more than a couple of times for now-- you would eventually come to the end of the trace.

You use **PgDn** to get a whole new screen of information. If you want to see just one or two more lines, press the **Down Arrow** cursor key once:

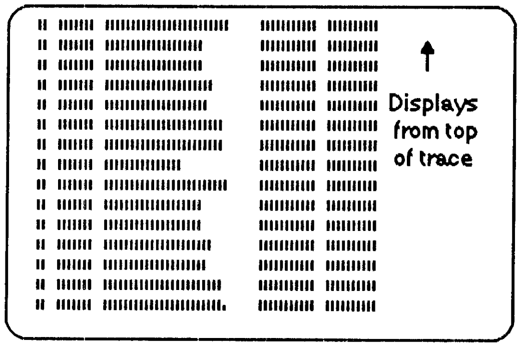


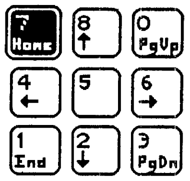


One additional line is shown, and everything else is scrolled up by one line. You can press the **Down Arrow** four or five times to see a 4 or 5 more lines.

You cannot move backward through the trace, but you can see the trace again from the top, and you can recall previously displayed lines from the history buffer. The discussion of the screen history appears in the next two pages.

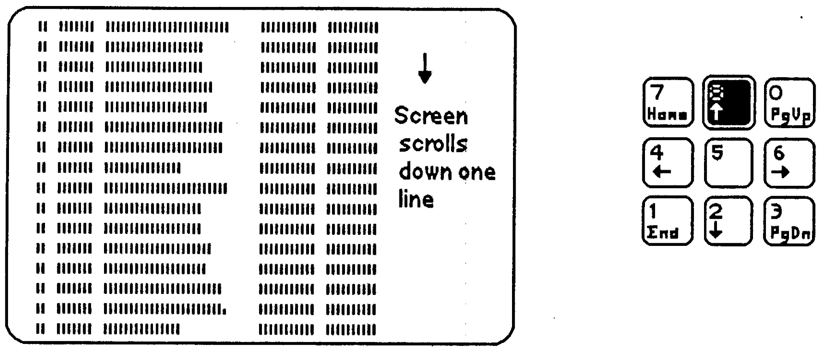
Press the **Home** key to see the trace again from the top:





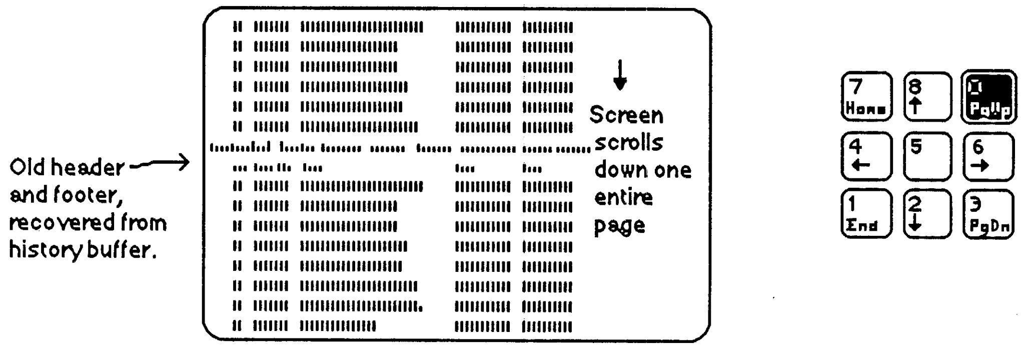
Screen history

The **Up Arrow** is not the opposite of the **Down Arrow**. Instead, it recalls lines from the line history. Try it once:



Though it might seem as if you are looking back through the trace, you actually are looking back through the screen history--the record of everything that has appeared on your screen.

Now do a **PgUp** to see the previous screenful of the history:


Old header and footer, recovered from history buffer.
Screen scrolls down one entire page
7 Home
8
9 PgUp
4
5
6
1 End
2
3 PgDn

You will notice that the screen display backed up a whole page, but the display looks a little funny. Since **PgUp** recovers all the lines that have gone by, you will get back the trace header and footer, but they will end up in the middle of the page.

When you scroll back, you look at a history of the screen. Almost everything that is scrolled off the top of the screen is recorded by the UniLab program. The exceptions are explained below. Every line that scrolls off the top of the screen is recorded and can be "played back" by using the **Up Arrow** and **PgUp** cursor keys.

You can retrieve information that has disappeared off the top of the screen. Any trigger spec, command, or display that scrolled off the top of the screen can be seen again by scrolling the screen down.

History exception: the split screen

When you split the screen, as discussed in the next subsection, only the information in the lower window is recorded in the history buffer. Lines scrolled off the top of the upper window are not recorded.

You can retrieve the line history into either window.

Summary of history and trace display cursor keys

Go forward with the **Down Arrow** or the **PgDn** cursor keys to see new analyzed data from the trace buffer.

Go backward with the **Up Arrow** or the **PgUp** cursor keys to see the history of the current session with the UniLab.

Increase the size of history buffer

You can have a maximum of 60K devoted to the history buffer. Use the command **?FREE** to get a report of the current memory allocation in the host, including how much RAM is available.

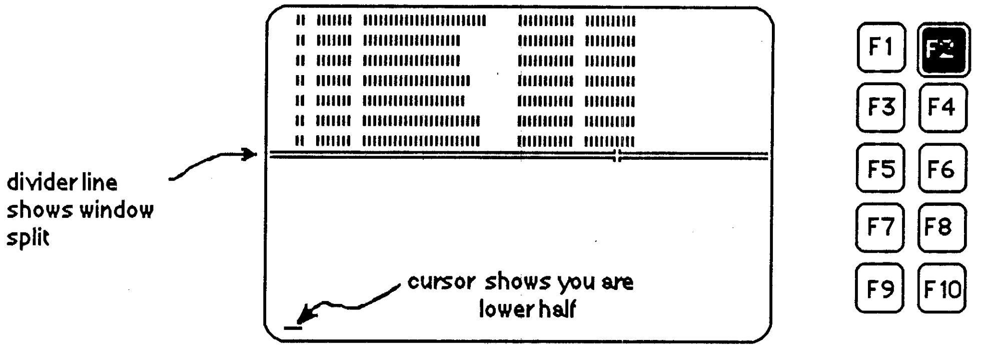
```
?FREE

20K allocated to history
40K allocated to symbol table

220K of host RAM free
```

Increase the amount of RAM devoted to the history buffer with the command <# Kbytes> **=HISTORY.** This command changes the amount of memory that will be allocated when the software is called from DOS. The new allocation will not take effect until you **SAVE-SYS,** exit the Unilab program and re-enter the newly saved software.

Windows

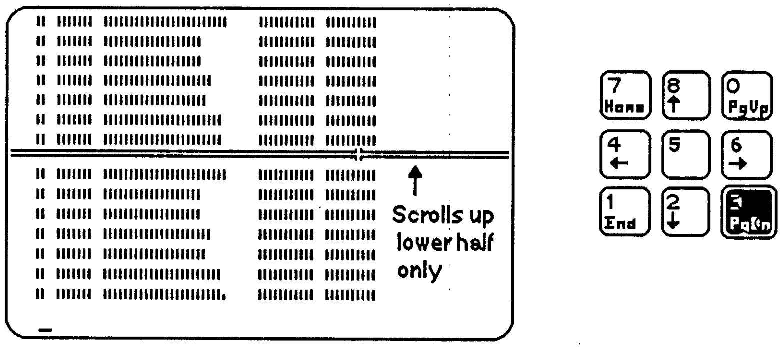
The UniLab software allows you to display information in either one of two windows, and provides two special disassembly windows as well.

Press **F2** to split the screen:


divider line shows window split
cursor shows you are lower half
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10

You will see a display like the one above. The lower portion of the screen has been wiped clean, and a horizontal dividing line has appeared in the middle of the screen.

The flashing cursor is now on the bottom line of the screen. Press the **PgDn** key:

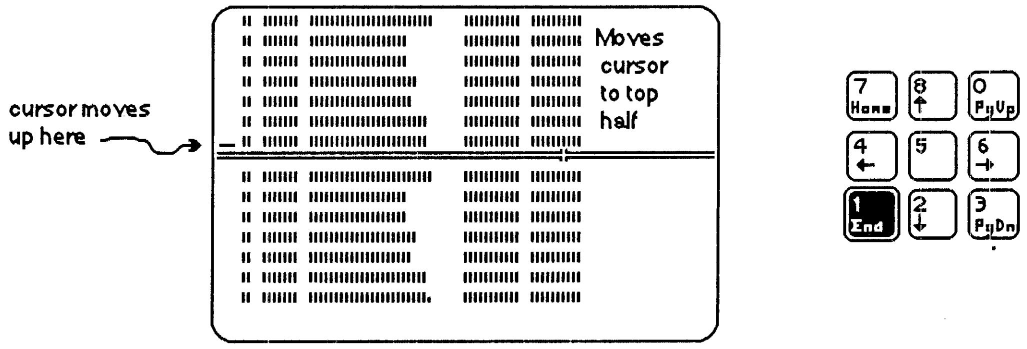


As you would expect, the trace listing scrolls up the screen, but it stops when it fills only the lower half of the screen. Press **PgDn** again, and notice that the trace listing scrolls by in the lower window, while the upper window remains the same.

Move from window to window

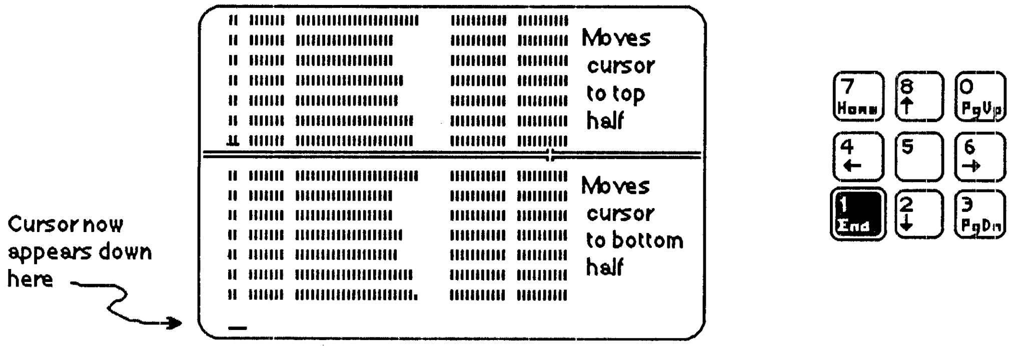
When yo u pressed **F2** that first time, you split the screen. The display is now set up for two viewing windows. All of your commands and their actions are displayed only in the lower window.

You can now use the **End** key to move from one window to another. Press **End** once and notice that the cursor goes into the upper window:



You can perform any UniLab command in the upper window, and the lower window display will stay the same. You will find this ver y useful for comparing two trace displays, or keeping part of a trace on the screen while you get a breakpoint display.

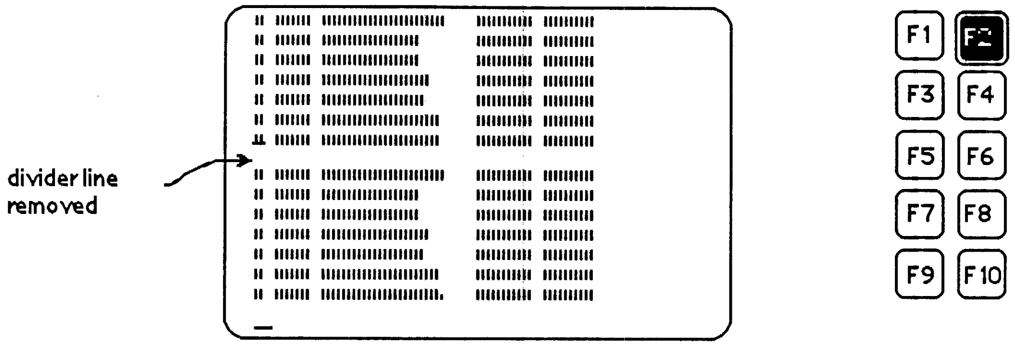
You can also be able to examine textfiles, as will be discussed in a few pages. Right now, press **End** again to move back into the lower window:



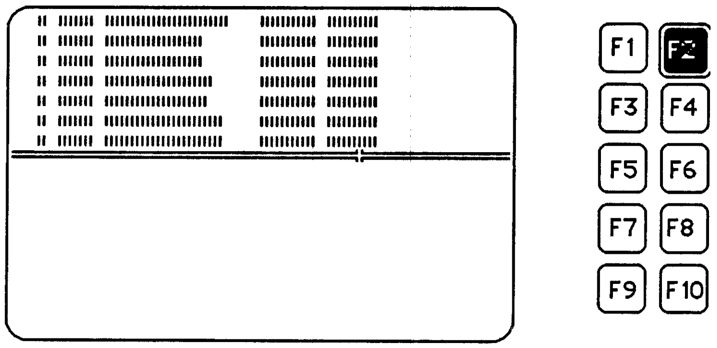
The **End** key moves you back and forth between the two windows. You can use it any time after you have split the screen.

Toggle split screen off and on

To get out of the split screen, press **F2** again (the same key that split the screen in the first place):



The dividing line disappears. If you scroll now, the entire screen will be used for the display. Press **F2** to split the screen again:

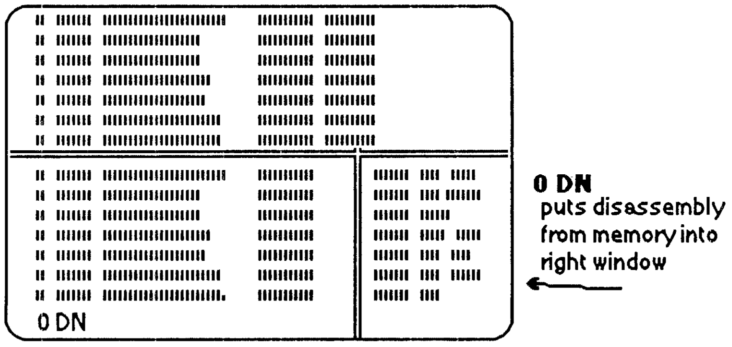


Show disassembly from memory in dedicated window

The right hand section of the screen serves as a dedicated window, which shows only disassembly from memory.

First, with a split screen, press **F9** again so that you will have a trace that starts from the reset address.

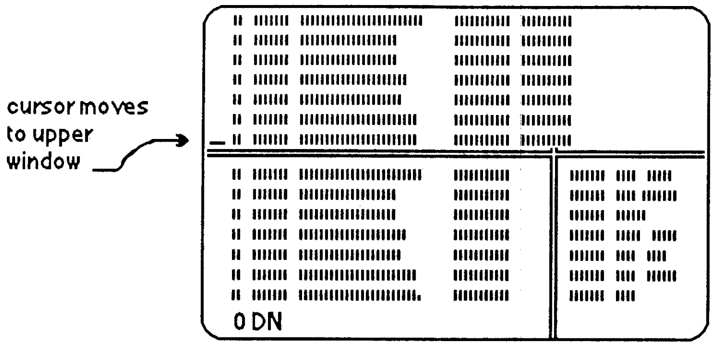
Type the command **0 DN**. Remember to press the carriage return (enter) key when you have finished typing the command. You will see a display like below:

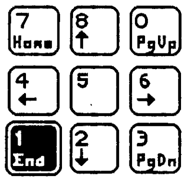


DN is a special word that disassembles from memory and displays in a dedicated window on the right-hand side of the screen. **DN** disassembles enough lines to fill up the right hand side of the current window. Compare this with **DM**, which needs two arguments-- the address to start disassembly from, and the number of lines to disassemble-- and displays into the current window.

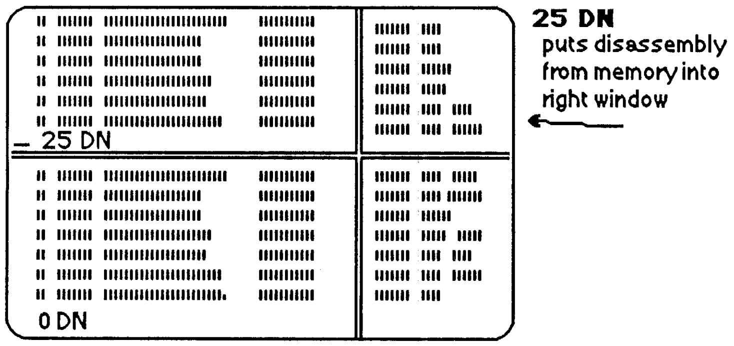
The special disassembly window will stay in place until you press **F2** again.

You can also disassemble into the upper right hand window. First press the End key to move the cursor up there:





Now use the **DN** command again. This time it will display in the upper righthand window:



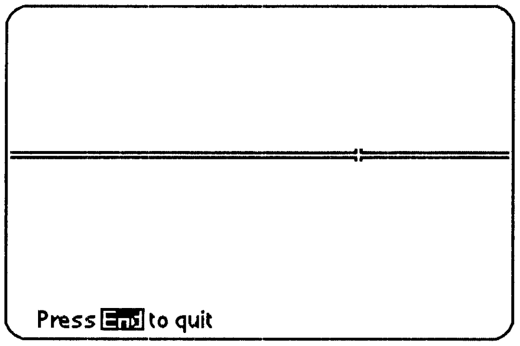
You can also use **DN** in the full screen.

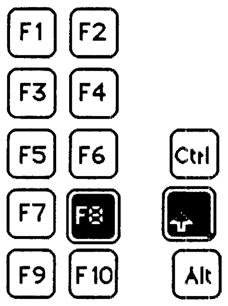
Change window size

You can change the default window size by moving the split line around with the cursor keys after you press **SHIFT-F8** (tap function key eight while holding down the Shift key).

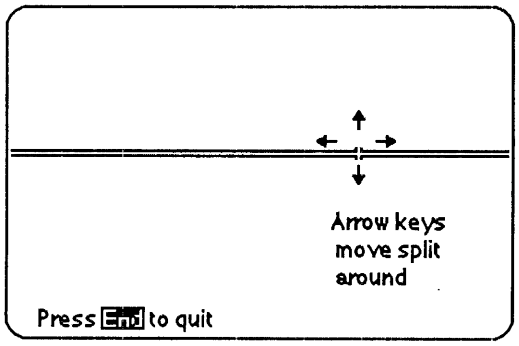
For example, if you would rather have a larger lower window, you can move the split line up.

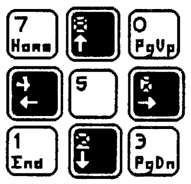
When you press **SHIFT-F8** you will see this display:





Now use the cursor arrow keys to move the split up or down, and right or left. Move the split up to make the top half a little smaller than it was.



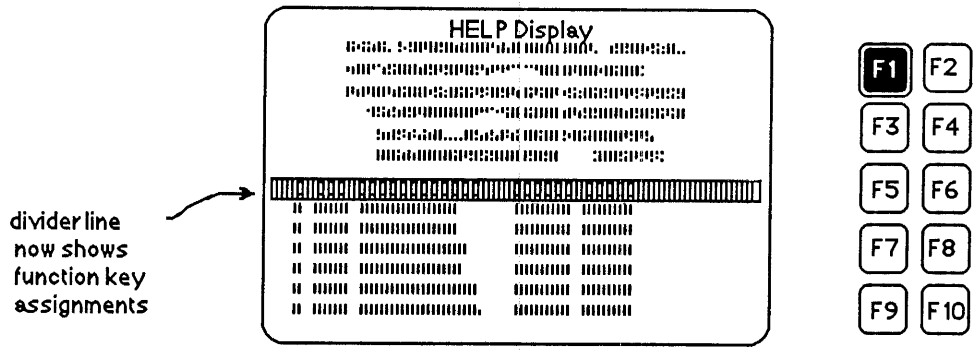


Press the **End** key when you have the split where you want it. No other keys will have any effect while you are setting the window partition.

