ORION Instruments

UniLab II™

Volume One
User's Guide

NOTE SOME OF OUR NEW FEATURES:

Extensive On-Line Help

Press function key 1 **(F1)** to get the opening screen help message.

While holding down **CTRL** press any of the function keys, **F2** through **F10**, to get one of the help screens. **CTRL-F1** tells you what subjects you can get help on.

To get the on-line glossary entry for a command, type **HELP** ‹command›.

To get an alphabetical listing of commands starting with some character or characters, type **WORDS** ‹character(s)›.

Windows

While in command mode, hit **F2** to split the screen.

The on-line glossary entries for **SPLIT**, **DN** and **MODE** will tell you more.

IN CASE OF TROUBLE

If your **UniLab** is not properly connected to serial port 1, then the program will stop while displaying: **Initializing UniLab...** Hold down the **CTRL** and **BREAK** keys at the same time to unfreeze the program.

After you have connected the instrument to the computer, type **INIT** to initialize the **UniLab**.

If you have a different problem, consult the Trouble-Shooting chapter.

ORION
Instruments
702 Marshall Street
Redwood City, CA 94063
(415) 361-8883

UniLab II™

Universal Development Laboratory

SOFTWARE INSTALLATION

Your UniLab software comes on two diskettes.

One diskette contains the executable files and overlays for your **UniLab**. Any disassemblers or debuggers you ordered are already installed in one of the **.COM** files on the disk. For example, if you ordered a **Z80** support package, there will be a file called **ULZ80.COM** on your diskette.

The second diskette contains the on-line glossary and its supporting files. If you have a hard disk, you will want to copy these files into your C:\ORION directory. If you have a floppy, you will need to have this second diskette in drive B: when you use either **HELP** or **WORDS**.

The instructions in this booklet tell you how to get your software ready to use. To find out how to connect the **UniLab** to your target board, refer to the **Installation** chapter of the manual. For information on how to use the program, read the **Guided Demo** and **Getting Started** chapters.

HOW TO INSTALL YOUR NEW SOFTWARE:

ON A HARD (WINCHESTER) DISK

1) Put the master **UniLab** diskette into drive A. Call up the UniLab hard disk install program by typing **A:INSTALL**.

The program will create a directory on your **C** drive called **\ORION**, and copy the UniLab files from the distribution diskette to that directory.

The install program adds two lines to the **AUTOEXEC.BAT** file in your root directory:

```
SET ORION=C:\ORION
SET GLOSSARY=C:\ORION
```

2) You must either create or change your **CONFIG.SYS** file, in your root directory. This file sets system variables, **files** and **buffers**. A sample CONFIG.SYS file is included on the distribution diskette.

Our software requires 16 files for full functionality:

```
files=16
```

If you don't put this line into CONFIG.SYS, some UniLab features will not work.

The UniLab software will perform better if you allocate at least 10 buffers:

```
buffers=10
```

but this is not necessary.

Now go to step 3 on the next page.

HOW TO INSTALL YOUR NEW SOFTWARE:

ON FLOPPY DISK SYSTEMS

1) Put your DOS diskette in drive A, and a blank, unformatted floppy in drive B. Use the DOS command **FORMAT B: /S** to format the blank disk as a "bootable" system diskette.

2) Put the master **UniLab** diskette into drive A. Use the DOS command **COPY A:*.* B:** to copy all the UniLab files onto the bootable diskette in drive B.

The distribution diskette includes the **CONFIG.SYS** and **AUTOEXEC.BAT** files necessary for your floppy disk system.

Put the new bootable diskette into drive A, and go on to step 3.