Now you can display the trace in the new window sizes.

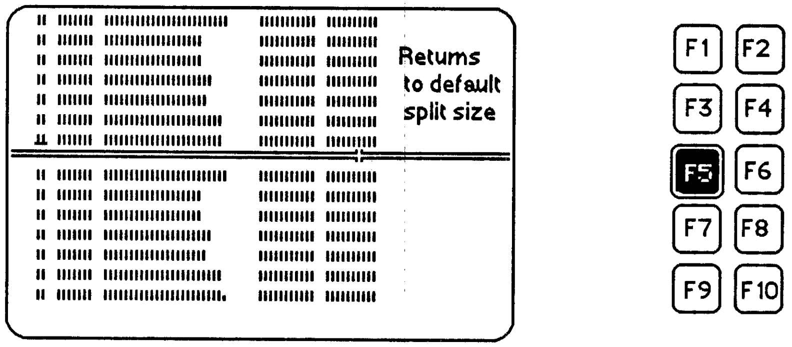
Split screens and help displays

When you have a the split screen, the HELP screen displays will automatically change the location of the split, so that the text of the help display fills the upper window. Press function key **F1** to see this:



Now press the **End** key again to move the cursor to the top window, and press the Home key. The text of the help screen will be overwritten by the trace, but you've got a different split from the one you set not long ago.

If you want to return to the split that you set before, press **F5** and the window split will return to that setting, the new default:



The window size you set with **Shift-F8** will be saved if you use **SAVE-SYS** to save the current configuration of the system to disk. Of course, you can use use **Shift-F8** at any time to change the split size again.

Command line editor

Overview

This special feature of the UniLab provides the ability to edit and repeat any command on the screen. It works both on commands that have just been typed and on commands that are retrieved from the screen history after scrolling off the top of the screen.

Move the cursor back up to previous commands with **CTRL-PgUp**. If you go too far press **CTRL-PgDn** to move down a line.

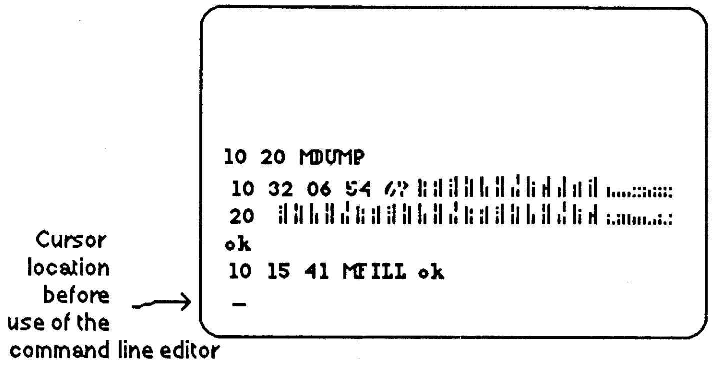
Once you are on the command you wish to edit, use the arrow keys to move around on the line.

The command line editor is always in "insert" mode. Any character you type will be inserted at the current cursor location. Use the backspace key or the **Del** key to delete a character.

Ctrl-Left Arrow moves you to the start of the line, while **Ctrl-Right Arrow** moves you to the end of the line.

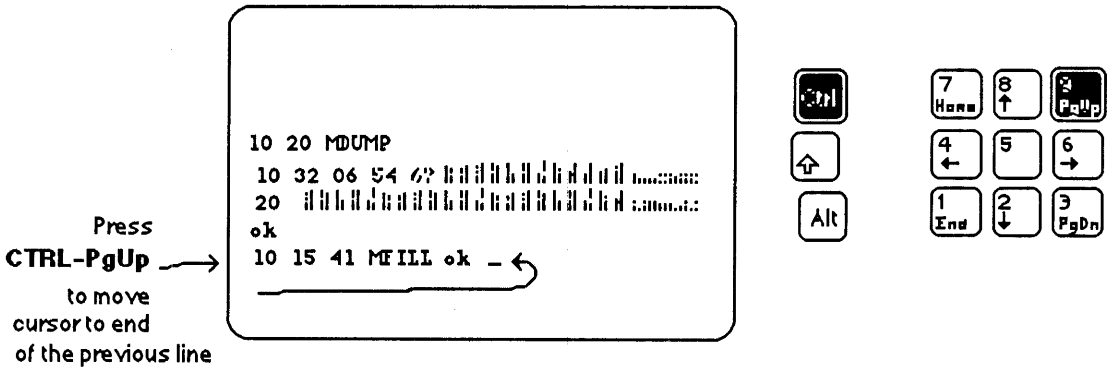
When you press the carriage return key, the UniLab will treat the line, <u>up to the current cursor position</u>, as a newly entered command.

Reissue a command without editing



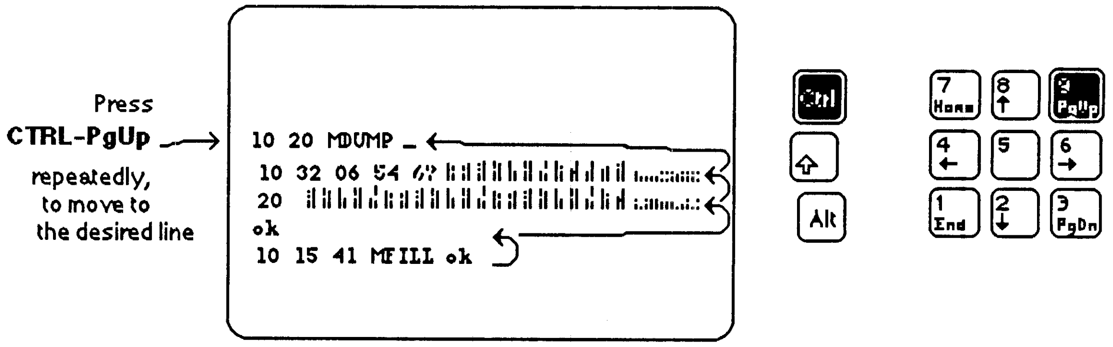
```
10 20 MDUMP
10 32 06 54 67
20
ok
10 15 41 MFILL ok
_
```

In this example, suppose you have dumped 20 bytes of memory, and then filled part of that memory area with hexadecimal'41 (ASCII for A). Now, to confirm that the correct data is in memory, you want to reissue the dump command with the command line editor.



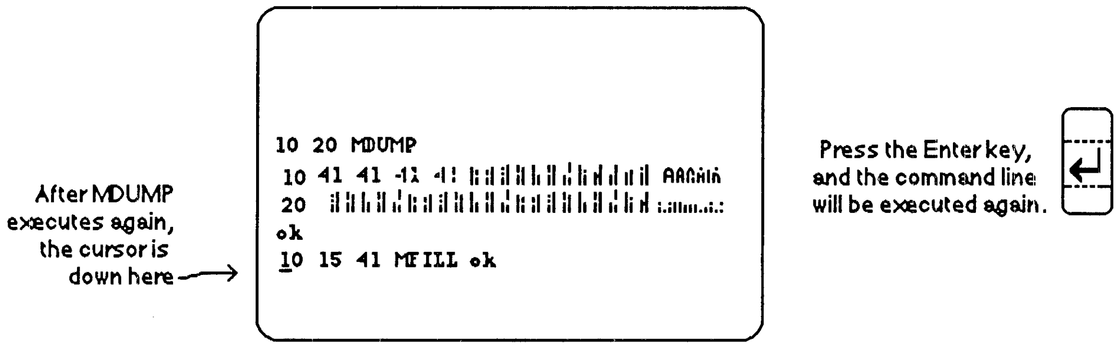
```
10 20 MDUMP
10 32 06 54 67
20
ok
10 15 41 MFILL ok _
```

The **Ctrl-PgUp** key combination moves the cursor up to the previous line. To move up a number of lines, press this key combination repeatedly, or hold both keys down for a brief period. If you move the cursor up too far, use **Ctrl-PgDn** to move back down.



```
10 20 MDUMP _
10 32 06 54 67
20
ok
10 15 41 MFILL ok
```

Once the cursor is on the correct line, press the Enter key, and the dump command will be executed again. The memory display generated by dump writes into the same place on the screen, but with the updated values:



```
10 20 MDUMP
 10 41 41 [illegible] AAAAA
 20 [illegible]
ok
10 15 41 MFILL ok
```

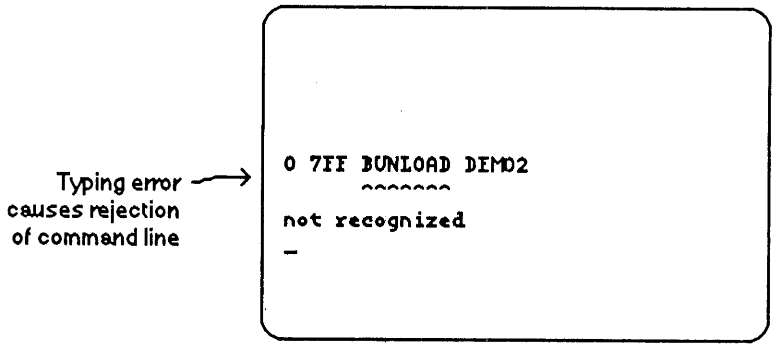
After MDUMP executes again, the cursor is down here
Press the Enter key, and the command line will be executed again.

The cursor ends up at the same place it was before the user typed the memory fill command. And because you might want to edit and reissue that line as well, it is not wiped clean. To use that line, press **Ctrl-PgUp** followed by **Ctrl-PgDn**.

Edit and then reissue a command line

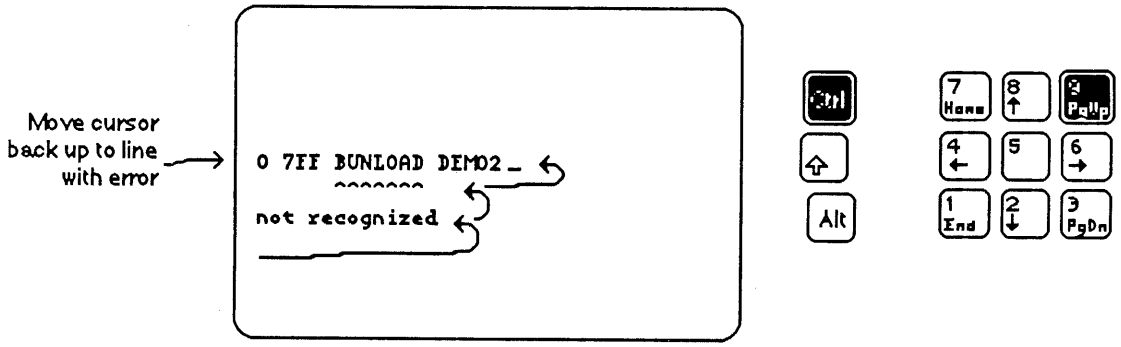
You can alter a command before re-using it. This is especially useful when you make some minor typo while writing a long command string, or when you wish to change a trigger spec slightly.

In this example, you have typed "**BUNLOAD**" by mistake, and now want to move back up to that incorrect command line, delete the **U** and type an **I** in its place.



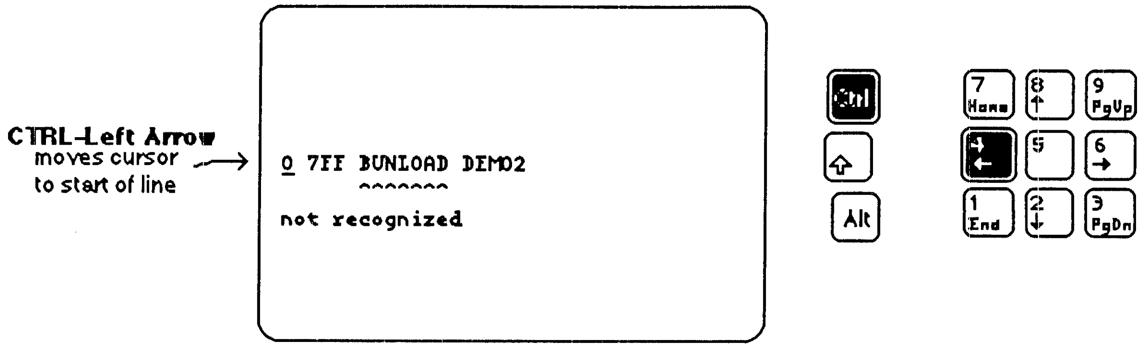
```
0 7II BUNLOAD DEMO2
      ^^^^^^^
not recognized
_
```

First you must move the cursor up to the mis-typed command line.



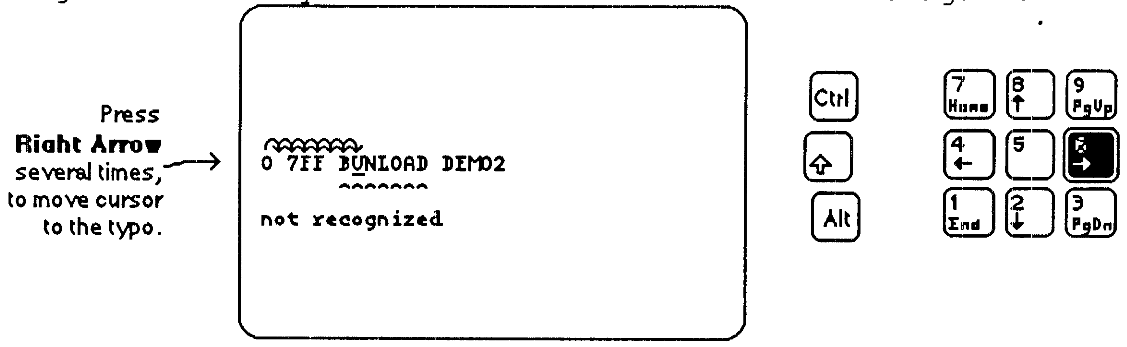
```
0 7II BUNLOAD DEMO2_
      ^^^^^^^
not recognized
```

Then, **Ctrl-Left Arrow** moves the cursor to the start of the command line:



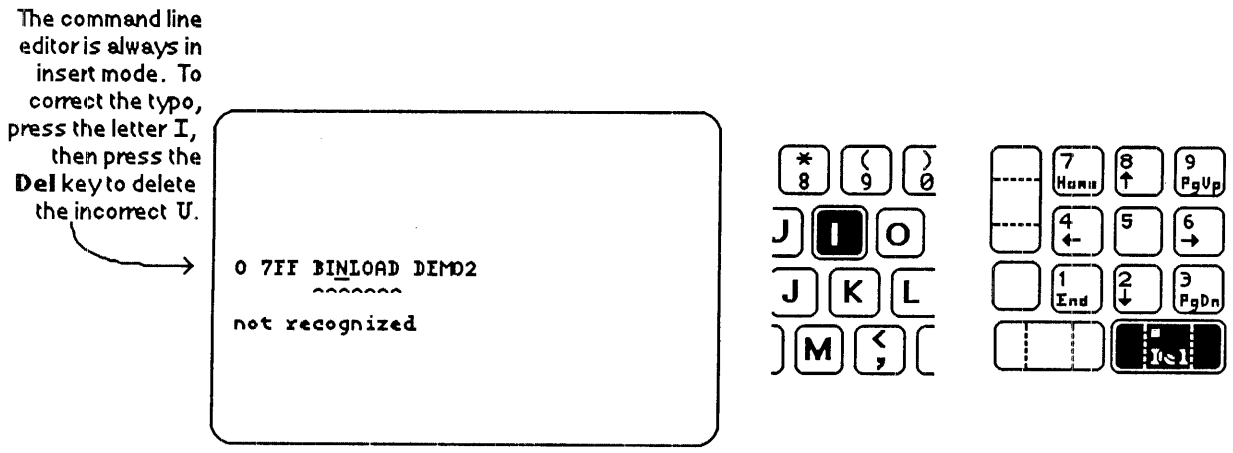
```
0 7II BUNLOAD DEMO2
      ^^^^^^^
not recognized
```

Repeated use of the **Right Arrow** key moves the cursor over to the **U** in **BUNLOAD**. If you press **Ctrl-Right Arrow**, the cursor will go all the way to the end of the command line again.



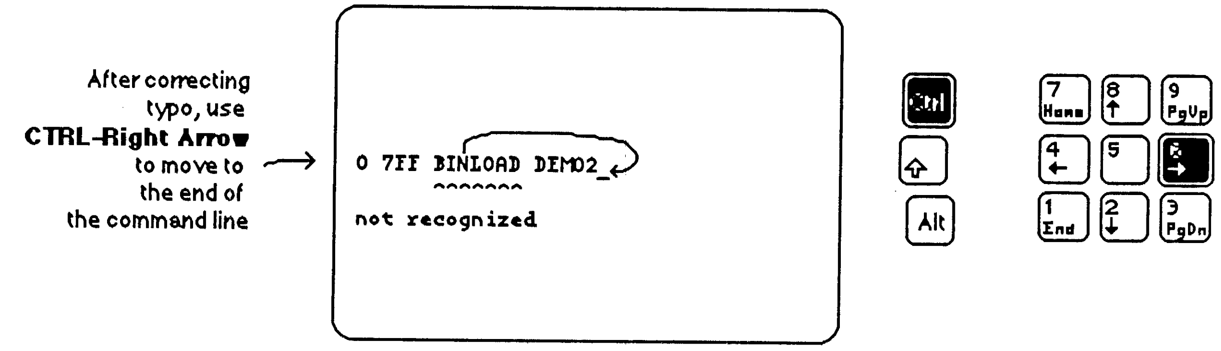
```
0 7II BUNLOAD DEMO2
      ^^^^^^^
not recognized
```

Now you can type an **I** to replace the **U**. Since the command line editor is always in "insert" mode, never in "overwrite," you need to use the **Del** key to delete the incorrect character. **Del** deletes the character that the cursor is on. The backspace key erases the character to the left of the cursor.



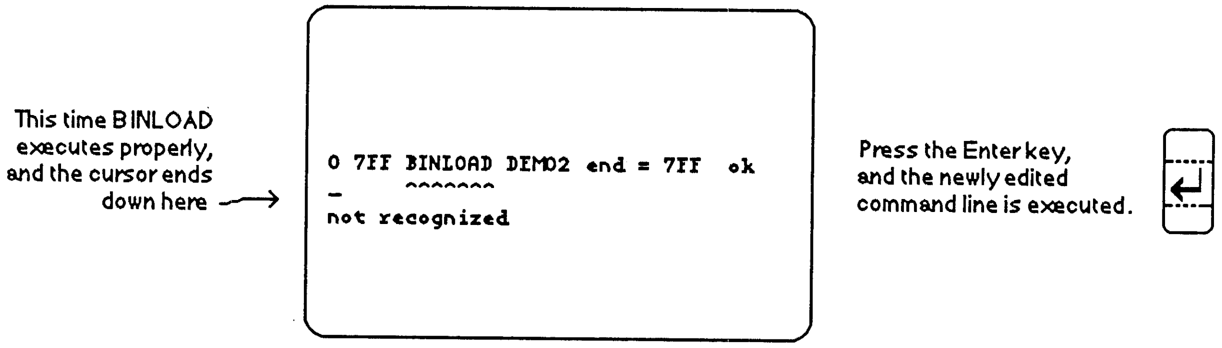
```
0 7II BINLOAD DEMO2
      ^^^^^^^
not recognized
```

Now, after the mis-typing has been corrected, use **Ctrl-Right Arrow** to move the cursor to the end of the command line. When you press the **Enter** key, the command line will only be interpreted up to the current location of the cursor. So, you move the cursor to the end of the line first:


After correcting typo, use
CTRL-Right Arrow to move to the end of the command line
Ctrl
Alt
7 Home
8
9 PgUp
4
5
6
1 End
2
3 PgDn

```
0 7FF BINLOAD DEMO2_
      ^^^^^^^
not recognized
```

Press the **Enter** key, and the newly edited command will be interpreted. This time it works, as indicated by the "end= 7FF" message. The "not recognized" message and the line of carats (^) are left over from the first time the command was interpreted.