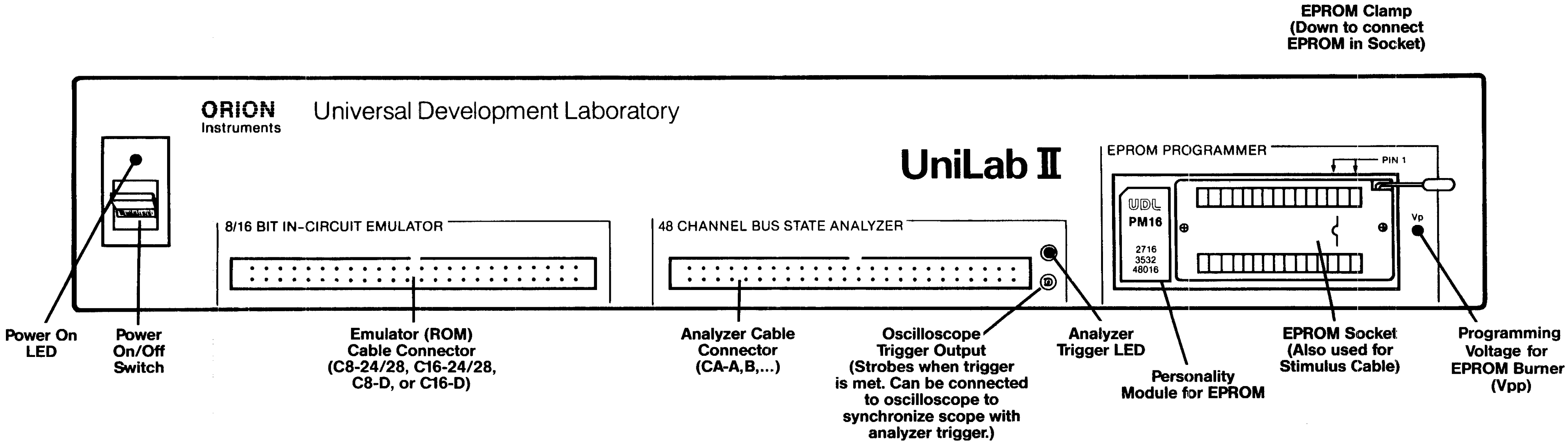
ON BOTH HARD AND FLOPPY

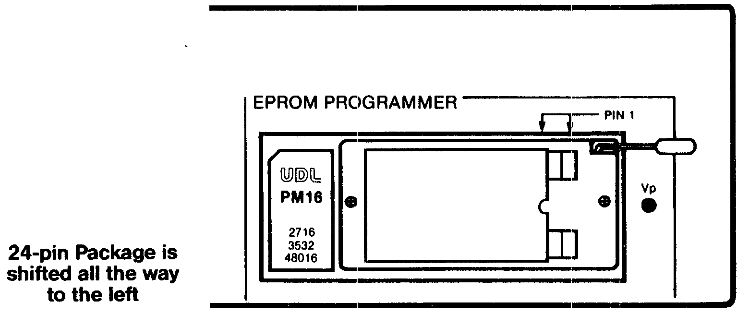
3) Put the distribution diskette away. Now reboot your computer system so that the new settings in the CONFIG.SYS and AUTOEXEC.BAT files can take effect. Reboot by turning the power off and then back on, or by holding down the **CTRL**, **ALT** and **DEL** keys at the same time.

4) After you have rebooted, execute the UniLab software by typing in the name of the **UL**xxx**.COM** file you were sent. For example, the command for the 8096 package is **UL96**. On hard disk systems you must type **\ORION\UL**xxx.

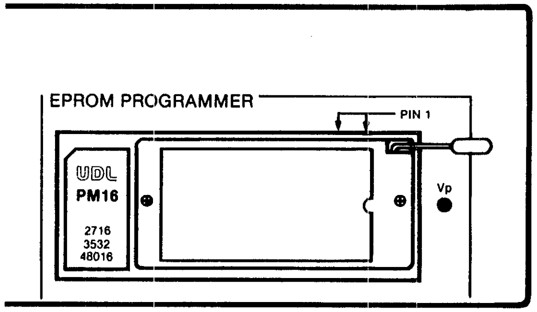
The software will take a few seconds to load, then print an opening screen and initialize the UniLab. See the next page if your software freezes during this process.

After that, you are ready for action. Hit **F10** to enter the fast new menu mode. Type in **BYE** when you want to exit from the program.





24 Pin EPROM in Programming Socket



28 Pin EPROM in Programming Socket

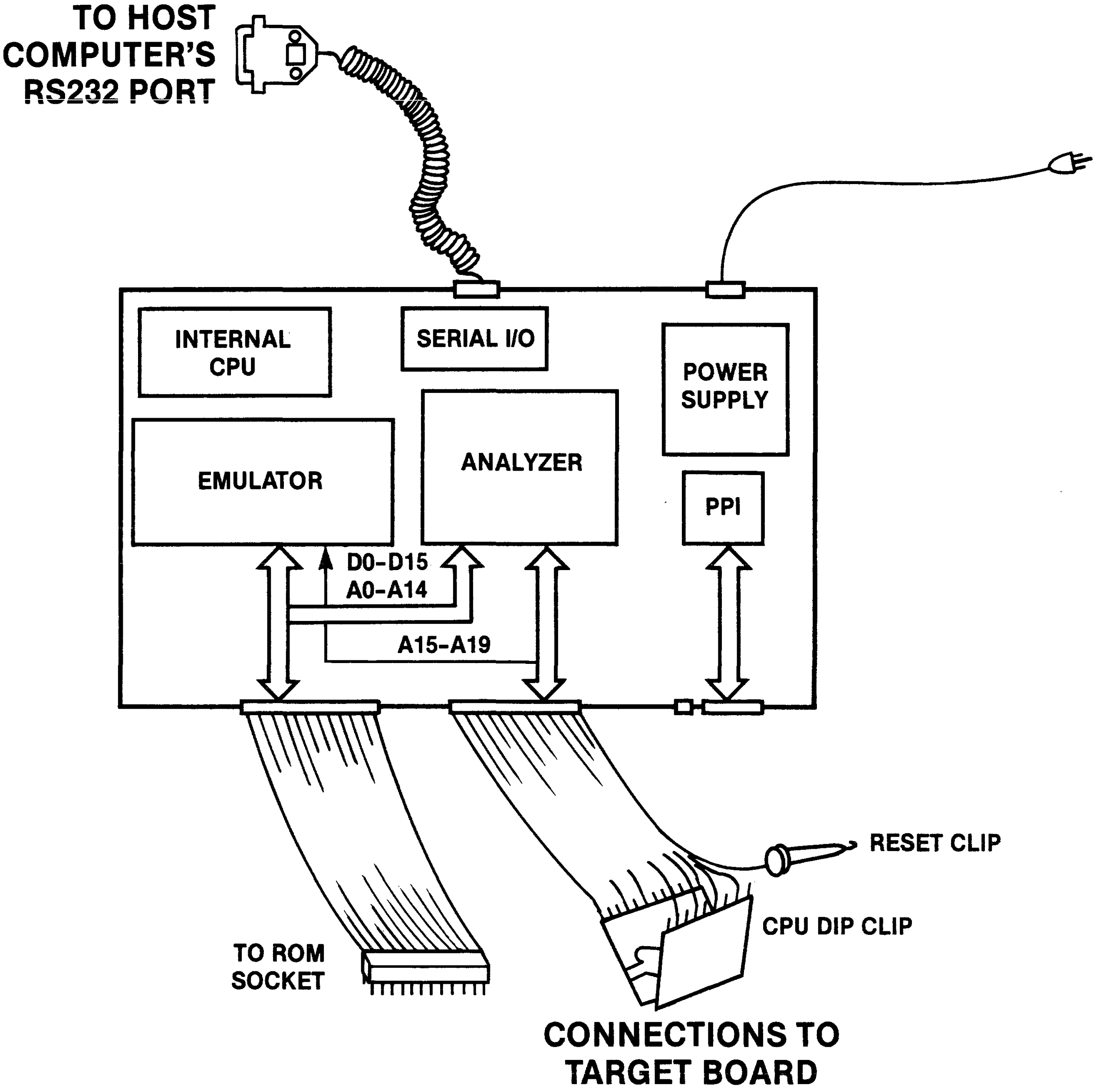

TO HOST COMPUTER'S RS232 PORT
INTERNAL CPU
SERIAL I/O
POWER SUPPLY
EMULATOR
ANALYZER
PPI
D0-D15
A0-A14
A15-A19
RESET CLIP
CPU DIP CLIP
TO ROM SOCKET
CONNECTIONS TO TARGET BOARD

Table of Contents
UniLab Manual

VOLUME ONE
User's Guide

Chapter Three: Guided Demonstration

Chapter Four: Getting Started-- The Menus, the Commands, and the Special Features

VOLUME TWO
Reference Manual

Chapter Six: The UniLab in Detail

Chapter Seven: UniLab Command Reference

Chapter Eight: Target Notes

Chapter Nine: TroubleShooting

APPENDICES:

Chapter One: The UniLab Method

Introduction

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 board. This lets you see how the board 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 Does the UniLab Do It?

The UniLab connects to your board's address, data, and control lines. Your microprocessor stays in control of the board-- but the UniLab is in control of your processor.

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.

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 debugging 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 act like older debugging systems, and stop your processor by setting a breakpoint.

A New Way to Solve Your Problems

The UniLab is a step ahead of traditional debugging techniques.

Older methods work best when you know the cause of the problem at the beginning of the debugging procedure. 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 can describe the symptom of your bug. Then you watch what the program does both before and after the symptom occurs. The symptom that you describe is called the "trigger." For example, you might start out by asking the UniLab to show you when your stack has grown too much, or you might want to see each re-write of a particular variable.

You start by asking general questions, and quickly zero in as your understanding of the problem improves.

If you are in the habit of single-stepping, or setting multiple breakpoints, you will appreciate the new, more powerful method of observing program flow with the UniLab.

Breakpoints and Single-stepping

The UniLab's unique approach to debugging 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 debuggers let you set breakpoints, look at and alter internal registers, and single-step through code.

In order 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."

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.

Chapter Two:
Installing The UniLab

Contents

For Me?

All the discussions that follow assume that your host computer is a PC compatible machine, running MS-DOS version 2.0 or higher. Unless you have this equipment, you will not be able to run version 3.0 and higher of the UniLab software.

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 you probably need a software up-grade.

Introduction

Controls

The UniLab instrument has almost no controls of its own. You control it through a program running on your personal computer.

The only important control is the ON/OFF switch. When you turn on the UniLab, the light above the switch goes on. Then the Unilab is ready to accept commands from the host computer.

File Names

All of your interactions with the UniLab will be done through a program called **ULxx.COM**, which runs under the host operating system.

Disassemblers and debuggers are shipped already installed as part of this file. To distinguish between different disassembler/debuggers, each **.COM** file has a descriptive name such as **UL48**, **UL88**, etc.

Other files included on your distribution diskette supply the help screens and other features.

Get Started Quickly. . . .

You might be impatient to get the UniLab hooked up and running.

If you want to jump right in, the **Quick Step by Step** section following this introduction contains stripped down instructions--and not much explanation.

Or Learn as You Get Started

You might have the time and desire to get more familiar with the UniLab while you set it up.