This time BINLOAD executes properly, and the cursor ends down here
Press the Enter key, and the newly edited command line is executed.

```
0 7FF BINLOAD DEMO2 end = 7FF  ok
_      ^^^^^^^
not recognized
```

Text files

Often, while working with the UniLab, you will need to view a text file. Just being able to look at the source code for your application is much easier than having to go dig though a printout (assuming you had the foresight to make one).

The UniLab **TEXTFILE** command provides a convenient way to view any text file, including source code files. The text always appears in the top window.

Open a textfile by typing:

TEXTFILE <name>

After a few seconds (or more if it's a large file), you'll see the first few lines of the text file in the top window.

The <name> is the full DOS pathname of your file. For example, if you have a textfile called MYFILE and it is on the diskette in drive B, then specify B:MYFILE just like you would in DOS.

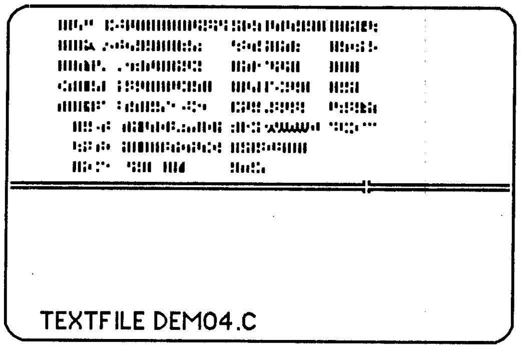
The file must be a DOS textfile. Most text editors have the capability of saving files as text only, or as "non-document" files. The textfile command will not accept a file that has embedded formatting commands. If you try to use TEXTFILE to open a file that is not text, you will get the message "Not a DOS TextFile."

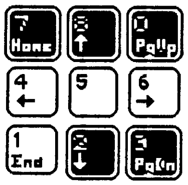
Cursor key assignments when viewing a text file

PgDn	shows the next screenful of the textfile
Down Arrow	shows the next line
PgUp	shows the previous screenful
Up Arrow	shows the previous line
Home	shows the file from the top

Unlike when you view the trace buffer, reverse scrolling is not showing you a line history, but rather is moving you around in the image of your text file. (Remember that lines in the top window are not recorded in screen history.)

Try moving around in this image of your text file. Remember that you cannot change your file with this command:





These cursor keys allow you to move through a text file

Chapter Four: Program Performance Analyzer

Contents

Introduction

This chapter describes the optional Program Performance Analyzer (PPA). The information here only applies if you have purchased that option.

This chapter includes a guide to analysis, together with typical examples and a PPA specific troubleshooting guide.

Related chapters

Before using this chapter, you will want to have the UniLab installed according to the directions in Chapter One.

The current chapter covers only the PPA. Chapter Two, the **Guided Demo,** is a brief introduction to other features of the UniLab. Chapter Three, **Operation,** is a guide to the UniLab features.

1. Overview of the Program Performance Analyzer (PPA)

1.1 Basics

What the PPA does

The Program Performance Analyzer (PPA) gathers data about your software and displays that data as both a numeric table and a bar graph.

Confidence in your data

The data reflects the true performance of your software, because the PPA gathers performance data while your microprocessor board executes at full speed.

No separate installation required

The PPA is already part of your UniLab system. The installation procedure for your UniLab is described in Chapter One.

Menu or command interface

You gain access to all the PPA features either through the commands or through the PPA menu. See sections 1.4 and 1.5 for details.

Address-domain and Time-domain Analysis

Use the Address-domain Histogram (AHIST) to determine the level of program activity in the entire program, or in any specified sub-section, down to a single byte.

Use the Time-domain Histogram (THIST) to determine the run time of any single range of code. Your results will be accurate to within 20 milliseconds.

Multiple pass Address-domain Analysis

Use the Multiple pass Address-domain Histogram (MHIST) to determine the execution time of any set of code ranges. MHIST gives you more accurate results than AHIST and expresses the results in milliseconds rather than in number of accesses to memory. As with THIST, the times will be accurate to within 20 milliseconds.

See section 1.2, **How to choose the correct mode,** for information on choosing among the three PPA commands.

Symbolic labels

It is easy to specify the code you are interested in. All you do is enter the symbolic names of your procedures and functions. The PPA automatically converts the symbols into addresses. Or, if you want, you can manually enter the addresses.

Save data and picture

Whether you use symbols or numbers, you only have to enter the information once-- the PPA software includes a utility for saving the set-up data, described in section 1.8 of this chapter. You can save the data from a test to show a colleague or to include in a presentation.

To make certain that you don't mix up your files, the PPA provides a full-screen width title-- ample room for including date and program information on the screen.

You can also use the graphs in written reports. The touch of a function key (**ALT-F9**) saves the image of the screen to a text file, which you can then print out or place in a document.

A complete solution

With the PPA you can solve the problems once you find them. The other features of the UniLab software provide you with all the tools you need to trace execution of your code, examine registers and memory-- even alter the code with a line-by-line assembler.

The PPA and your software's profile

The PPA collects data about the behavior of a program as it executes. You then compare the actual behavior of your program with the expected profile or "signature" of the program. You can locate problems in your code by noting the differences from your design expectations. And you can optimize your code by watching its behavior during actual running conditions.

In fact, you can alter the input conditions and watch how your program responds-- or compare two test runs gathered under two different input conditions.

1.2 How to choose the correct mode

You call up the PPA with one of three commands: **AHIST**, **THIST**, and **MHIST**. Or, you can invoke any of these three from the PPA menu.

Both THIST and MHIST have two data collection modes, as described in section 1.3.

Address or Time domain ?

If you are interested in the general behavior of your program, or in the relative resource use of several modules, then you should use one of the two address domain histograms, AHIST or MHIST.

If you want to know the execution time of the main loop or of any <u>single</u> sub-section of the code, then you will want to use THIST, the time domain histogram. THIST is especially useful for looking at a section of code whose execution time <u>varies</u>.

AHIST or MHIST ?

You should use either AHIST or MHIST when you want to collect data on <u>several</u> address ranges of your program. The fundamental difference between the two: for each range you specify, AHIST gives you a count of the number of times memory in the range is accessed, while MHIST tells you the elapsed time between access to the first address and access to the last address in the range.

In addition, AHIST gives you faster but less accurate results. MHIST gives you greater accuracy by collecting data on only one bin at a time. Note that the bin limits you set up are preserved when you move back and forth between AHIST and MHIST.

The two modes have a similar appearance, but act very different. AHIST collects data on all address ranges at once, but goes through a collect-analyze cycle. So it misses any events that occur while it is analyzing.

MHIST cycles through your bins, collecting data on only one bin at a time. It first finds the mean time for each address range, then finds the number of times the first address in each range is executed. Altogether MHIST requires twice as many passes as you have bins. For each "pass" you will want to restart the operation you are interested in, as described in section 5.2.

When MHIST is needed

Use MHIST whenever you need accuracy and AHIST is not able to provide it. AHIST will give you false results whenever there is "shadowing" or "swamping."

Shadowing

Shadowing occurs when AHIST consistently shows the execution of only a small section of code-- and the rest of the code falls into the shadow of this routine.

For example, you have bins set up to watch all your routines, except for the status loop, and your histogram shows activity only in the section of code that occurs immediately after you leave a status loop. While AHIST is busy analyzing that data, the rest of your code executes. When AHIST looks at the bus again, your program is back in the status loop.

When you have this problem, usually one bin will show a <u>low</u> level of activity and the others will show none. However, sometimes several bins will show a little activity.

Swamping

Swamping is a closely related problem: AHIST sees <u>only</u> the execution of the module that occurs most of the time. When the program does execute some other module, AHIST usually is analyzing the trace buffer. When you have this problem, usually one bin will show a <u>high</u> level of activity. While getting swamped AHIST <u>might</u> sporadically catch part ofthe execution of modules other than the dominating one.

When to use MHIST

If the data you capture with AHIST exhibits neither shadowing nor swamping, then you can be fairly confident of your results. If you do see problems, then we recommend that you use MHIST rather than AHIST.

MHIST assumptions

MHIST will work best if your program has certain characteristics:

1) It executes (approximately) the same operation or series of operations during each "pass" of data collection. See below and in section 5.

2) The first address and last address in each routine (as specified on the MHIST screen) is accessed only once each time the routine is called.

Getting valid results with MHIST

There are three ways to make certain that your program will perform the same series of operations during each pass. See section 5.2 for more details.

1) Have reset enabled (**RESET**) so that the program starts again with each pass. You will also need to have any inputs read by your system set up in the same configuration or sequence.

2) Have reset disabled (**RESET'**) and have your program start the operation of interest when the stimulus lines change. You simply put the stimulus cable in the PROM programmer socket, and then connect the lines to the appropriate point on your board. Just before each pass, stimulus outputs S0 through S3 strobe high and then low again, while S4 through S7 strobe low and then high again. For more information on the stimulus outputs, consult section 5.2 of this document and section 8 of Chapter Six.

3) Have reset disabled and manually start the operation of interest just before each pass.

1.3 The three PPA modes

The three PPA commands, **AHIST, MHIST** and **THIST,** give you access to three different ways to examine your program.

AHIST: address-domain histogram

In the address-domain mode, the PPA measures program activity in each user-selected address-range bin.

The column labeled "COUNT" shows, for each address range, the number of times any byte in that range is used. The graph shows what percentage of observed activity takes place in the address range of each bin.

MHIST: multiple-pass address-domain histogram

MHIST allows you to measure the absolute execution time of each address range bin (in milliseconds).

You can display either an address-domain graph of the total execution times, or a chart of average execution time, number of times called and total execution time. You will probably want the chart display when gathering data, since it gives you a clearer idea of how MHIST gathers data.

MHIST: two ways to start

You can start MHIST in one of two modes:

the <u>Manual loop</u> start (**F1**) and
the <u>Timed loop</u> start (**ALT-F1**).

In either mode, you can stop at any time by pressing either the **ESCAPE** key or function key 10 (**F10**).

Manual loop

When you press **F1,** MHIST will determine the average execution time of the address range in the first bin. MHIST will continue this operation until you press any key. It will pause, and wait for you to press another key. Then it will determine the average execution of the second bin.

After all the average execution times have been determined, MHIST will take another set of passes, determining the number of times each bin is called. It will continue to count the number of times the routine is called until you press a key, then it will pause and wait for you to press any key before it moves on to any other. bin.

Timed loop

When you press **ALT-F1** MHIST will prompt you for the length of time (in milliseconds) that you want to gather data on each bin. Then it will go through the same series of operations as the Manual loop, but without any need for you to touch the keyboard.

THIST: time-domain histogram

In the time-domain mode, the PPA measures the time spent executing a section of code.

You specify the section by pressing **F9** and entering the start and stop address.

The column labeled "COUNT" shows the number of times the duration falls into each time bin. The graph displays this information as percentages of the the total number of executions. The screen also displays the <u>mean</u> run-time.

THIST: Two ways to collect data

There are two ways to start the THIST PPA:

the Entry-Exit start and
the Code Range start.

Entry-Exit

When started with **F1,** the PPA records the elapsed time starting when your program accesses the first address in the range and only stopping when your program accesses the last address. Note that this is the same mode used by MHIST when it gathers average execution time data.

Code Range

When started with **ALT-F1,** the PPA records the elapsed time starting when the program fetches an instruction from any address within the range and stopping when the program fetches any instruction from outside the range.

IMPORTANT: The Code Range mode will only work properly if the macro **FETCH** is defined for your processor-specific software package. Check the Glossary section of your Target Application note.

1.4 PPA command Summary

The power of the PPA is available through either the PPA menu, described in section 1.5, or the the PPA commands, described below.

AHIST
Calls the address-domain histogram, described in section 4 of this chapter.

HLOAD <filename>
Loads the histogram data stored in the file, and then calls up the proper histogram. Use this command only on files saved with **HSAVE**.

HSAVE <filename>
Saves the data from the last histogram displayed, after exiting from the histogram.

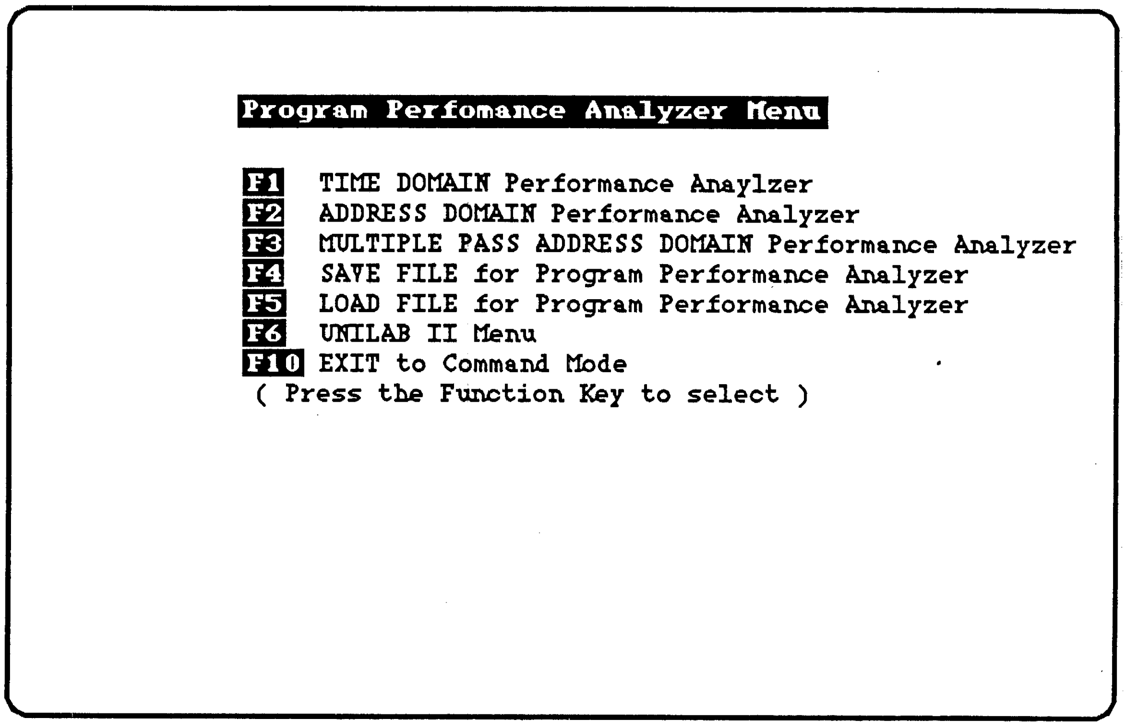
MHIST
Calls the multiple-pass address-domain histogram, described in section 6 of this chapter.

SSAVE
Saves the screen image as a DOS text file. This command is always assigned to function key **ALT-F9**, whether you are in the PPA or not.

THIST
Calls the time-domain histogram, described in section 5 of this chapter.

1.5 The main PPA menu

All the PPA features can also be summoned from the PPA menu. Press the **ALT** key and the **F10** key at the same time:


Program Perfomance Analyzer Menu

F1 TIME DOMAIN Performance Anaylzer
F2 ADDRESS DOMAIN Performance Analyzer
F3 MULTIPLE PASS ADDRESS DOMAIN Performance Analyzer
F4 SAVE FILE for Program Performance Analyzer
F5 LOAD FILE for Program Performance Analyzer
F6 UNILAB II Menu
F10 EXIT to Command Mode
(Press the Function Key to select)

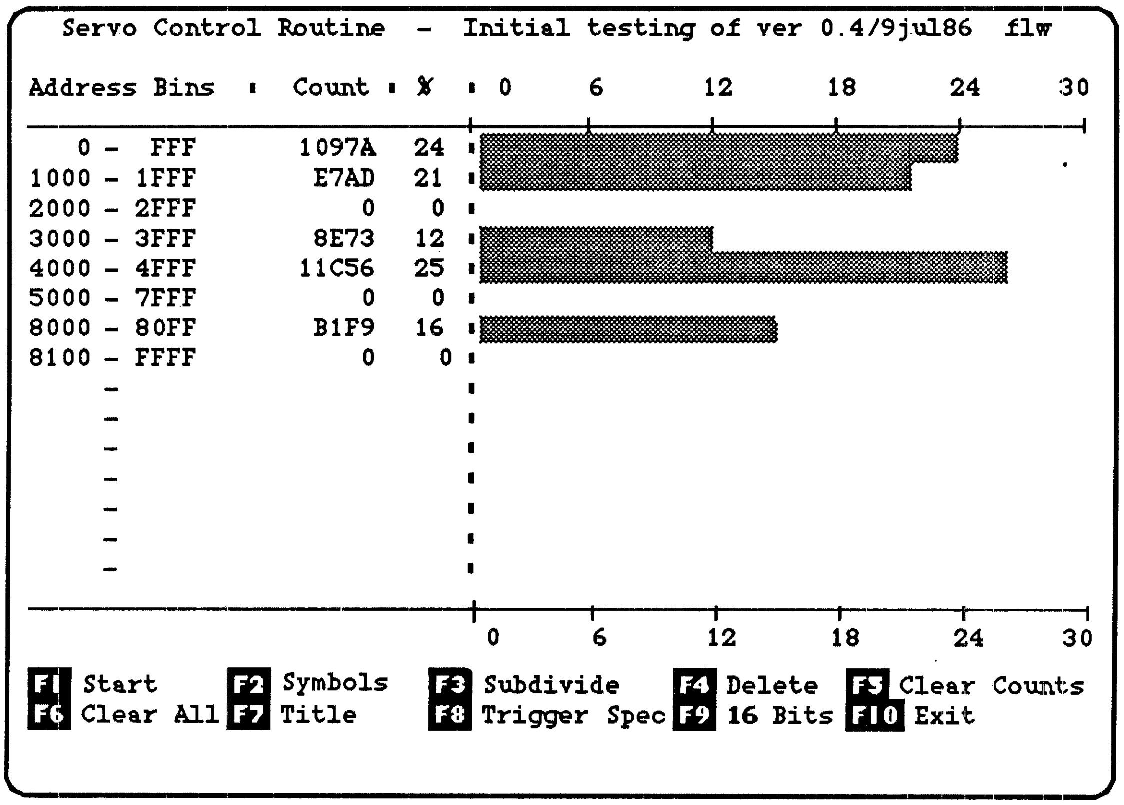
This menu lets you select the PPA commands with function keys:

F1 is equivalent to **THIST**.
F2 is equivalent to **AHIST**.
F3 is equivalent to **MHIST**.
F4 is equivalent to **HSAVE** (you will be prompted for a file name).
F5 is equivalent to **HLOAD** (you will be prompted for a file name).
F6 will allow you to go to the other menus. This is the same as pressing **F10** from the command mode.
F10 returns you to command mode.

1.6 The interactive screen: AHIST, MHIST and THIST

All data and commands are entered into interactive screens. The screen is called up with one of the three PPA commands, **AHIST, MHIST** or **THIST**. This section discusses all three modes. Sample screens of each appear in the next several pages.

The AHIST screen display

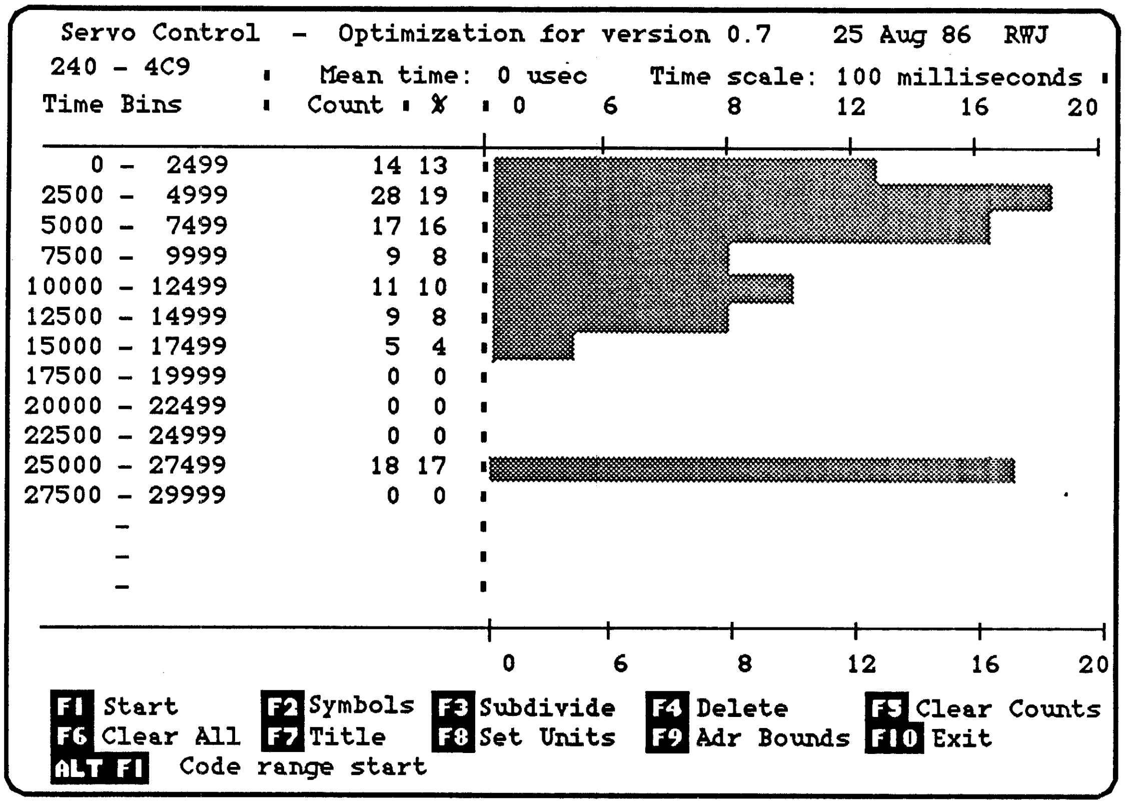


```
  Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Address Bins  ▮  Count ▮ %  ▮ 0     6      12      18      24     30

   0 -  FFF     1097A  24 ▮
1000 - 1FFF      E7AD  21 ▮
2000 - 2FFF         0   0 ▮
3000 - 3FFF      8E73  12 ▮
4000 - 4FFF     11C56  25 ▮
5000 - 7FFF         0   0 ▮
8000 - 80FF      B1F9  16 ▮
8100 - FFFF         0   0 ▮
       -                  ▮
       -                  ▮
       -                  ▮
       -                  ▮
       -                  ▮
       -                  ▮
       -                  ▮

                            0     6      12      18      24     30
F1 Start      F2 Symbols  F3 Subdivide    F4 Delete  F5 Clear Counts
F6 Clear All  F7 Title    F8 Trigger Spec F9 16 Bits F10 Exit
```

The top line

The top line of the screen is reserved for 80 characters of text. You can enter a title or notes into this field, after pressing **F7**.

The THIST screen display



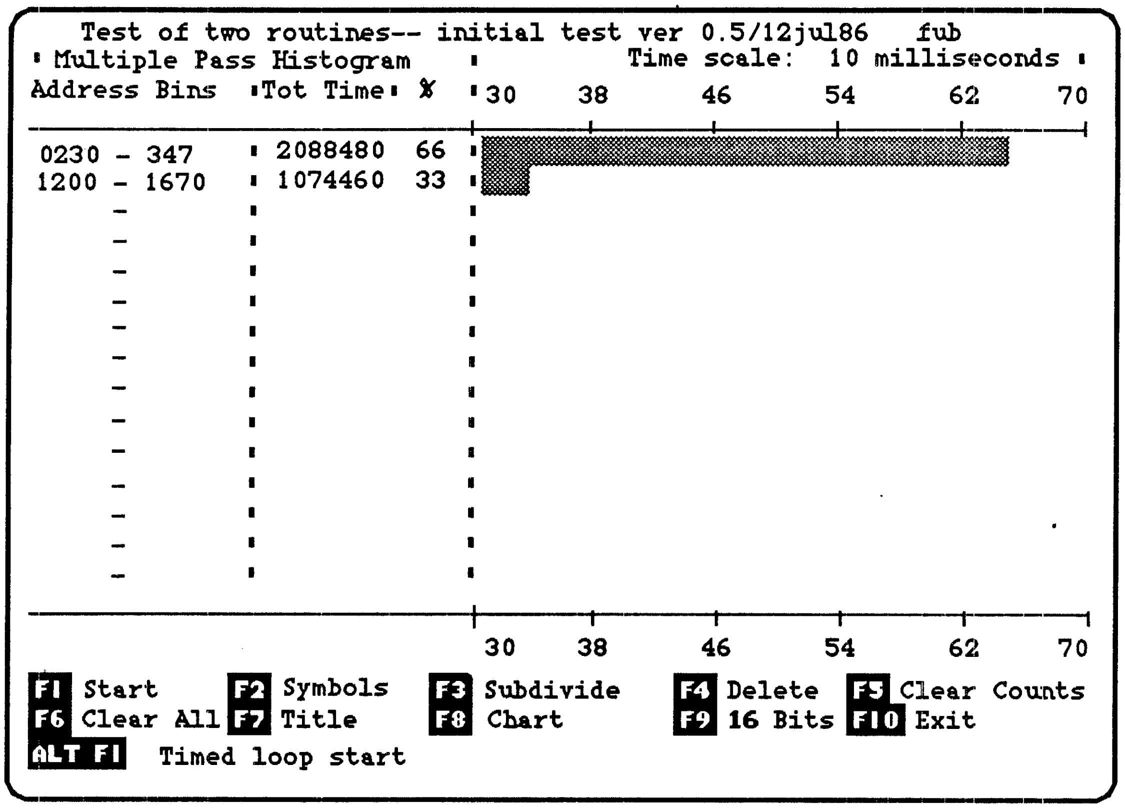
```
 Servo Control  -  Optimization for version 0.7    25 Aug 86  RWJ
240 - 4C9       ' Mean time: 0 usec    Time scale: 100 milliseconds '
Time Bins       ' Count ' %  ' 0     6      8      12     16     20

     0 -  2499      14 13 '
  2500 -  4999      28 19 '
  5000 -  7499      17 16 '
  7500 -  9999       9  8 '
 10000 - 12499      11 10 '
 12500 - 14999       9  8 '
 15000 - 17499       5  4 '
 17500 - 19999       0  0 '
 20000 - 22499       0  0 '
 22500 - 24999       0  0 '
 25000 - 27499      18 17 '
 27500 - 29999       0  0 '
       -                  '
       -                  '
       -                  '

                            0     6      8      12     16     20
F1 Start      F2 Symbols  F3 Subdivide  F4 Delete     F5 Clear Counts
F6 Clear All  F7 Title    F8 Set Units  F9 Adr Bounds F10 Exit
ALT F1  Code range start
```

The second line

The second line is blank in AHIST, but shows the address bounds, the mean time and the time scale in THIST. In MHIST it shows the time scale (currently unchangeable) and the name "Multiple Pass Histogram."

You set the value of the address bounds for THIST by pressing **F9** and entering hexadecimal values. You can also enter a symbolic label-- see the discussion on the bins and symbolic labels, in section 1.7.

MHIST graph display



```
   Test of two routines-- initial test ver 0.5/12jul86   fub
: Multiple Pass Histogram    :           Time scale:  10 milliseconds :
Address Bins  :Tot Time: %  :30     38      46      54      62     70
------------------------------+------+-------+-------+-------+-------+
 0230 - 347   : 2088480  66 :
 1200 - 1670  : 1074460  33 :
```

Column titles

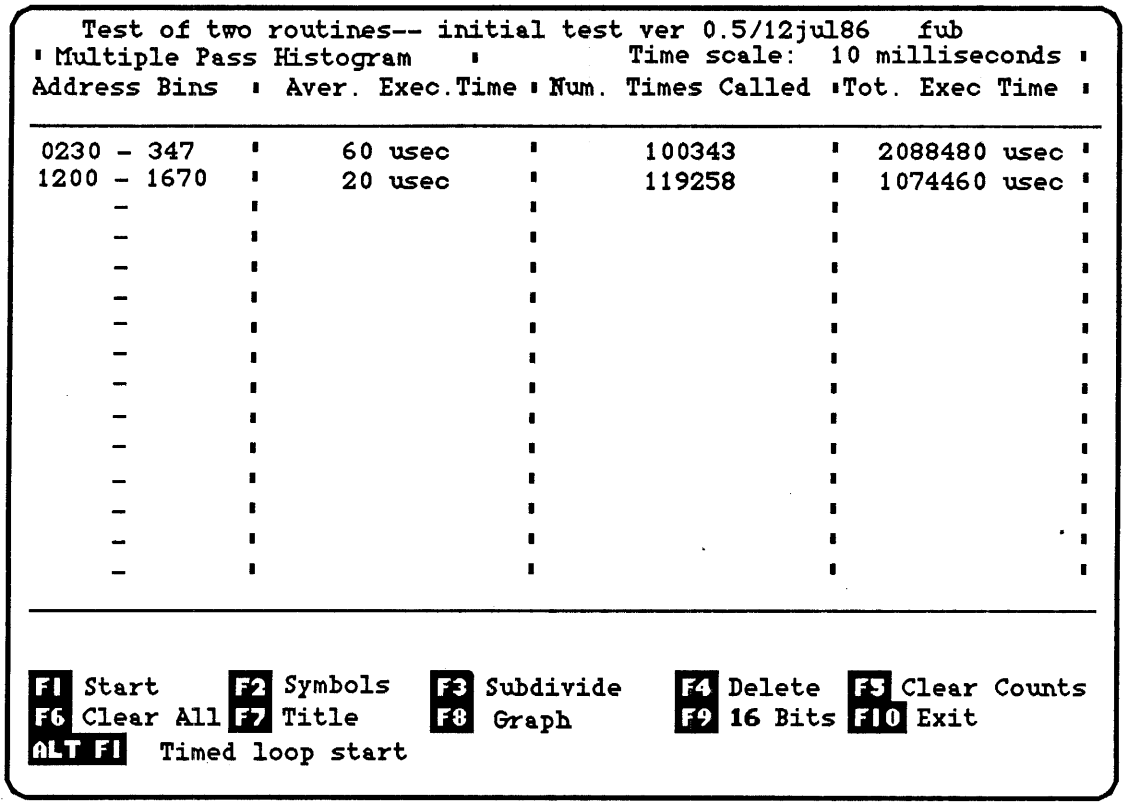
The third line contains the scale of the graph and the column titles.

The graph area

The most active portion of the screen shows a bar graph (histogram) of your program's execution. Each bar represents the proportion of activity or percentage of run-times that fall into the corresponding bin.

The scale across the top varies automatically as the data changes. It automatically adjusts the scale and the display to make the best use of the screen space.

MHIST chart display



```
   Test of two routines-- initial test ver 0.5/12jul86   fub
 ' Multiple Pass Histogram   '        Time scale:  10 milliseconds '
 Address Bins  ' Aver. Exec.Time ' Num. Times Called 'Tot. Exec Time '
 ----------------------------------------------------------------------
 0230 - 347    '    60 usec      '      100343       '  2088480 usec '
 1200 - 1670   '    20 usec      '      119258       '  1074460 usec '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
        -      '                 '                   '               '
 ----------------------------------------------------------------------

 F1 Start      F2 Symbols   F3 Subdivide    F4 Delete   F5 Clear Counts
 F6 Clear All  F7 Title     F8  Graph       F9 16 Bits  F10 Exit
 ALT F1   Timed loop start
```

Count and % Columns

The information expressed in the graph is also displayed numerically. The two columns immediately to the left of the graph show the raw counts and the percentages.

Function key menu

The menu of commands appears below the graph area. These commands are operated by the function keys, **F1** to **F10**.

Except for **F8** and **F9**, all of the functions are the same for both AHIST and THIST.

The message areas

The two lines at the bottom of the screen are used for error messages and other miscellaneous purposes.

The line immediately above the function key menu is used only for status messages. AHIST and THIST will display only the "Now collecting data" message, while MHIST will display one of four messages, depending on the status of the data collection process:

```
Collecting average execution times.
Collecting number of times called.
Paused: press any key to continue.
Paused: will resume in a moment.
```

1.7 The PPA and Symbolic Labels

On the left side of the screen you enter for each bin either a symbolic label or two numbers. Press **F2** to toggle between entering numbers and entering symbolic labels.

In AHIST and MHIST a bin is a range of memory addresses, such as 10 to A0. In THIST a bin is a range of time, such as 10 to 20 milliseconds. As the PPA collects information about your program, it sorts the data into your bins.

AHIST and MHIST addresses are always entered in hexadecimal format. THIST time values are always entered in decimal, while the address is entered in hexadecimal. . Time values are always output in decimal.

You can label each bin with a name, up to 14 characters long. When you enter a label the PPA will look in the symbol table for a symbol with the same name. The value of the symbol will be used as the lower limit of that bin.

If the PPA finds the symbol, then it will look for a second symbol, the same as the first except for an with "X" as a suffix. The PPA will use the value of this symbol as the upper limit, the eXit point.

For example, you can label the third bin with the name FUNC1. The PPA will use the value of the symbol FUNC1 as the lower limit, and the value of the symbol FUNC1X as the upper limit. Of course, if you have not defined the symbols or have not loaded the symbol file into the UniLab system, then the PPA will not translate them.

See section 1.5 of Chapter Six in the UniLab Reference Manual for detailed information on loading symbol files.

You can turn off this symbol translation feature by disabling the UniLab's symbol table, with **SYMB'**.

Press **F2** to toggle between the display of bin limits and the display of the bin name.

1.8 Saving Histograms

There are two ways to save the graph generated by a PPA--- you can save the data, or you can save the image. If you save the data, you can later reload the same histogram and look at the data or generate a new graph with the same setup. If you save the image of the screen then you can include the image in the text of a report.

Save the data

If you wish to save the setup of a histogram as a file, you may do so with the command **HSAVE** <filename>. Issue. this command after you exit from the histogram. This file includes both the setup information and the current state of the data at the time the program halted.

To re-load the saved histogram, you will use the command **HLOAD** <filename> . After HLOAD loads the data, it calls up the correct PPA mode.

Save the image

While in a histogram, press function key **F9** while holding down the **ALT** key (**ALT-F9**). You will then be prompted to type in the name of the file. When you enter a name and press the <CR> key, the image of the screen will be saved.

You can look at the image with the DOS command **TYPE**. You can also include the file in reports by importing the text file into your word processor. When you do this, you will probably have to play with the margin settings a little, to get the image to look right. The graphs use the full 80 columns of the CRT display, and you may have to edit the display, or use a compressed printing mode to print it out. The printout of the image will not show the bar graph correctly unless you have an IBM-compatible printer.

2. Ready the Target Program

2.1 Basics

The PPA analyzes a program running on your microprocessor control board. The program can reside either in the UniLab's emulation memory or in a memory chip on your board.

So, to ready the program for analysis, all you must do is:

> enable emulation ROM and load the program,
>
> OR
>
> disable emulation ROM and put your ROM chip on your board.

Program in emulation memory

Normally you will load your program into emulation memory, from either a disk file or a ROM or EPROM.

In either case you first enable emulation memory.

Program in ROM or EPROM

For final testing, you may wish to run your program directly from a memory chip on your target board.

In this case, you must first use the **EMCLR** command to disable emulation memory.

2.2 Run the program from emulation memory

Normally you will load your program into emulation memory, from either a disk file or a ROM or EPROM.

Enable memory

First you must enable emulation memory with the UniLab command

<start address> **TO** <end address> **EMENABLE**.

The address range for this command will depend on the memory map of the particular control board you are using. Refer to the UniLab Reference Manual, Chapter Six, Section 2, for a more detailed discussion. See also the entry for **EMENABLE** in Chapter Seven, **Command Reference**.

Load program from a disk file

Most people will use one of two commands to load a program from a file to emulation ROM, depending on the format of the file produced by your assembler or compiler:

1. Binary format. Load the file with:
   <from addr> <to addr> **BINLOAD** <filename>
   The filenames usually end in **.BIN**, **.COM**, or **.TSK**. The screen prompts for the filename if it is not included on the command line.

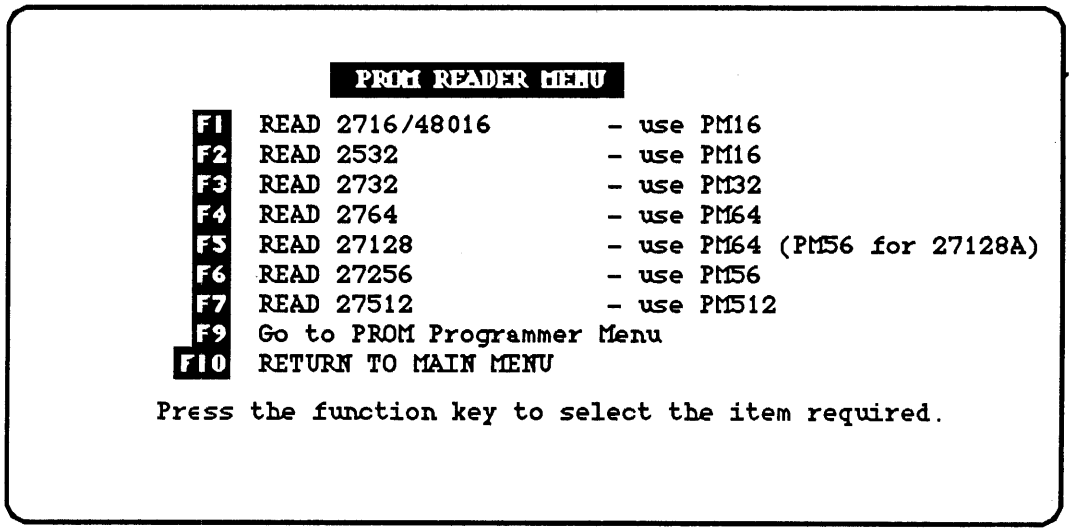
2. Intel^{tm}-format HEX object format. Load the file with:
   HEXLOAD <filename>
   You are prompted for the filename if it is not included on the command line. You do not specify a loading address with **HEXLOAD**.