To learn more about the instrument while you get it ready, you can use the **Detailed Step by Step,** which contains more thorough instructions and longer explanations.

The Best of Both Worlds

You might want more detail on only one step of the installation process.

While you are following the quick set-up procedure, you can read more about any topic by dipping into the detailed procedure. The 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 it out.

Test Procedure

After the first three steps, you will want to run the simple test program that is included with your software. You should compare the results you get with the printout of the standard results, which appear both in the **Target Notes** chapter of the Manual and in the Disassembler/Debugger writeup for your microprocessor.

If there are no differences, then you can be certain that you have connected everything properly, and get on to your work with confidence.

Automated Test Procedure

A UniLab command, <#_of_cycles_to_compare> **TCOMP** <filename>, compares the current trace with the trace stored in a file. Turn to page 2-42 for more details.

Most of the disassembler packages include a file containing a known good trace (consult Appendix J). That trace is contained in a file with the suffix **.TRC** on your distribution diskette. If your package does not include a trace, you can visually compare the trace you get to the trace in your Disassembler/Debugger writeup.

16-bit Systems

If your target system has a 16-bit data bus, you follow the same installation procedure described in this chapter, except that you must use a 16-bit emulator cable. 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.

Whether the odd or even byte is the most significant byte of the 16-bit word depends upon whether your processor follows the Intel or Motorola model.

Where to Go After You Have Everything Hooked Up

Some people prefer jumping into a task, and figuring out what they need to know on the fly. The Menu system lets you learn the commands as you go along.

Other people prefer to get more background before they start working with a program. The next chapter, **Guided Demo,** takes you through the process of loading a program into emulation memory and using the UniLab to debug it. You should follow this demo at your computer, but be aware that what you see on the screen will be different if you have any processor besides the Z80.

The **Getting Started** chapter gives you an overview of the capabilities of the UniLab. And you will find that the **In Detail** chapter lives up to its name, thoroughly covering the use of the UniLab, including the optional Graphical Performance Measurement feature (**AHIST** and **THIST**).

For your convenience the on-line help screens appear in the **On-Line Help** chapter. The **Command Reference** chapter gives you a definition, explanation and example for every command.

The separate writeup on your Disassembler/Debugger package provides information on the specifics of using the UniLab with your microprocessor.

If You Have Trouble

The **TroubleShooting** chapter starts with a list of symptoms. The SOLUTIONS IN DEPTH section of that chapter then helps you solve your problems.

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

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.

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 demo/training package. 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, because that is 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.

After the UniLab gets an instruction, it 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.

UniLab Inputs

The bus state analyzer 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-**.

UniLab Outputs

The emulator looks at the twenty bit address and the read signal, and responds with eight- or 16-bit data when appropriate.

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

On most processors-- those that have a non-maskable hardware interrupt-- a Non-Maskable Interrupt signal is sent out on the wire labeled **NMI-**. This wire sometimes requires special handling.

Quick Step-by-Step

1. **CONNECT THE UniLab TO HOST** (page 2-11)
   Connect the UniLab to your host computer, and turn it on.

2. **SOFTWARE INSTALLATION** (page 2-14)
   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.

   Reboot your system and then start up the UniLab program. Check Appendix H to determine whether you need a patch word.

3. **CONNECT THE UniLab TO TARGET BOARD** (page 2-25)
   You should 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 the ROM from its socket on your board, and put the ROM plug of the emulator cable in its place. Connect the other end of the cable to the emulator socket.

   Clip the DIP clip onto your microprocessor and connect the wires from the emulator cable and the analyzer cable to the proper pins. See **Target Notes** chapter or use the **PINOUT** command. Connect the other end of the analyzer cable to the analyzer socket.

   Connect the **RES-** wire to the proper place in the reset circuit. See pages 2-33 to 2-35 for details.

   Turn on the power supply to your target board.

4. **CHECK OUT YOUR EQUIPMENT** (page 2-40)
   From within the UniLab program, press function key 10 (**F10**) to enter the menu mode. Use **F2** to select the "LOAD OR SAVE A PROGRAM" sub-menu, and select the "LOAD A SAMPLE PROGRAM" option by hitting **F4**.
   F10 will then return you to the Main Menu, where you can use **F4** to select the "USE THE ANALYZER" sub-menu. In that menu, use **F1** to select "RESET AND TRACE FIRST CYCLES."

   Now go to page 3-1 to learn more about the UniLab.
   When you want to exit, just type in the command **BYE.**

Detailed Step by Step

You might want more information than you can find in the very sparse **Quick Step-by-Step.** You will find detailed assistance in the next pages. The numbered headings follow the outline of the **Quick** section, but here you get at least one page, usually several pages, for each heading rather than only a few sentences.

Each numbered task has been broken down into several subtasks. Some troubleshooting help appears in each section, but the main help comes in the separate **TroubleShooting** chapter.

1. **CONNECT THE UniLab TO HOST**
FIND THE CORRECT PORT
SERIAL PORT OF AT
CONNECT THE CABLE
TURN ON THE UniLab
TROUBLE?