The UniLab system also supports **HEXRCV** and **MLOADN** commands, for, respectively, loading from a remote system and loading from host RAM. Refer to the UniLab Reference Manual, Chapter Six, Section 2, on Readying and Loading Memory, as well as to Chapter Seven, Commands.

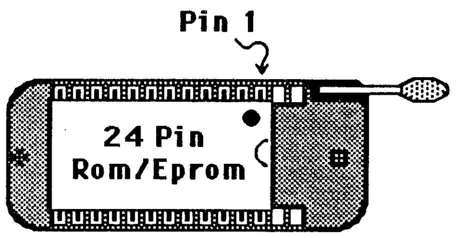
Read Program from ROM chip

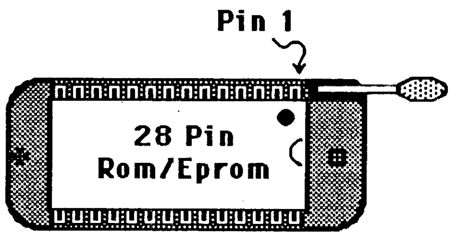
You can also use the UniLab software to read a program into emulation memory from a ROM or EPROM. UniLab supports all popular devices (refer to Appendix G in the UniLab Reference Manual).

To read a program from a ROM or EPROM, first get into UniLab command mode. Place the chip into the UniLab's prom socket, as shown below. Press **F10** for the main menu, then press **F9** to get the PROM reader menu.



```
                PROM READER MENU
F1   READ 2716/48016        - use PM16
F2   READ 2532              - use PM16
F3   READ 2732              - use PM32
F4   READ 2764              - use PM64
F5   READ 27128             - use PM64 (PM56 for 27128A)
F6   READ 27256             - use PM56
F7   READ 27512             - use PM512
F9   Go to PROM Programmer Menu
F10  RETURN TO MAIN MENU
Press the function key to select the item required.
```





Note that the 24-pin chips plug into the UniLab socket shifted all the way to the left.

2.3 Run program from a ROM on the Target Board

You may want to check the program as it runs in a ROM or EPROM on your control board. This will allow you to confirm that the program runs on the target board in its final form.

Enter **EMCLR,** to keep the UniLab emulation ROM off the bus. You can then use the PPA in the usual way.

3. The Address-Domain Analyzer

3.1 Simple Procedure

The program should already be in emulation ROM, or in a ROM or EPROM chip on your board, as described in the previous section of this chapter.

1. Enter the command **AHIST**, to bring up the blank AHIST screen.

2. Enter the desired address limits (in hexadecimal) into the two left columns or press **F2** and enter symbolic labels for each bin.

3. Press **F1** (Start).

Your microprocessor will start executing your code, and the PPA will start collecting data about your program. Press any key to stop program execution.

3.2 The Address-Domain Histogram

The Address-Domain Histogram shows the level of program activity in each range of addresses you specify. To analyze a new program, first load the program into memory. Then call up the AHIST screen with the command **AHIST**.

Specifying address ranges: Strategy

Memory addresses can be entered in several ways. The total range of addressable memory can be entered in one bin, and the range expanded over several bins using **F3** (Subdivide) and the cursor down key on the numeric keypad. This divides the address range equally among the bins. This is the preferred method for early studies of the program.

Alternatively, you can enter the range separately for each bin. This is the way to examine specific functions, or analyze non-contiguous sections of code.

When examining a small section of code, it may be useful to assign the remainder of memory to a single bin, so that activity outside of the section under review can be monitored. This strategy can backfire-- the PPA can end up missing the data you want to see because it is spending so much time monitoring and sorting the activity in the remainder.

Changing bin limits

When the sections of interest are identified, parts of the code can be discarded using **F4** (Delete). Other parts can be expanded for closer study using **F3** (Subdivide). Bin limits can be changed by simply typing over the address entries, so that a bin retains the same label. However, if the label is a valid symbol, the value of the symbol will override the number that you type in.

Symbolic labels

You can use a symbolic label to assign bin limits. Press **F2** to toggle between the display of addresses and the display of symbols.

After you enter a name, the PPA will search the symbol table.

If that symbol name is found it will use that value to replace the lower limit of the bin.

If it finds the symbol in the symbol table, the PPA will search for a symbol to use as the value for the upper limit. It looks for the <u>same</u> symbol name with an additional suffix "X."

Defining labels

The convention described above allows you to use a single name to refer to a functional range of memory.

You will have to use the appropriate label to mark the beginning and end of each range of interest in your source file. For example, label the entry point of your initialization routine with the name "PGM_START," and label the exit point of that routine with the name "PGM_STARTX."

You will then need to generate a symbol file with your assembler/linker and load that symbol file into the UniLab symbol table. See section 1.5 of Chapter Six of the UniLab Reference Manual for detailed information on symbols.

You can also make use of the UniLab command **IS** to define symbols. You will find this especially useful for defining the exit points of ranges that already have symbolic names for the entry points. For example, to define an area of memory 100 (hexadecimal) bytes long that starts at the already defined label "SORTLOOP," you type:

```
SORTLOOP 100 +   IS  SORTLOOPX
```

Many high-level languages create labels only for the entry points of functions. If you are working in a high-level language, you might find that you will have to use the **IS** command to create labels for the exit points.

Saving symbols

You can save the current symbol table with **SYMSAVE**, and load it again later with **SYMLOAD**. These commands are especially useful after you have defined symbols with **IS**, or selectively deleted symbols.

Use **SYMLIST** to get an ordered list of the current symbol file. You can then delete symbols with <symbol #> **SYMDEL**.

Labels and addresses

The PPA will automatically translate a symbol into a value when you enter the AHIST screen and each time you either move the cursor through a label field or enter a new name as a symbolic label. When the PPA finds a symbol, it overwrites the current value of the bin limit with the new value that it has found. Of course, the PPA does nothing if the label you specify is not in the current symbol table.

You can turn off the translation of symbols by disabling the UniLab symbol table. Use the UniLab command **SYMB'** to turn off the translation of symbols. **SYMB** re-enables this feature-- as do **IS**, **SYMFILE**, and **SYMLOAD**.

The address bus -- 16 or 20 bits

The address bins are normally interpreted as 16-bit numbers, even if you enter a five digit hex number (the most significant digit is ignored). Press **F9** to toggle between a 20-bit and 16-bit address bus.

A 20-bit address bus will allow the PPA to differentiate between addresses in different 64K segments. For example, 30000 and 40000 (hexadecimal) are both interpreted as the same address (0000) when you specify 16-bits, but are seen as two different addresses when you specify 20-bits.

It is a good idea <u>not</u> to select 20 bits unless you really need it. The high four bits of the address inputs to the UniLab hardware, A16 to A19, are usually not connected and "float high" on processors with a 16-bit address bus. This means that the UniLab sees the 16-bit address 0000 as the 20-bit address F0000. In some cases, such as the Z80, these high order address lines are used to tell the UniLab more about the activity of other control lines on the target microprocessor.

Therefore, unless your processor has a 20-bit address bus, the high four bits of the address should be ignored by the Program Performance Analyzer. This is the default condition.

Even if your processor has a 20-bit address bus, you should, for convenience, keep the PPA in 16-bit mode unless you need to distinguish between addresses in 64K segments. If you need to select addresses from a program that occupies more than one 64K segment, use **F9** to toggle to 20 bits and enter the additional digit in *all* addresses to specify the 64K segment (using ranges from 00000 to FFFFF).

Naming screens

F7 (Title) opens a field one line by 80 characters across the top of the screen. A title for the screen can be entered here, or notes to identify the screen should it be stored and retrieved at a later date. See subsection 1.8, Saving Histograms.

Modifying the trigger spec

The PPA uses the underlying UniLab command language to communicate with the UniLab hardware.

Function key **F8** gives you access to this same command language. You can use this function key to alter the trigger specification (the trigger specification tells the hardware what bus cycles to collect information on). You can alter the trigger spec so that data is not collected until after some event occurs, or so that only certain types of bus cycles are collected-- only reads, or only those with a certain data byte.

F8 opens a trap door into UniLab software environment. This means that once you press **F8** you can enter any UniLab command. However, we recommend that you only use this feature to alter the trigger spec and perform other simple commands, such as turning symbols on and off.

You can filter the trace by specifying some further criteria that each cycle must meet, besides addresses. For example, limit the trigger specification to only read cycles with the command: **READ**.

You can delay the collection of data until after a "qualifying event" has occurred, with the command **AFTER** <qualifying event>. For example, wait until address 4500 appears on the bus with the command: **AFTER 4500 ADR** .

You can safely change any aspect of the trigger except the trigger addresses, which are automatically set by the PPA. Refer to Chapter Six, Section 4, in the UniLab Reference Manual for more information on triggers.

After you press **F8**, you are back in the UniLab environment until you press <CR>. Remember that the function keys are assigned different commands in UniLab. Be aware that if UniLab commands are invoked for a purpose other than altering the trigger, then you might have to use the command **AHIST** to re-enter the PPA.

Clearing data

F5 clears acquired data, so that another program run can be performed using the same bin allocations. **F6** clears both data and bin allocations, so that you can enter new bin limits and names.

Exiting AHIST

F10 exits the AHIST screen, and returns the system to the UniLab environment.

3.3 The Function Keys

This subsection lists the names and uses of the function keys in the menu.

F1 START

Starts the collection of data. If RESET is enabled the target program restarts from the first address. This is the recommended procedure.

Press any key to stop the data collection.

F2 NAMES/ADDRESSES

Each bin can be given a symbolic label. Pressing this key toggles between the bin address range and the label. See the discussion under section 3.2 for more details.

F3 SUBDIVIDE

This key enables bins to be subdivided. Place the cursor on any bin and press **F3**, which causes SUBDIVIDE in the menu to be displayed in reverse video. Then move the cursor down, and press **F3** again. The initial bin range is now divided as equally as possible among the bins between the starting and ending cursor positions.

F4 DELETE

Place the cursor over a bin and press **F4**. The name and bin limits of the chosen bin are deleted, and the bins below move up to fill the space.

F5 CLEAR COUNTS

Pressing this key resets the data count to zero.

F6 CLEAR ALL

Pressing this key clears bin limits, bin names, and data count. (Y/N) confirmation is required.

F7 TITLE

This function allows an 80-column title or notes to be entered across the top of the screen.

F8 TRIGGER SPEC

This key opens a "trap door" to the UniLab environment, to enable the trigger specification to be modified.

CAUTION: All function keys revert to UniLab functions until <CR> is pressed. Do not press any function key until you have typed in the command to alter the trigger spec and pressed <CR>.

F9 16 BITS/20 BITS

Toggles between 16 and 20-bit addresses. You will normally leave this set to 16 bits, especially for processors such as the Z80 that only have a 16-bit address bus.

F10 EXIT

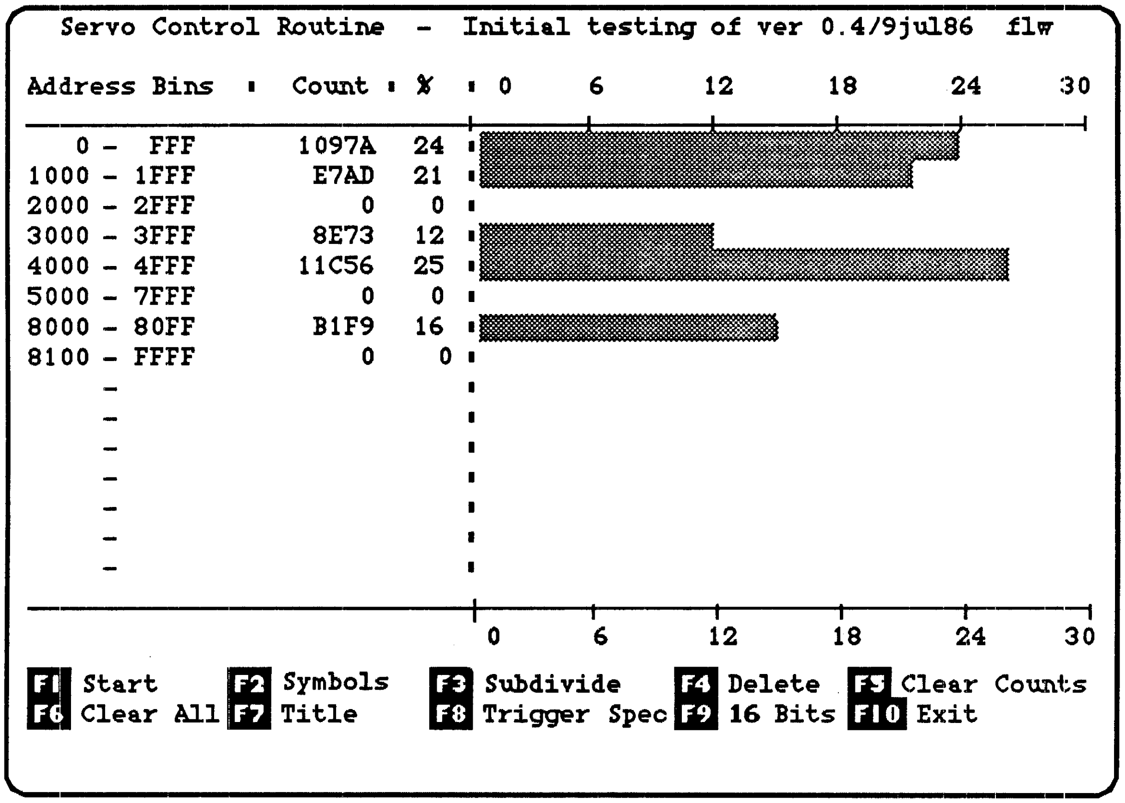
Returns you to the UniLab environment. Requires (Y/N) confirmation.

3.4 A Practical Example of AHIST

Suppose you have just completed version 0.4 of a program and you need to know how well it works. There is a main loop in ROM1 that accesses routines resident in ROM2 and ROM3. Unknown to you, there is a bug that causes a stack overflow. When this happens, the program attempts to write to ROM3 instead of to the stack. This occurs on a subroutine call, and the address on the stack tells the program where to return to. The program then tries to read back from ROM3 the address to RETURN to. At this point, the program goes berserk, accessing the wrong areas.

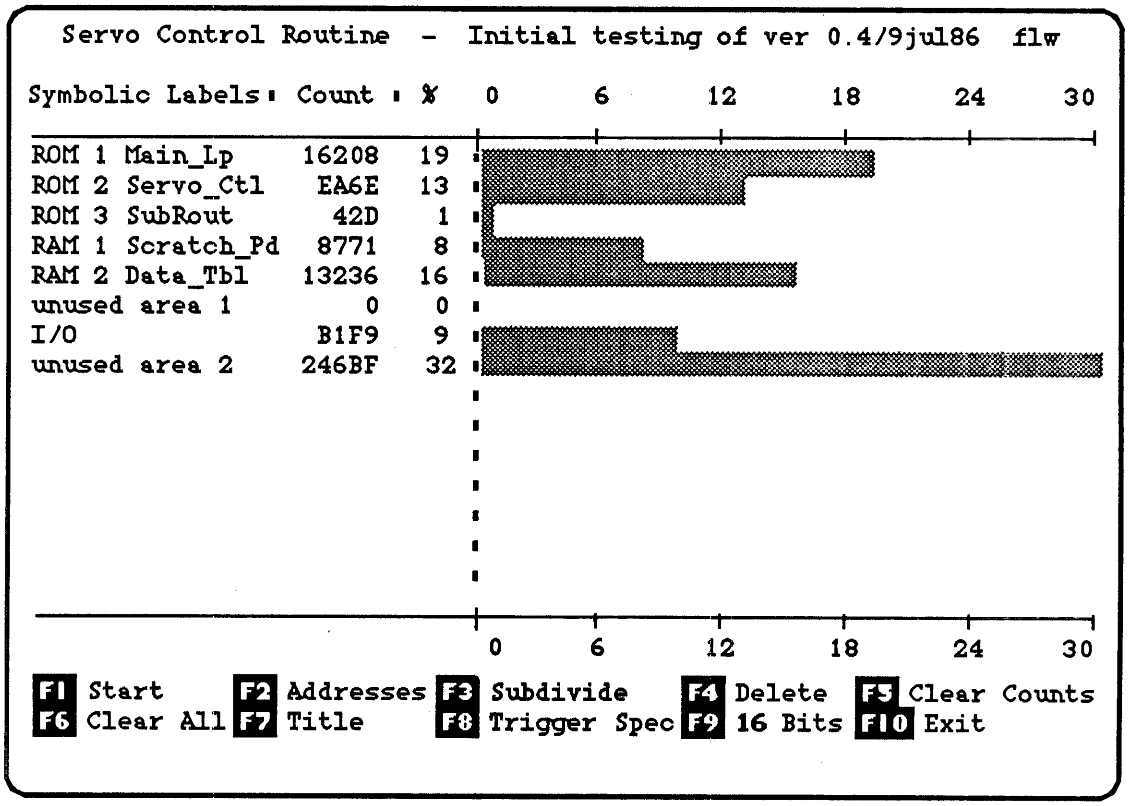
The program is run under AHIST. You type in address ranges for each chip on the board, the I/O and two unused address ranges. You use **F2** to label each bin, and press **F7** to give the screen a title.

You press **F1** to start the first run, which goes well for a time, executing the code in ROM1 and ROM2, as well as all four of the RAM chips. I/O operations also look good.



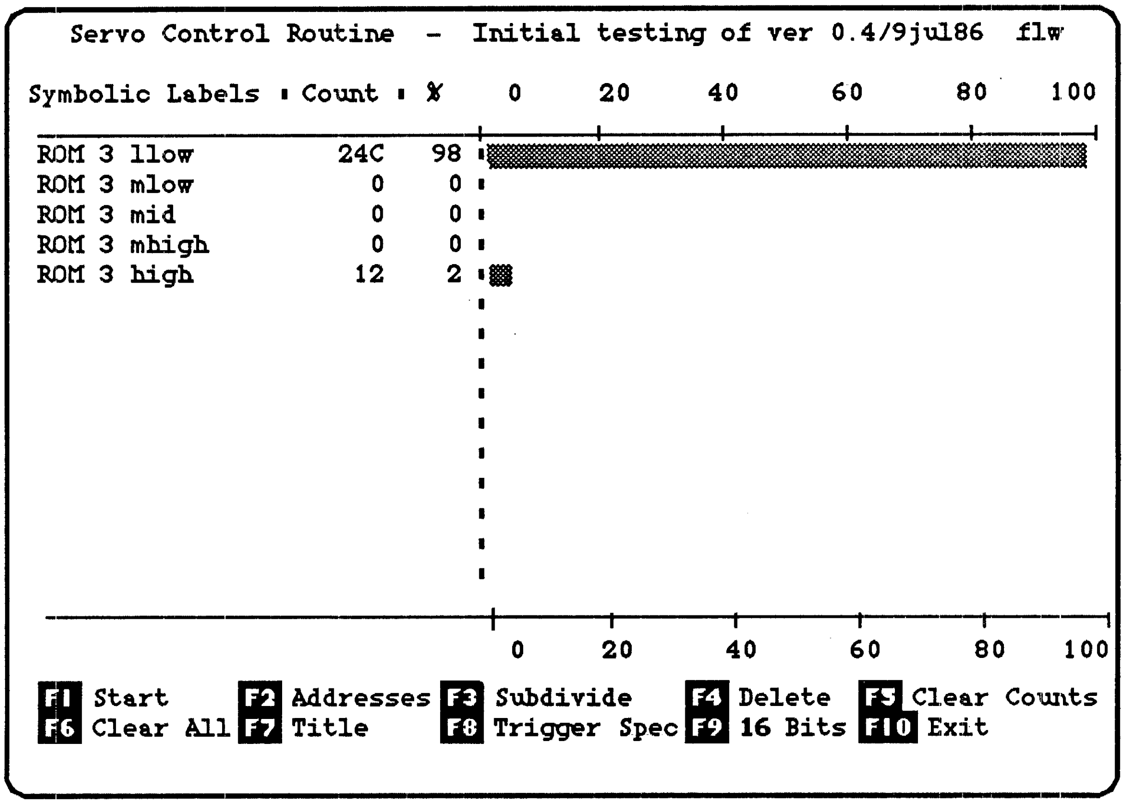
```
  Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Address Bins  ▮  Count ▮ %  ▮ 0    6     12    18    24    30
   0 -  FFF     1097A  24 ▮
1000 - 1FFF      E7AD  21 ▮
2000 - 2FFF         0   0 ▮
3000 - 3FFF      8E73  12 ▮
4000 - 4FFF     11C56  25 ▮
5000 - 7FFF         0   0 ▮
8000 - 80FF      B1F9  16 ▮
8100 - FFFF         0   0 ▮
       -                  ▮
       -                  ▮
       -                  ▮
       -                  ▮
       -                  ▮
       -                  ▮
       -                  ▮
                            0    6     12    18    24    30

F1 Start      F2 Symbols  F3 Subdivide     F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title    F8 Trigger Spec  F9 16 Bits  F10 Exit
```

Suddenly the program goes wild, running into an area of memory where it should not be. You can see that the program has reached ROM3 but not yet executed much code there. Hit any key to stop.