First connect UniLab, THEN start up software

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.

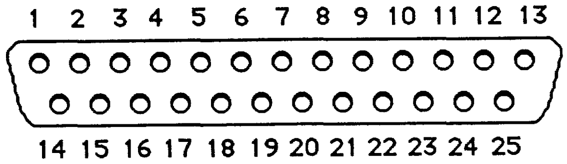
When you do start up your software, the UniLab must be properly connected and turned on-- otherwise the software will freeze up (hit **CTRL-BREAK** to unfreeze). So we recommend that you first connect the UniLab, then install your software.

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.

If you do not have a serial port on your computer, then you will need to purchase and install a serial port board before you can connect to the UniLab.

DB-25 Connector



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.

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).

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.

The easiest 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 DRIVE

REBOOT YOUR COMPUTER
START UP THE UniLab PROGRAM
PATCH REQUIRED?

INSTALL THE SOFTWARE

If you have a hard disk on your computer, you will want to use the INSTALL.BAT file to install the UniLab software on your hard disk.

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

It 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 file.

You will need to create or alter your CONFIG.SYS file so that DOS allows the UniLab software to have 16 files open and use 10 buffers. The CONFIG.SYS file resides in your root directory **(C:\)**. It should contain these two lines:

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

If these system variables are already set to higher values, that is fine. However, some of the UniLab features will not work if "files" is set too low. If "buffers" is smaller than 10, then the UniLab software will run slower.

The new settings will not take effect until you reboot your computer.

ON A HARD DISK (continued)

Procedure

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

A:INSTALL

The INSTALL program will copy all of your software into a directory called ORION, and either create or alter the AUTOEXEC.BAT file in your root directory.

In fact, INSTALL does everything for you except create or alter the CONFIG.SYS file in your root directory (**C:\**). You will find a sample CONFIG.SYS file on the distribution diskette.

You will also need to copy the files from the second diskette to your **C:\ORION** directory. Unless you copy to the hard disk the glossary file and its supporting files, the commands **WORDS** and **HELP** <command> will not work.

ON A FLOPPY DISK DRIVE

Explanation

You need to set up two "environment strings" and two system variables before you can use all the features of the UniLab software.

Make a new "bootable diskette" and always use it to boot up your system, so that those environment characteristics are always set to the values that your new software requires. You can either copy the entire UniLab system onto the bootable diskette, or just the CONFIG.SYS and AUTOEXEC.BAT files that are included on the distribution diskette.

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.

Of course, if you wanted to run the UniLab software from drive B you would have to change this to:

```
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
```

If these system variables are already set to higher values, that is fine. However, some of the UniLab features will not work if "files" is set too low. If "buffers" is smaller than 10, then the UniLab software will run slower.

The new settings will not take effect until you reboot your computer.

The settings tell the program to look on drive B: for the glossary files. That means you must have the glossary diskette in drive B: in order to use **HELP** <command> or **WORDS** <character>.

ON A FLOPPY DRIVE (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. To copy all the UniLab files to the newly formatted diskette, use the DOS command:

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

Instead, you might decide to put only the AUTOEXEC.BAT and CONFIG.SYS files on the newly formatted diskette.

Just remember that each time you use your UniLab software, you must be sure that you booted up the computer from a diskette with the correct CONFIG.SYS file on it.

REBOOT YOUR COMPUTER

Explanation

The new settings in the CONFIG.SYS file will not take effect until you reboot your computer-- and the settings of the system constants "files" and "buffers" can only 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\UNI.** 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 new 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:**.

Rebooting Procedure

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** and **.VIR** files for the software to run properly, you only call one file by name-- the command file, which ends in **.COM.**

Use the DOS command **DIR** to get a listing of all your UniLab files. You will see one with **.COM** 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.

After it has displayed that screen, it sends an initialization message to the UniLab and waits for a reply from the UniLab. When it receives a reply the program 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 debugger.
         ______________________________________________________________
          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: Tyoe 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.