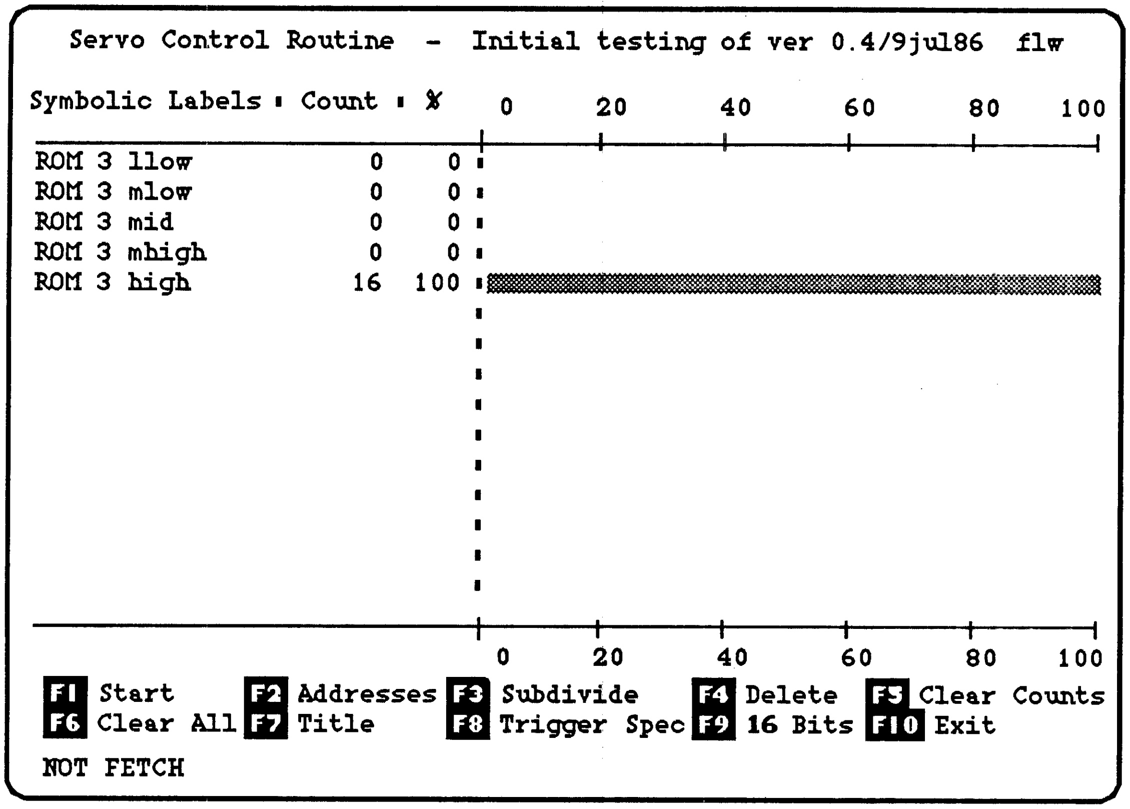
```
 Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Symbolic Labels: Count : %   0      6      12      18      24      30

ROM 1 Main_Lp     16208  19
ROM 2 Servo_Ctl    EA6E  13
ROM 3 SubRout       42D   1
RAM 1 Scratch_Pd   8771   8
RAM 2 Data_Tbl    13236  16
unused area 1         0   0
I/O                B1F9   9
unused area 2     246BF  32

                             0      6      12      18      24      30

F1 Start      F2 Addresses F3 Subdivide    F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title     F8 Trigger Spec F9 16 Bits  F10 Exit
```

You expand the ROM3 bin over more bins, using **F3** and the cursor down key, clear the data (**F5**) and run the program again. This time it crashes within the first section of ROM3. It mysteriously accesses the top of ROM3 twelve times.



```
   Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Symbolic Labels : Count : %    0     20     40     60     80    100

ROM 3 llow        24C    98
ROM 3 mlow          0     0
ROM 3 mid           0     0
ROM 3 mhigh         0     0
ROM 3 high         12     2

                               0     20     40     60     80    100

F1 Start      F2 Addresses F3 Subdivide    F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title     F8 Trigger Spec F9 16 Bits  F10 Exit
```

You now suspect a stack overflow. Just to be sure you change the trigger, pressing **F8** and entering **NOT FETCH** <CR>. The carriage return puts you back into AHIST.



```
  Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Symbolic Labels : Count : %    0     20     40     60     80    100

ROM 3 llow          0     0 :
ROM 3 mlow          0     0 :
ROM 3 mid           0     0 :
ROM 3 mhigh         0     0 :
ROM 3 high         16   100 :▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

                               0     20     40     60     80    100
 F1 Start      F2 Addresses F3 Subdivide    F4 Delete   F5 Clear Counts
 F6 Clear All  F7 Title     F8 Trigger Spec F9 16 Bits  F10 Exit
 NOT FETCH
```

You press **F1** to begin a new test. Now the histogram shows that the access to the top of the ROM is not a fetch. Since the program seemed to be running normally in the other expected areas of memory, this is probably a bad read or write. The histogram shows no access to ROM1 or ROM2, because the processor only fetches from those chips.

At this point you can leave the PPA and look at the program using the UniLab command language. What you see there verifies your findings. With this information, you make modifications necessary for version 0.5 to run perfectly.

4. The Time Domain Analyzer

4.1 Simple Procedure

The program should already be in emulation ROM, or in a ROM or EPROM on your board, as described in section 3 of this chapter.

1. Enter **THIST**, to bring up the blank THIST screen

2. Enter time limits into the time bins at the left of the screen.

3. Press **F9** and enter address bounds at the top left of the screen,
   OR
   press **F2** and enter a symbolic label.

4. Press **F1** to Start the program in the Entry-Exit mode,
   OR press **ALT-F1** to Start in the Code Range mode.

4.2 The Time-Domain Histogram

This feature shows you how long any specified routine takes to execute each time the routine executes. Simple routines will always take the same amount of time, unless they are interrupted.

Time bins and time scale

The collected times are sorted into the bins, for which you have specified time-ranges which bracket your guess at the actual execution time. For example, if you expect a routine to take about 10 milliseconds, then you could set the time scale to 1 millisecond (using **F8**) and subdivide a range of 0 to 30 over all the bins.

The histogram will show you the percentage of execution times that fall into each bin-range. The overall mean-time is shown at the top of the screen.

Starting THIST

There are two ways to start collecting data with the THIST PPA: Entry-Exit and Code Range. These two starts will give different results from the same setup.

The Entry-Exit start (**F1**) measures time starting when the UniLab sees the first address in the specified range. All activity, including calls and jumps to other routines, is measured until the program accesses the last address in the range.

The Code Range start (**ALT-F1**) records the time of execution from the initial fetch of any address in the code range until the program fetches an instruction from outside the range.

> ***IMPORTANT:*** The Code Range mode will only work properly if the macro **FETCH** is defined for your processor-specific software package. Check the Glossary section of your Target Application notes.

4.3 An Analogy for Understanding THIST

An analogy may help in understanding the working of the time histogram.

The investigation begins

Let's say that a woman, Rosalie, thinks that Kevin, her husband, is spending too much time in a particular tavern. She decides to get some hard data before confronting him. Rosalie develops a system where she goes to the bar every day to collect data on how long Kevin is there. When she comes home, she puts the day's reading into files: 0-2:00 hours, 2:01-4:00 hours, 4:01 to 6:00 hours, and so on. Once a week she looks through the files and updates her "bar" graph.

Subdividing

The second file starts to get fat, so she starts more files for that category, say 2:01-2:30, 2:31-3:00, 3:01-3:30, and 3:31- 4:00.

Entry-Exit

At first, when she goes to the bar, Rosalie parks outside where she can watch the entrance and exit. When she sees her husband go in the entrance, she starts her stopwatch, and lets it run until she sees him come out the exit. She then determines that he has spent 3 hours and 27 minutes in the bar.

This, of course, is the Entry-Exit method. Using this method, the unfortunate woman has no way of knowing that Kevin left the bar through the bathroom window and spent over 2 hours in the apartment across the back alley, before returning to the bar by the back way and finally leaving through the front door.

Code Range

Upon becoming aware of the situation, Rosalie changes her data-collection method. She takes up residence inside the bar (in a suitable disguise-- fake nose and glasses). She then observes whether her husband is actually present, running her stopwatch only while he is there. If Kevin leaves by the back door, the window or teleportation, Rosalie still knows how long he spent in the bar. She has, in fact, just invented the Code Range method.

The moral

The method you use depends on what you want to know. The wife in our example may in fact be more interested in how long her husband is away from home than in how long he is drinking at the bar. In this case, Entry-Exit would be perfectly appropriate.

The code-range method is better for "in-line" code-- a function or procedure that stays within a small area of memory while executing. Your code probably does not look like this, except for specialized assembly language routines.

More typically, your code will contain many jumps and branches, and most routines will call to subroutines. The entry-exit mode is more suitable for use with this type of assembly language coding. Code generated by high-level languages also tends to have this sort of structure. The entry-exit mode lets you see how much time is taken up by your routine, including any "errands" that it might run to other areas of memory.

4.4 The Time-Domain Function Keys

This is a list of the function keys displayed, together with their descriptions.

F1	start gathering data in the Entry-Exit mode.
ALT-F1	start gathering data in the Code Range mode.

F2 through **F7**
These keys have the same functions as the address-domain keys.

F8 **SET LIMITS**
This sets the units of the time scale: 10 microseconds, 100 microseconds, 1 millisecond, and 10 milliseconds. For example, a bin limit value of 2 means 20 microseconds when the scale is 10 microseconds. (Note that the screen uses the abbreviation usec for microseconds.)

Use the right arrow key to change the time scale; press the **END** key in the numeric keypad to return to the THIST screen.

NOTE: The resolution for the time histogram is accurate to within 20 microseconds. It will not be useful to set bin limit values in smaller increments.

F9 **ADR BOUNDS**
Address Bounds. These two addresses are entered in hexadecimal on the second line of the screen, or you can press **F2** and then enter a symbolic label. See the discussion of symbols in section 3.2 of this chapter for more information.

The address bounds are the starting and ending address of the code to be tested. <CR> returns the cursor to the bin limit section of the screen.

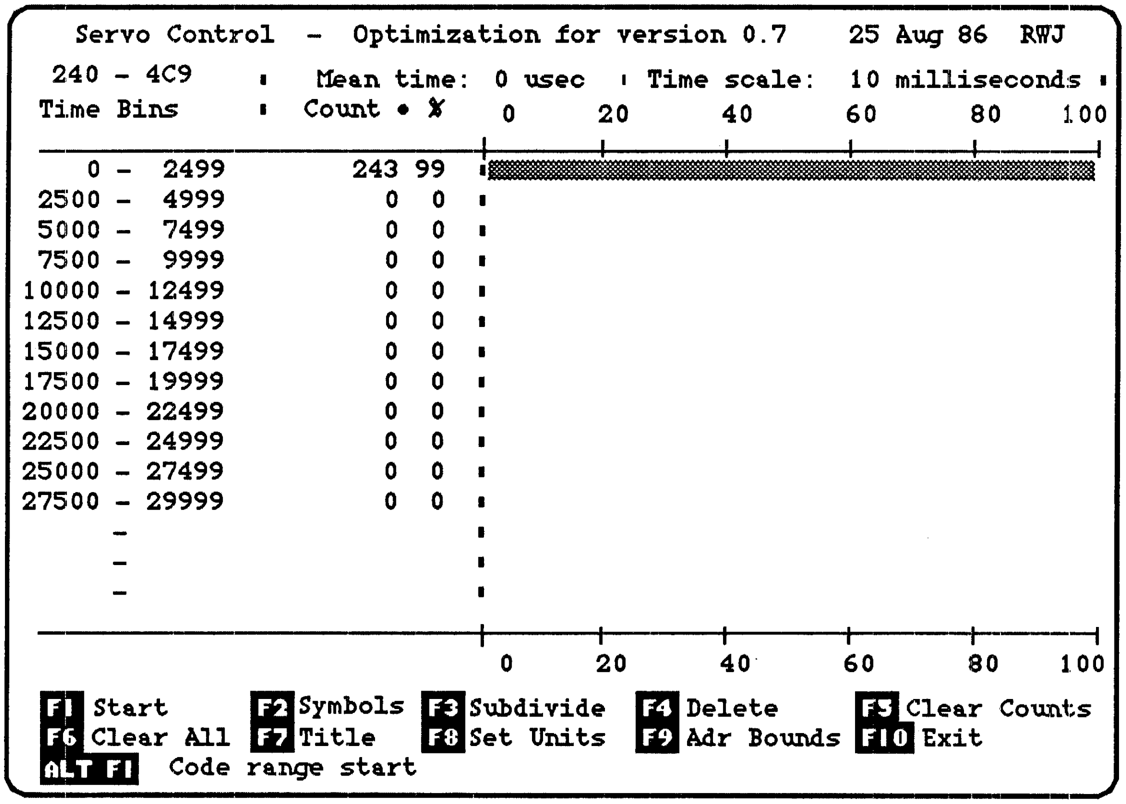
F10 **EXIT**
Returns you to the UniLab environment.

4.5 A Practical Example of THIST

Your program is running, and you're ready to begin optimizing the code. You decide to start by running a time test on the main loop. Unknown to you, an initialization routine is called repeatedly, and takes a long time to execute. The routine should only be called once.

You enter **THIST** and get the blank screen. You enter the address range (**F9**), using the start and stop addresses of the main loop. You set the scale to 10 milliseconds (**F8**) using the right arrow to set the time, and the **End** key (same as numeric key pad #1) to return to the THIST screen. This relatively large time scale ensures that even events that take longer than you would think are covered. You enter a time range of 0 to 30000 into the top bin, and subdivide the bin into twelve bins (press **F3**, then the down arrow 11 times to indicate a total of 12 bins, and press **F3** again)

. You press **F1** to start and run in Entry-Exit mode. All the data falls into the top bin.

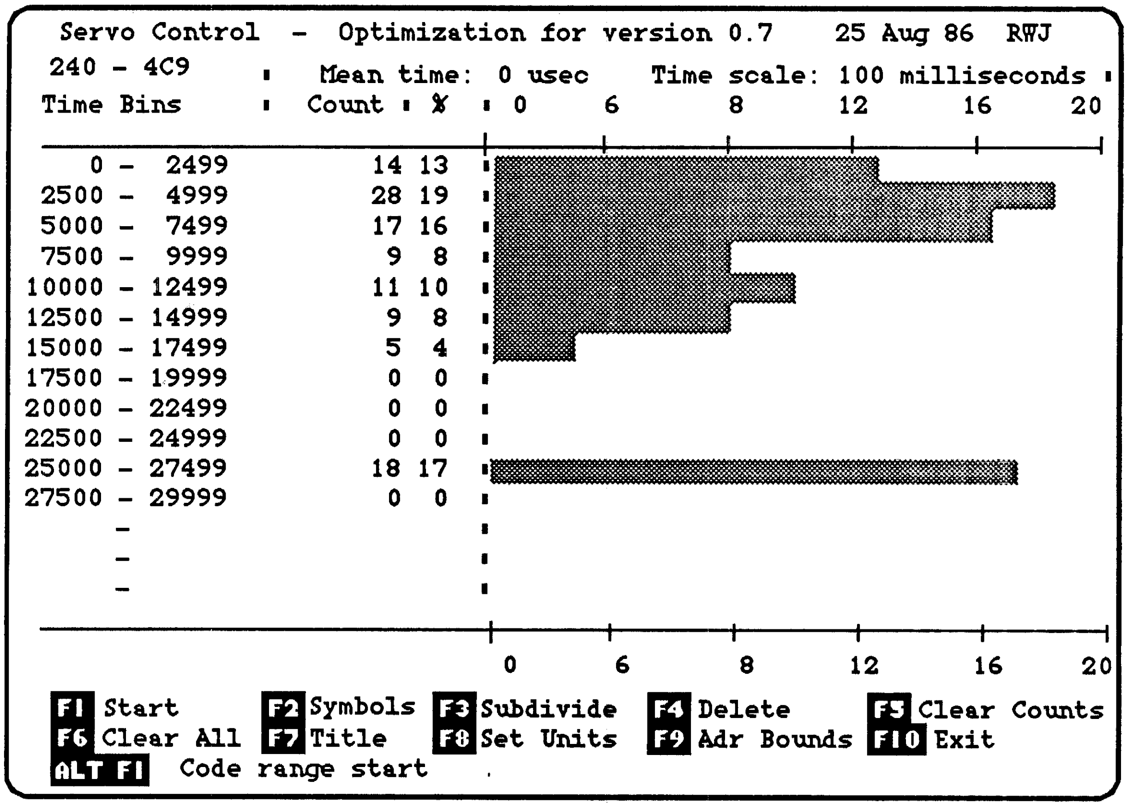


```
  Servo Control  -  Optimization for version 0.7    25 Aug 86  RWJ
 240 - 4C9      |  Mean time:  0 usec  | Time scale:  10 milliseconds |
Time Bins       |  Count • %   0      20      40      60      80     100
                               +-------+-------+-------+-------+-------+
    0 -  2499      243 99 |▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
 2500 -  4999        0  0 |
 5000 -  7499        0  0 |
 7500 -  9999        0  0 |
10000 - 12499        0  0 |
12500 - 14999        0  0 |
15000 - 17499        0  0 |
17500 - 19999        0  0 |
20000 - 22499        0  0 |
22500 - 24999        0  0 |
25000 - 27499        0  0 |
27500 - 29999        0  0 |
      -                   |
      -                   |
      -                   |
                               +-------+-------+-------+-------+-------+
                               0      20      40      60      80     100
F1 Start      F2 Symbols  F3 Subdivide  F4 Delete      F5 Clear Counts
F6 Clear All  F7 Title    F8 Set Units  F9 Adr Bounds  F10 Exit
ALT F1   Code range start
```

Since you don't like this display, you clear the counts with **F5**.