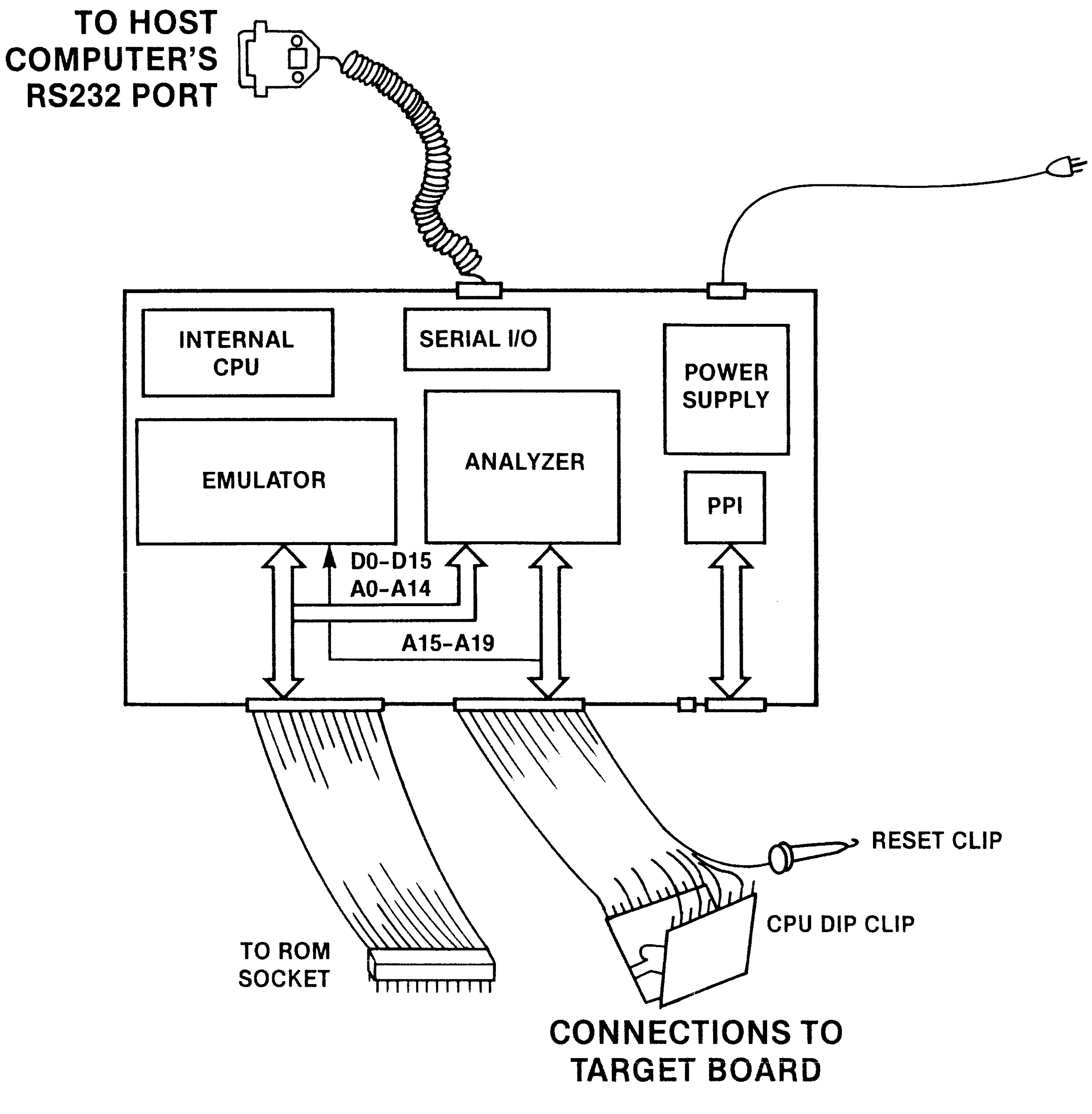
If you get an "**RS-232 Error #**xx" then you probably have a background task running, such as a printer spooler, or have a "bus contention problem" on the serial ports of your computer. Both problems can be quickly solved. See the **TroubleShooting** chapter.

PATCH REQUIRED?

Some Orion software packages support more than one processor, and require you to type in a "patch word," which tells the software which processor you will use the software with.

Appendix H tells you which processors require a patch word, and what the patch word is. You will also find that information in the Disassembler/Debugger notes for your software package.

After you enter the patch word, you should use **SAVE-SYS** to save the patched version of the software. That way, you only need to enter the patch word once. We recommend that you save the patched program to the same file that you used to call the program in the first place (that is, **UL48**, or **ULZ80**, or whatever).


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

3. CONNECT THE UniLab TO THE TARGET BOARD

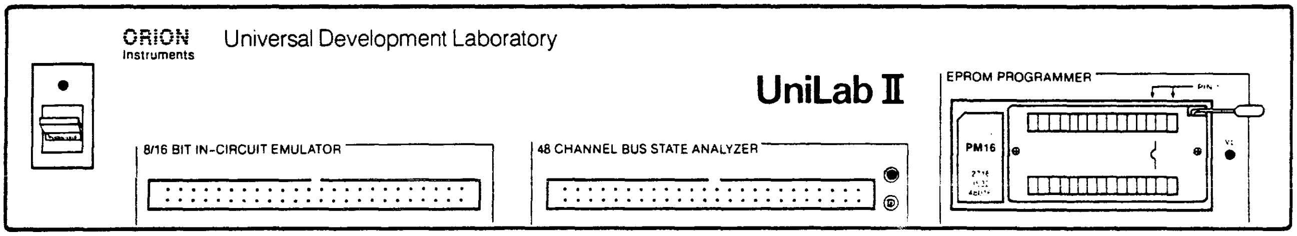
OVERVIEW
ALL ABOUT CABLES
TAKE PROM OFF BOARD
PUT ROM CABLE IN ROM SOCKET
PUT DIP CLIP ONTO MICROPROCESSOR
ATTACH PROPER WIRES TO THE CLIP
ATTACH THE RESET WIRE
ATTACH THE NMI WIRE
PLUG CABLES INTO UniLab CONNECTORS

OVERVIEW

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

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.