Then you change the time scale to 100 microseconds with **F8**, and run the program again.

You now get a much better display of data. The times fall into several bins-- including a surprising number in the 25000-27499 bin. You suspect that the large number of execution times indicates a problem. You decide to investigate further.



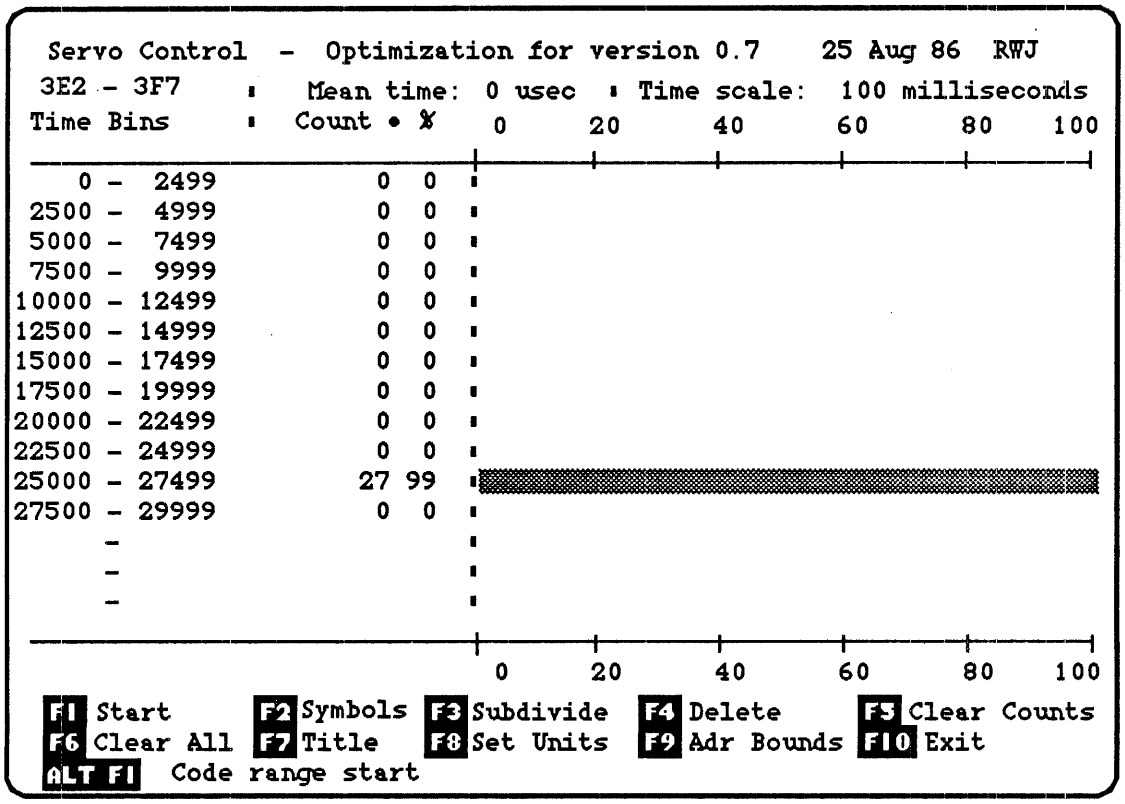
```
 Servo Control  -  Optimization for version 0.7    25 Aug 86  RWJ
240 - 4C9      ▮  Mean time: 0 usec    Time scale: 100 milliseconds ▮
Time Bins      ▮  Count ▮ %  ▮ 0     6       8       12      16     20

    0 -  2499        14 13 ▮
 2500 -  4999        28 19 ▮
 5000 -  7499        17 16 ▮
 7500 -  9999         9  8 ▮
10000 - 12499        11 10 ▮
12500 - 14999         9  8 ▮
15000 - 17499         5  4 ▮
17500 - 19999         0  0 ▮
20000 - 22499         0  0 ▮
22500 - 24999         0  0 ▮
25000 - 27499        18 17 ▮
27500 - 29999         0  0 ▮
      -                    ▮
      -                    ▮
      -                    ▮

                             0     6       8       12      16     20
 F1 Start      F2 Symbols  F3 Subdivide  F4 Delete      F5 Clear Counts
 F6 Clear All  F7 Title    F8 Set Units  F9 Adr Bounds  F10 Exit
 ALT F1  Code range start
```

In order to narrow down the problem area, you clear the counts again (**F5**) and change the second address (**F9**) to point at the half-way mark of the main loop. This time when you press **F1** you don't get any long execution times.

You now know that the long execution times are caused by code that either resides in or is called from the second half of the main loop. You press **F9** again, and change both the starting and ending address, so that you are examining only the third quarter of the main loop.

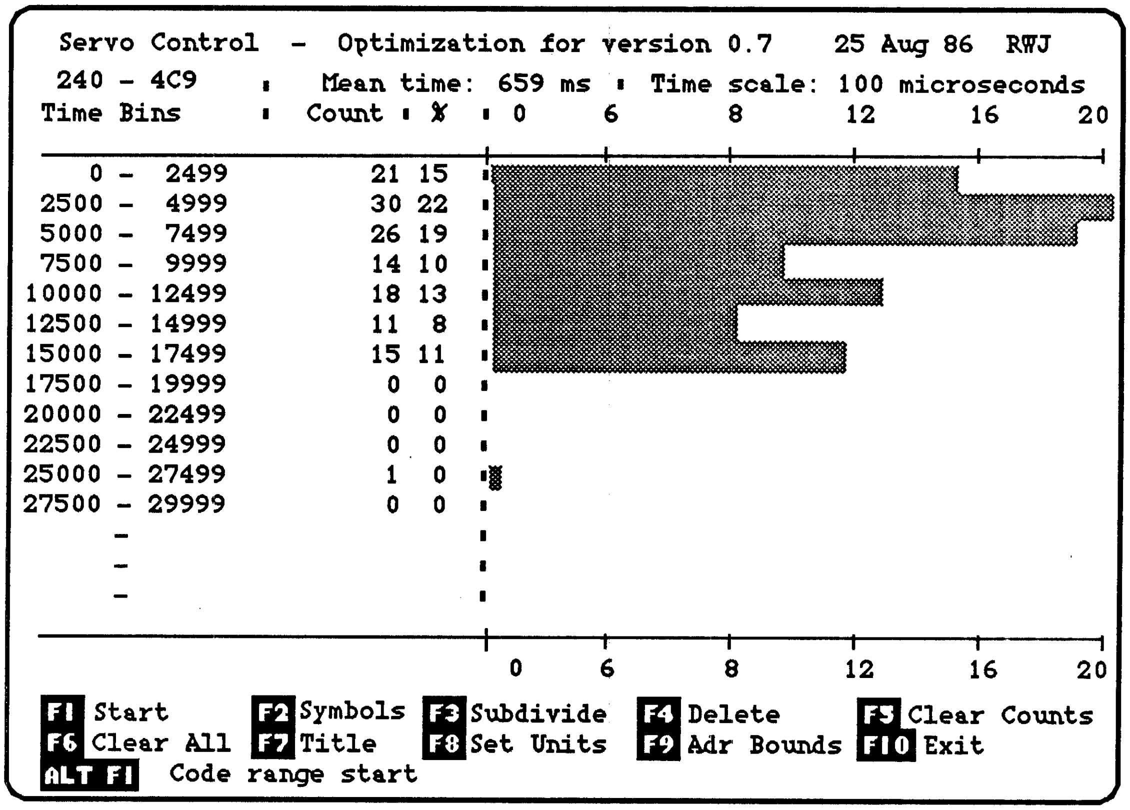
This time, the code often takes a long time to run. You look at the listing for that part of your program and find a call to an initialization routine that you suspect is delaying the program execution. You clear the counts again, and this time enter addresses just before and just after the suspicious call .

You press **F1** once again, and immediately see that there are multiple, lengthy run times.



```
 Servo Control  -  Optimization for version 0.7    25 Aug 86  RWJ
 3E2 - 3F7      :  Mean time:  0 usec  : Time scale:  100 milliseconds
Time Bins       :  Count • %    0     20     40     60     80    100
-------------------------------+------+------+------+------+------+
    0 -  2499          0  0  :
 2500 -  4999          0  0  :
 5000 -  7499          0  0  :
 7500 -  9999          0  0  :
10000 - 12499          0  0  :
12500 - 14999          0  0  :
15000 - 17499          0  0  :
17500 - 19999          0  0  :
20000 - 22499          0  0  :
22500 - 24999          0  0  :
25000 - 27499         27 99  :▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
27500 - 29999          0  0  :
        -                    :
        -                    :
        -                    :
-------------------------------+------+------+------+------+------+
                                0     20     40     60     80    100
 F1 Start      F2 Symbols  F3 Subdivide  F4 Delete     F5 Clear Counts
 F6 Clear All  F7 Title    F8 Set Units  F9 Adr Bounds F10 Exit
 ALT F1  Code range start
```

Now the program can be corrected, and the test of the routine rerun. You have found the bug and exterminated it. The trace of the corrected program shows, properly, only one very long execution time, indicating that the initialization routine is now called only once.



```
 Servo Control  -  Optimization for version 0.7    25 Aug 86  RWJ
 240 - 4C9      :  Mean time: 659 ms : Time scale: 100 microseconds
Time Bins       :  Count : %  : 0     6      8      12     16     20

    0 -  2499        21 15
 2500 -  4999        30 22
 5000 -  7499        26 19
 7500 -  9999        14 10
10000 - 12499        18 13
12500 - 14999        11  8
15000 - 17499        15 11
17500 - 19999         0  0
20000 - 22499         0  0
22500 - 24999         0  0
25000 - 27499         1  0
27500 - 29999         0  0
      -
      -
      -
                              0     6      8      12     16     20
F1 Start       F2 Symbols  F3 Subdivide  F4 Delete      F5 Clear Counts
F6 Clear All   F7 Title    F8 Set Units  F9 Adr Bounds  F10 Exit
ALT F1   Code range start
```

5. The Multiple-Pass Address-Domain Analyzer

5.1 Simple Procedure

The program should already be in emulation ROM, or in a ROM or EPROM chip on your board, as described in section 3 of this chapter.

1. Enter the command **MHIST**, to bring up the blank MHIST chart screen (NOTE: if you enter MHIST after leaving AHIST, then the bin limits will be preserved. Within MHIST you can use **F6** to clear the bin limits).
2. Enter the desired address limits (in hexadecimal) into the two left columns or press **F2** and enter symbolic labels for each bin. Note that MHIST allows overlapping bins, or nested ones for that matter.
3. Press **F1** to Start collecting data in Manual Loop mode,

   OR

   press **ALT-F1** to Start in Timed Loop mode.

In **Manual Loop** mode, your microprocessor will start executing your code, and the PPA will determine the average execution time of the first bin. You can press any key to make the PPA pause, then when you press any key again, the PPA will collect data on the second bin. When the PPA has the average execution time of all bins, it will start collecting the number of times called for each bin. So to move from bin to bin you will need to press a key to pause and then again to continue.

In **Timed Loop** mode, you will be prompted to enter the length of time to collect data on each bin. You must enter this (decimal) number in milliseconds. The maximum is approximately 65000 milliseconds, or 65 seconds. The PPA will then determine the average execution time and the the total number of times called for each bin. You will not need to press another key after you enter a valid number and press carriage return. If you do press a key, you will prematurely stop collecting data on the current bin.

In **either** mode, you can stop at any time by pressing either **F10** or the **ESC** key. And in both modes the figures you get will be expressed in milliseconds and will be accurate to within 20 milliseconds.

5.2 The Multiple-Pass Address-Domain Histogram

See also section 3.2. The information in that section applies to MHIST, except for the discussion of specifying trigger specs. Additional information specific to MHIST appears below.

The Multiple-pass Address-Domain Histogram shows the average execution time, number of times called and total execution time of each bin. The average execution time is approximate, while the number of times called is exact. To analyze a new program, first load the program into memory. Then call up the MHIST screen with the command MHIST.

Changing from Chart to Graph

When you call MHIST, it will start up displaying the Chart screen. This display shows you the average execution time, the number of times called and the total execution time for each bin.

Press **F8** to toggle between chart display and the graph display. The graph display shows you only the total execution time and a histogram of the total execution time.

Both displays highlight the number that is being altered while gathering data.

Getting valid results

The results that MHIST gives you are only valid if your target software is executing the same series of instructions each time you start it up. For this reason you should use one of three strategies:

1) The simplest is to have reset enabled (**RESET**), which will cause the target program to start over from the beginning each time. With this method you can perform a timed loop start.
2) Disable reset (**RESET'**) and manually start some operation on your target system at the start of data collection for each bin. With this method you would want to perform a manual loop start.
3) Disable reset (**RESET'**) and use the stimulus outputs to trigger some operation on your target system at the start of data collection for each bin. MHIST automatically sends a strobe out on the stimulus outputs at the start of data collection for each bin. With this method you can use a timed loop start.

MHIST normally holds the outputs S0 through S3 low and S4 through S7 high. Just before MHIST starts collecting data on each bin, it reverses these outputs and then returns them to their usual values.

line #:	0	1	2	3	4	5	6	7
normal value:	0	0	0	0	1	1	1	1
value during strobe:	1	1	1	1	0	0	0	0

Note that MHIST sends these signals out whether or not you use them.

Additional stimulus information

You can only use the stimulus outputs if your program reads an input value and then takes an action when a certain value appears-- or if you are willing to write the extra code to test the value and take an action. The best way to use these stimulus outputs is to look for the positive-going or negative-going edge.

To use the stimulus outputs you need to connect the stimulus cable to the PROM programmer socket, as shown below. The ends of the cables are labeled.

See the UniLab Reference Manual, section 8 of Chapter Six, if you need additional information on the stimulus outputs.

5.3 An analogy for MHIST

An analogy may help in understanding the multiple-pass histogram.

A suspicion is hatched

Divorce seems imminent for Rosalie and Kevin, the couple who starred in the THIST analogy.

Kevin has become suspicious of his wife, and has decided to keep track of how she spends her time. He decides that if would be too obvious to tail her. Instead he will watch outside of three locations he knows Rosalie frequents: her place of work, a health spa and an cappuccino bar.

Catching the average time

He spends a day in front of each place, watching when she enters and leaves. Each time she leaves he writes down the elapsed time and resets his stopwatch. That way he is able to determine how long she spends inside each place, on the average.

A data collection problem

Kevin realizes that his wife might only step outside for a breath of fresh air, and then immediately go back inside. He would not be able to spot these entrances, since he is busy writing down the elapsed time and resetting his watch. So Kevin doesn't even know if Rosalie behaves this way.

MHIST has the same difficulty-- while the PPA is recording the elapsed time and resetting the clock, your program could re-enter the routine you are monitoring. That's why you need the second pass through each bin.

Determining the number of events

Kevin, who has lost his job by now, decides to spend another three days on the project. This time, he spends a day in front of each place, just watching the entrance and keeping track of how often his wife goes in. He figures that once he knows how often Rosalie goes into each building, he can multiply the average stay by the number of stays and thus calculate the total amount of time she spends in each building.

The husband's assumptions

Kevin has made two assumptions:

1) Rosalie's behavior is constant from day to day.
2) The average time he calculated is valid, even though it might not include <u>all</u> of the visits Rosalie made to each location. That is, Kevin assumes that the visits he misses (when determining average time) do not deviate from the mean.

Of course, we make similar assumptions when using MHIST.

Kevin's method: stimulus and response

Kevin has cleverly manipulated Rosalie, to protect the first assumption. He knows that she will always follow the same routine after they have an argument. So on the morning of every day that he wants to gather data, he starts the same fight with her.

Of course, this couple has been having the same argument every morning for the last five years, so Rosalie suspects nothing.

The moral

Remember that your program must perform the same operation during each pass, or your results will not be valid.

As for Kevin and Rosalie: they were thinking of divorce, but then realized that no one else would be able to put up with their bizarre behavior.

5.4 The Function Keys

This subsection lists the names and uses of the function keys in the menu.

F1 START

Starts the collection of data. If RESET is enabled the target program restarts from the first address each time the PPA starts to collect data on a bin. This is the recommended procedure.

Press any key to pause, and then press any key again to continue with the next data collection.

The PPA will stop collecting data when you have looped through all the bins twice-- once for average execution time and once for number of times called. Or you can press **F10** or the **ESC** key to stop the data collection

ALT-F1 TIMED LOOP START

Starts the collection of data in the timed loop mode.

F2 through **F7**

These keys have the same functions as they do under AHIST.

F8 CHART/GRAPH

This key toggles between the chart display and the graphical display of the data collected by MHIST. When you are collecting data, you will probably want the Chart display, since it gives you a clear idea of what the PPA is doing.

F9 16 BITS/20 BITS

Same as under AHIST, toggles between 16 and 20-bit addresses. You will normally leave this set to 16 bits, especially for processors such as the Z80 that only have a 16-bit address bus.

F10 EXIT

Returns you to the UniLab environment. Requires (Y/N) confirmation.

6. Troubleshooting

6.1 Operating problems

Here are some hints that will help you avoid problems.

Target Program Loading

If the program is run in the emulation memory, the proper range of emulation ROM must be enabled (**EMENABLE**). If the program is run from the chip, the emulation ROM must be disabled (**EMCLR**).

The **RSP'** command can be used to disable the debug for a completely transparent emulation during the use of the PPA. When you do this, be sure to re-enable the debug with **RSP** before attempting to set a breakpoint.

If the debug is not disabled, it will insert code into the ORION reserved area for your processor. Refer to the UniLab Reference Manual and the Target Application notes for further information.

Symbolic labels

Use the UniLab command **SYMLIST** to verify that your symbol file has loaded properly. You can use <symbol #> **SYMDEL** to delete any unwanted symbols.

The PPA will look at all the symbolic label fields whenever you enter AHIST or THIST. If it finds a symbol then it will update the address bin. If you don't want it to do this, then disable the symbol table with **SYMB'**. You can enable the symbol table again with **SYMB**. Several other commands also re-enable the symbol table: **IS**, **SYMFILE**, and **SYMLOAD**.

The PPA will not try to find the symbol for the exit address (with suffix "X") unless it finds the symbol for the entry address.

If AHIST or MHIST Does Not Run Properly:

- check that the 16-bit address (**F9**) is selected, unless the program operates across
  64K segments.

- make certain that RESET is in the state you want (either enabled or disabled).

If THIST Does Not Run Properly:

- check that the address range entered is within the range of the program under test.

- check that the starting and ending addresses are correct.

If you are getting the wrong information using **ALT-F1** (Code Range mode):

- make sure that the macro **FETCH** is defined for your processor- specific software package.

- if you have a processor that has "extra" bus cycles, as many Motorola processors do, make certain that those bus cycles do not appear to be fetches from an address outside of the normal code range.

If you get an "RS-232 error" after using the PPA

The UniLab hardware can end up in an indeterminate state if the PPA is exited from abnormally. The command **INIT** will not work-- but all you have to do is turn the UniLab off for a second, turn it on again and then type in **INIT**.

6.2 PPA Error Messages

Bad Range - can't subdivide - This error occurs if you try to subdivide a bin with an invalid or missing number, or one that has the lowbound larger than the highbound.