ORION
Instruments
Universal Development Laboratory
UniLab II
EPROM PROGRAMMER
8/16 BIT IN-CIRCUIT EMULATOR
48 CHANNEL BUS STATE ANALYZER
PM16

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 must match the letter on the pinout diagram in the **Target Notes** chapter.

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 special 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 be certain to 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. Here the UniLab picks up and asserts control signals. Sometimes the **NMI-** wire requires a special connection, similar to that for the **RES-** wire.

Reset Wire and the NMI Wire

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

Complications

These two outputs from the UniLab are "open collector" (RTL)-- which means the signal coming from the UniLab is not strong enough to pull down the output of a logic element (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.

TAKE PROM OFF BOARD

The UniLab emulates the Read Only Memory (ROM) of the target board. To avoid bus contention problems, you will need to take off your target board any PROM chips you plan to emulate.

With the power to the target system turned off, remove all the ROMs that you will be emulating.

Exception

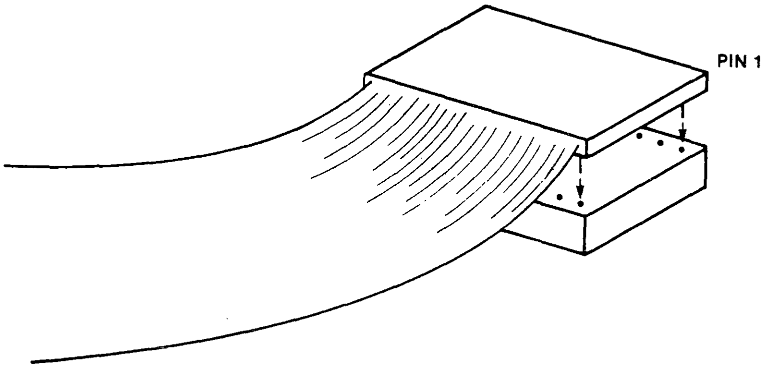
The only time you will want to keep the ROM chips on your target board is when you want to watch the execution of a program running from the chip. Before you can execute a program from a chip, make certain that you disable the emulation ROM with **EMCLR,** and disable the debugger functions with the Mode Panel option **SWI VECTOR** (mode panel 3) or with the command **RSP'.**

When you want to run the program in a ROM, you would probably be better off using the **PROM READER MENU.** You can read the program from the chip into the UniLab's emulation memory, and then run the program while it resides in emulation memory.

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.