Boundaries for bins overlap - AHIST and THIST will not produce a histogram until this error is fixed. It occurs if any two ranges of addresses or times share a region. For example,

1000-2000 and 1500-2500

or even just

1000-2000 and 2000-3000.

Can't create file - This error occurs if the name you specify for the file is an invalid name, or if DOS cannot create a file for some other reason.

Disk full - This error occurs if you try to use **TSAVE** or try to save a screen image file when there is not enough room on your disk.

File not found - If you try to load a non-existent file with **HLOAD**, you will get this message. Common errors are misspelling, using the wrong file extension, or not specifying the proper path. You can look at the disk directory at any time by pressing **F10** to leave the PPA display and then typing

DOS DIR (or **DOS DIR A:** or any valid DOS command).

You must be in the UniLab environment (command mode) to do these DOS functions.

Invalid or missing number - This error occurs if you try to run a test with no bin defined, or with one limit defined but not the other, or with a label entered but no values, or with a numeric field containing a value that is not a number in the base you are using. For example, FF is not a number in decimal (which THIST requires). You will not be able to produce a histogram until you correct the mistake.

Invalid start and stop address for THIST - This error tells you that one of the two address bounds that you gave to THIST is missing or is not a number in the base you are using.

Lowbound is larger than highbound - This error occurs if a bin has a starting value that is higher than the ending value. You cannot produce a histogram until you correct the mistake.

Not enough bins available - This error occurs if you try to allot more than 15 bins using **F3** (Subdivide). This can occur if you already have several bins alloted and then try to subdivide one of them among all the bins. Be sure to delete enough bins to allow room for expansion.

RS-232 error #XX - This error can occur after an abnormal exit from the PPA software. You will have to turn off the UniLab, wait a second and turn it back on. Then you will be able to type the command **INIT**.

7. Specifications

7.1 Operating method and limitations

AHIST

The AHIST PPA works by collecting 170 bus cycles at a time in a trace buffer and sorting them into the bins that you specify. The PPA filters the data it gathers, so that the addresses of the 170 cycles fall between the highest and lowest address that you specify.

Generally, you can be confident of your results once the percentage data has stabilized.

However, this method has two main limitations:

1) If you are looking exclusively at a range of code that seldom occurs, you will get a "shadow" effect. The first 170 cycles of the routine will be gathered, and then the PPA will not gather data during the time it is sorting the data into bins.

2) If you are looking at a very small range of code that occurs periodically, and also looking at a range of code that occurs continuously, you can get a "swamped-out" effect. The trace buffer will be continually filling up with cycles from the code that is continuously executing, and might miss the execution of the small code range.

MHIST

The multiple-pass histogram collects average times the same way that THIST does, and then collects the number of times called by keeping a count of the number of times that the first address is accessed.

The times are accurate to within 20 milliseconds, and the count of the number of times called is accurate up to FFFF hex, provided that the first address is accessed only once each time the routine is called.

THIST

The THIST PPA works by timing the duration of a function. It sorts each sample into a bin as soon as it finds one. You can have confidence in the results to within 20 microseconds.

Chapter Five: On-Line Help

Contents

Introduction

The UniLab software provides you with extensive on-line help. The help facilities give you the assistance you need to solve problems and to avoid confusion.

The Menu system, demonstrated in chapter two and fully mapped out in chapter three, helps you gain familiarity with the use of the UniLab instrument. With the menus, you can work with the UniLab right away and learn the command language as you go.

The command **MESSAGE** will give you information on the most recent updates and additions. Your distribution diskette includes a **README.TXT** file which contains update information, some of which applies to all UniLab software, some of which applies only to specific Disassembler/DEBUG packages.

The On-Line Help covered in this chapter includes:

on-line glossary entries (abridged version of the **Command Reference** Chapter)
alphabetical lookup capability
reminders when parameters are missing
reassignable function-keys, assigned to the most common commands
help for the mode panel options
help by category

1. Command Reference

The on-line version of the command reference includes the definition of the UniLab commands and features. Type in

HELP <command>

to get the information on your screen. If the entry takes up more than one screen, you will be prompted to press the PgDn (page down) key:

PgDn for more.

To use the **HELP** feature, you must have the DOS variable **GLOSSARY** set properly, as is explained in the installation chapter.

The on-line glossary contains the same information that appears in the printed manual. It is formatted slightly differently, and is updated regularly.

Command Reference Example

HELP BYE

```
 BYE              no parameters
Exits from UniLab program.
      USAGE
          To return to DOS.  Use SAVE-SYS first, if you want to
          save the current state of the system.
          Use DOS instead if you want to execute just a few DOS
          commands and then return to the UniLab program.
 ok
```

HELP MFILL

```
 MFILL           <from addr> <to addr> <byte>  MFILL
Fills every location in an area of memory with the same byte.
      USAGE
          A good way to check that memory address and data lines
          connect properly on the target board.  Use in
          combination with MDUMP.
          One way to find out what is happening on your board
          when LTARG test program will not run:  fill a block of
          memory with NOOP instructions, starting at the reset
          address, and then use STARTUP.   You should see a trace
          of consecutive addresses.
     EXAMPLES
     1200 1300 20 MFILL
          fills locations 1200-1300 with the value 20 hex.
```

2. Alphabetical Lookup

If you forget the full name of a command, you can look up the names of all the commands that start with a particular character. Type in

WORDS <character>

to get a list of all the commands that start with that character.

Or use

WORDS <command>

to get a list of commands, starting from that command. The list shows the first line of each command reference entry, which tells you what parameters the command requires (type **HELP** <command> to see the full entry).

Note the **F8** that appears to the right on some of the entries-- this indicates that the command is also a mode panel feature (press function key 8 to get the mode panel).

Some commands are assigned to other function keys. The name of the key will always be shown to the far right.

Alphabetical Lookup Example

WORDS N

```
NMIVEC       no parameters     F8
NMIVEC'      no parameters     F8
NORMB        no parameters
NORMM        no parameters
NORMT        no parameters
NOT          NOT  <trigger description>
NOW?         no parameters
ONLY         ONLY  <trigger description>
ORG          <address>  ORG
PAGE0        no parameters
PAGINATE     no parameters     F8
PAGINATE'    no parameters     F8
PCYCLES      <count>  PCYCLES
```

3. Reminders

If you forget what parameters a command requires, enter the command by itself to get a message describing the required inputs. For example, if you enter **MFILL**, you will get the following message:

"Requires the **First-address** the **Number-of-bytes** and a **Value**"

4. Function Keys

In menu mode, the function keys **F1** through **F10**, are assigned to menu choices.

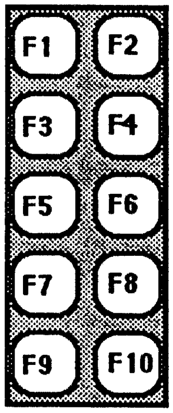
When you enter the command mode, the function keys are automatically reassigned to some of the most common UniLab commands. This allows you to execute with a single key-stroke any command that does not have parameters.

Altogether, you can have forty features assigned to the function keys. Each function key can be assigned four commands:

> you get one function by pressing a function key by itself,
>
> a second by pressing the function key while holding down the **ALT** key,
>
> a third by pressing the function while holding down the **SHIFT** key,
>
> and a fourth by pressing the function key while holding down the **CTRL** key.

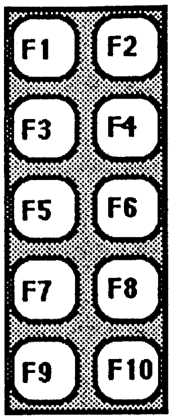
Turn to the next page for a chart of function key assignments.

Function Key assignments when no other key held down



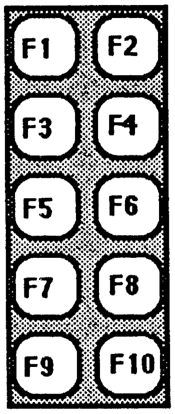
HELP with general instructions for using glossary. Also Function Key assignments.	F1	F2	SPLIT mode - Enter/Exit split screen mode.
Next Step - Execute next instruction. Will not follow jumps or branches.	F3	F4	NMI - Issue pulse on NMI- line to target, to gain DEBUG control or to single step through code.
Restore window split to Default sizes.	F5	F6	---
TSTAT - Display current trigger spec.	F7	F8	MODE - Bring up pop-up mode panels for changing display or system modes.
STARTUP - Issue reset pulse to target and trace first cycles of target operation.	F9	F10	MENU - Enter/Exit menu mode.

Function Key assignments when Ctrl key held down



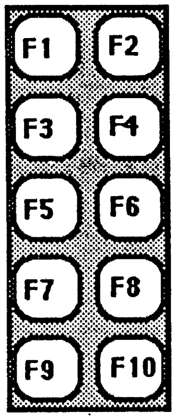
Help for using on-line displays	F1	F2	Help for using windows
Help for Debuggers	F3	F4	Help for simple analyzer triggers
Help for Emulation memory functions	F5	F6	More help for analyzer triggers
Help for loading/ saving programs	F7	F8	Help for mode panel switches
Help for displaying/ altering memory	F9	F10	Help for trace display

Function Key assignments when ⇧ key held down



List Function Key assignments for Shift	F1	F2	MEMO - Bring up system editor for use as custom memo pad
----	F3	F4	ASC - Show ASCII characters and hexadecimal code
----	F5	F6	----
----	F7	F8	WSIZE - Set new window split size
RES- - Pulls RES- output line low, and holds it low	F9	F10	----

Function Key assignments when Alt key held down



List Function Key assignments for Alt	F1	F2	----
----	F3	F4	----
----	F5	F6	----
----	F7	F8	----
SSAVE - Save the screen image as a text file	F9	F10	Call up the Program Performance Analyzer Menu

5. Help for the Mode Panels

You use the mode panels to toggle options on and off.

On-line help for the Mode Panels makes them as easy to understand as they are to use.

Press **F8** or type **MODE** to get into the mode panels.

While in a mode panel press **F1** to get the help display for the current option. These brief help displays appear in the following pages. Turn to the Mode Panel section of the Chapter Six if you need more detail.

The help display for each option includes the name of the command that the mode panel replaces. Type

HELP <command>

if you need more information on any particular feature.

Example: Panel One

```
 1. ANALYZER modes
DISASSEMBLER
SYMBOLS
RESET
```

Help with the DISASSEMBLER option of Mode Panel
This option toggles the processor-specific disassembler.
Turn off when examining most filtered traces.
The equivalent commands are: **DASM** **DASM'**

Help with the SYMBOLS option of Mode Panel
Toggles translation of numbers into symbolic names.
Define symbols with **IS** , or load from file with **SYMLOAD**
or **SYMFILE** . The equivalent commands are: **SYMB** **SYMB'**

Help with the RESET option of Mode Panel
When enabled, the processor is reset whenever the
analyzer starts up. Turn off to catch trace of program
in progress. The equivalent commands are: **RESET** **RESET'**

Example: Panel Two

2. DISPLAY modes
MISC COLUMN
CONT COLUMN
MISC # BASE
PAGINATE
FIXED HEADER

Help with the <u>MISC COLUMN</u> option of Mode Panel
When enabled, shows the MISCellaneous inputs to the UniLab (wires M0 through M7) on the trace display. The equivalent commands are: **SHOWM** **SHOWM'**

Help with the <u>CONT COLUMN</u> option of Mode Panel
When enabled, shows on the trace display the CONTrol inputs (C4 to C7), along with the high four bits of the address (A16 to A19). The commands are: **SHOWC** **SHOWC'**

Help with the <u>MISC #BASE</u> option of Mode Panel
Changes the base in which the MISCellaneous inputs are displayed. Toggles between binary and octal. The equivalent command is: <base> **=MBASE**

Help with the <u>PAGINATE</u> option of Mode Panel
When enabled, stops the trace display when screen fills. Disable only when you want to log entire trace to a file or a printer. The commands are: **PAGINATE PAGINATE'**

Help with the <u>FIXED HEADER</u> option of Mode Panel
Labels the columns of the trace display with a fixed header, rather than one that scrolls up with the display. Lower window only. The equivalent commands are: **HDG** **HDG'**

Example: Panel Three

```
 3. LOG modes
LOG TO PRINT
LOG TO FILE
PRINTER
NMI VECTOR
SWI VECTOR
```

Help with the LOG TO PRINT option of Mode Panel
When enabled, logs on the printer any commands that alter memory, such as **M!** and **MM!** . See also **PRINTER** option. The commands are: **LOG LOG'**

Help with the LOG TO FILE option of Mode Panel
Starts logging all screen output to the logfile. Create the file with **TOFILE** <name>, which can appear on the DOS command line. The commands are: **TOFILE TOFILE'**

Help with the PRINTER option of Mode Panel
When enabled, logs all screen output to the printer. The commands are: **PRINT PRINT'**

Help with the NMI VECTOR option of Mode Panel
When disabled, turns off the UniLab software's use of the hardware interrupt feature of your microprocessor. Disable if your target board needs to use that feature, or to have nearly transparent emulation. **NMIVEC NMIVEC'**

Help with the SWI VECTOR option of Mode Panel
When disabled, turns off all the DEBUG features of the UniLab software, such as **RB** and **N** . Turn off for completely transparent emulation. The commands: **RSP RSP'**

6. Help Screens: By Category

The command **HELP** by itself (or press **F1**) gives you general information on commands that give you help with the UniLab software.

If you press a function key while holding down the **CTRL** key, the UniLab program will display a help message for one of the categories of commands. Use **CTRL-F1** to display the selection of help screens available

These help screens appear on the following pages.

Example: General Help F1

```
HELP is available on-line by entering HELP or F1.
Enter  HELP command  to see the definition of "command"
Type  WORDS command  to see a list of commands.

Use the function key F10 for MENU mode operation and quick
  access to most common commands.
More help is available on the Ctrl-F1 to Ctrl-F10 Keys.
Press Ctrl-F2 for display of cursor key functions.

Type MESSAGE for current messages.
F1=HELP F2=SPLIT F3=N F4=NMI F5=DEF F6=NOP F7=TSTAT F8=MODE F9=STARTUP F10=MENU
```

Example: Using On-Line Help **CTRL-F1**

Help By Category

Hold down CTRL and tap one of the function keys to get a few hints on using the UniLab.

Help On:

Using On-line Help	C1	C2	Windows
DEBUG Commands	C3	C4	Simple Triggers
Enabling Memory	C5	C6	More on Triggers
Load/Save Programs	C7	C8	Mode Panels
See/Alter Memory	C9	C10	Trace Display

Type **HELP** <command> for more information about any command. **HELP BYE,** for example, will give you information about the command that you use to exit from the UniLab program.

Example: Windows **CTRL-F2**

<u>**Help for Windows**</u>

Windows make it easy for you to organize information on your screen. Once you split the screen, you can show different parts of a trace in the upper and lower windows, compare a trace to the disassembled program, examine source files, etc.

SPLIT or **F2** to enter split window mode.
SHIFT-F8 to change window size w/cursor keys.
END key to move from one window to another.

Other window commands: **DN** **TEXTFILE.**

Example: DEBUG Commands **CTRL-F3**

Note that this help screen includes information on the processor specific DEBUG that you are using.

<u>**Help for The DEBUG**</u>

You use the DEBUG commands to look at the internal state of your microprocessor, single-step through your program, and examine or change target board RAM. But first you have to <u>**Establish Debug Control**</u> with **NMI** or by setting a breakpoint with **RESET <address> RB.**
Once you have established debug control, you can resume program execution with a breakpoint set at another address with **<address> RB.** You single step through a program starting at a breakpoint, with **F3** or **N,** if you don't want to see execution of jumps and branches. Use **NMI** to follow jumps and branches.
All commands for reading and changing memory work on RAM when stopped at a breakpoint. <u>**CTRL-F9**</u> tells you more about memory.

[PROCESSOR SPECIFIC INFORMATION FOR Z80]

m n OUT writes m to port n. **n INP** reads port n.
EINT re-enables target IRQ'S after bp, **DINT** leaves disabled
reg change: **n =AF** **n =BC** **n =DE** **n =HL** **n =IX** **n =IY**
locations **38-3D** reserved, overlay starts at **3D**

Example: Simple Triggers **CTRL-F4**

Simple Trigger Commands

You use the trigger commands to describe bus conditions. When the UniLab's bus state analyzer sees the event you described, it will "trigger" and capture a record of the bus activity. The simplest trigger searches for an address on the bus and freezes the trace buffer five bus cycles after finding the address: **NORMT** <address> **ADR S**

The **NORM**x words clear out previous trigger specs. **ADR** tells the analyzer that you want it to monitor the address lines.
S starts the analyzer.
S+ shows you bus activity starting after the end of the current trace buffer.
Type **HELP** <command> for more information on these and other trigger words: **NORMM** **NORMB** **DATA** **CONT** **MISC**
CTRL-F6 gives you more hints about triggers.

Example: Enabling Memory **CTRL-F5**

Help for Enabling Memory

Before you load a program into the UniLab's emulation memory you must first enable the memory.
You specify the upper four bits of the address with
<hex digit> **=EMSEG**
and then specify the remaining 16 bits of the address with
<value> **TO** <value> **EMENABLE.**
To see the current status of memory, use **ESTAT.**

Example: More on Triggers **CTRL-F6**

More on Triggers: Filters, Qualifiers and Reset

You can fill a trace buffer with only the bus cycles that match a description (filtering), specify pre-conditions for trigger, (qualifiers), or turn reset of your target board on or off.
Precede a trigger spec with **ONLY** to get a filtered trace.
See also **1AFTER 2AFTER 3AFTER.** To avoid confusion, turn off your disassembler while reading a filtered trace.
Precede a trigger spec with **AFTER** to make the condition described by the spec a precondition for trigger.
See also **PCYCLES PEVENTS.**
RESET enables resetting of your target board-- your program starts over whenever you start the analyzer with **S** or **S+.** If you want to capture a trace of a program in progress, disable resetting with Mode Panel **F8** or with **RESET'.**

Example: Load/Save Programs **CTRL-F7**

Help for Loading and Saving Programs

Load programs from ROM with the ROM reader Menu: **F10** then **F9.**
Load from disk files with **HEXLOAD** <file name> for Intel Hex format files, or <from addr> <to addr> **BINLOAD** <file name> for binary files.
Load from host RAM with <from srce> <to srce> <target> **MLOADN.**
Save a program to disk with
<from addr> <to addr> **BINSAVE** <file name> .

Example: Mode Panels **CTRL-F8**

Help for Mode Panels

The mode panels, entered with **F8** and left with **END**, allow you to change display options, save information to the printer or a file, turn off the DEBUG, etc. **F8** also moves you from one mode panel to the next.
To get more information about any of the options of the display panel, press **F1** while in the mode panel. Also try **HELP MODE.**

Example: See/Alter Memory **CTRL-F9**

Help for Examining and Altering Memory

Unless you have debug control (press **CTRL-F3** for more on that) you can only operate on emulation ROM.

<address> <count> **DM**	disassembles from memory.
<from address> <to address> **MDUMP**	dumps a section of memory.
<byte> <address> **M!**	stores a byte of data.

Use **HELP** <command> for info on: **M? MM! MM? ORG MFILL MMOVE.**

Example: Trace Display **CTRL-F10**

Reading through your Trace Display

HOME shows you the trace display starting from the top
PgDn shows you the next page, while **Down arrow** shows one more line.
<n> **TN** shows the trace starting from step n .
Note that **PgUp** and **Up Arrow** show you history , not trace display.

Index for Volume One

The full index can be found at the end of Volume II.