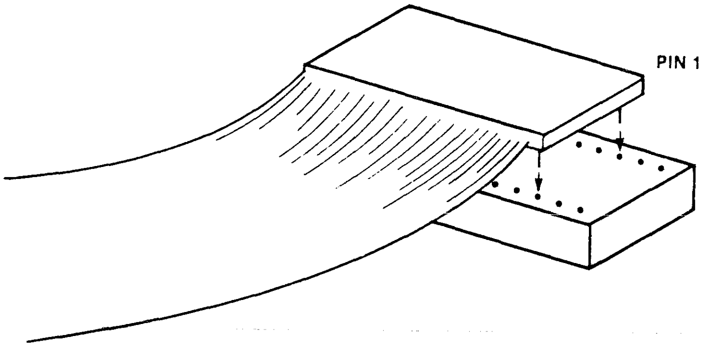


24 Pin ROM Plug
in 24 Pin Socket

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

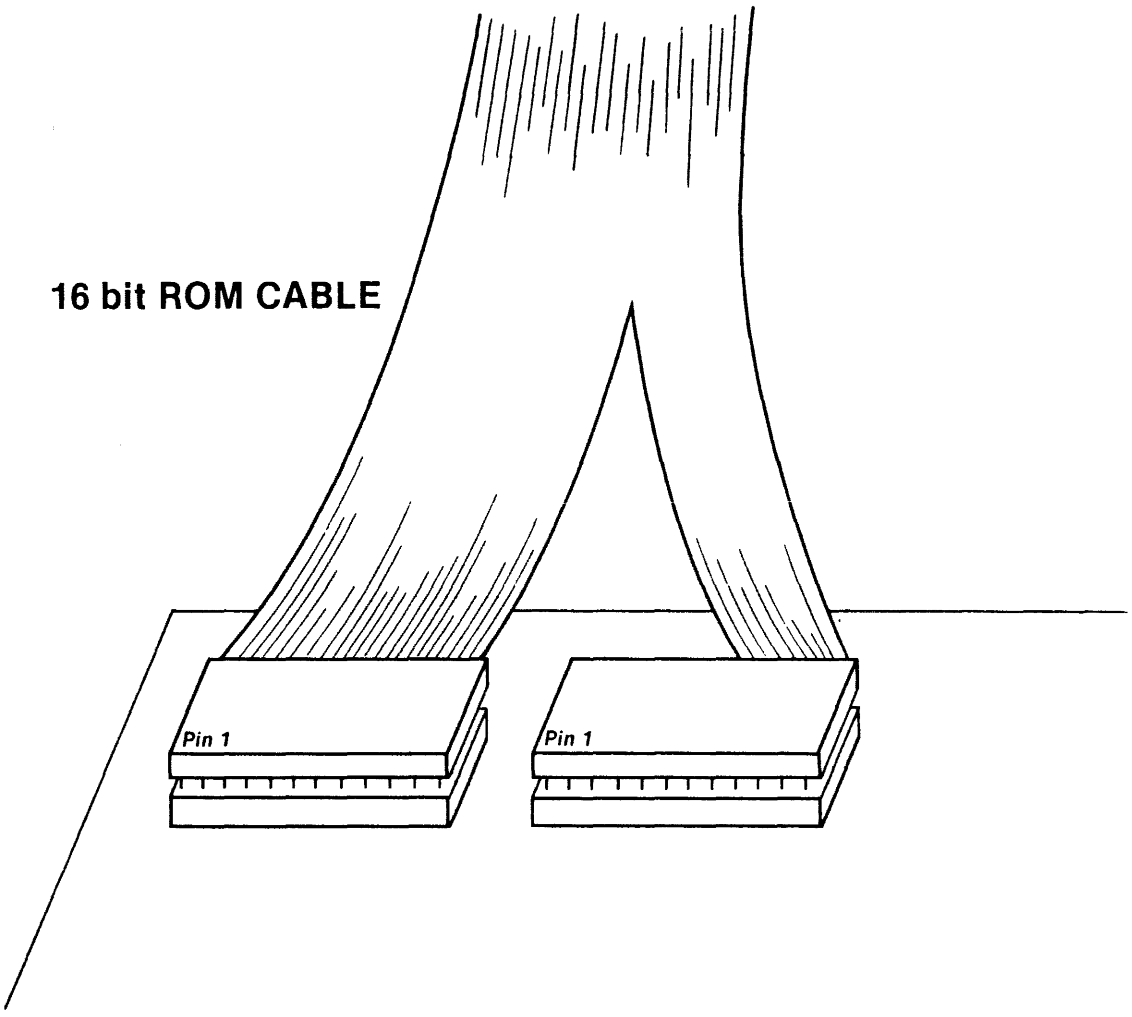
A 24-pin ROM plug can also go into a 28-pin socket, if pin one of the cable goes into pin three of the ROM it replaces. That will leave four unfilled positions on the socket, 1 & 2 and 27 & 28.



24 Pin ROM Plug
in 28 Pin Socket

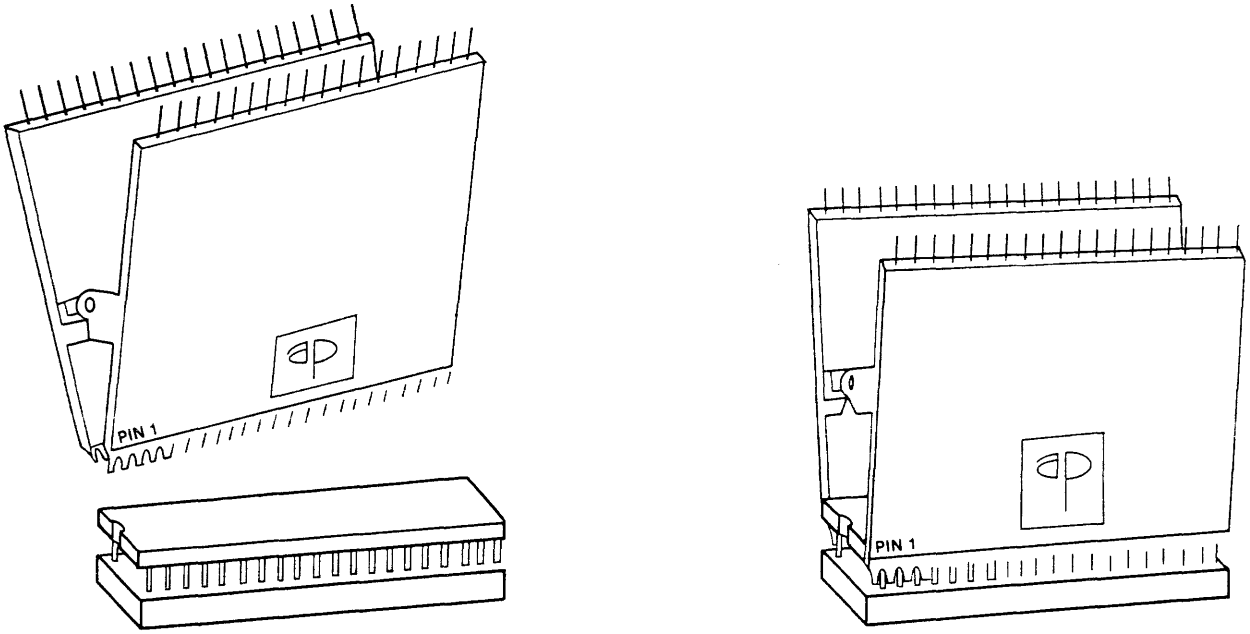
16-bit ROM Plugs

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.

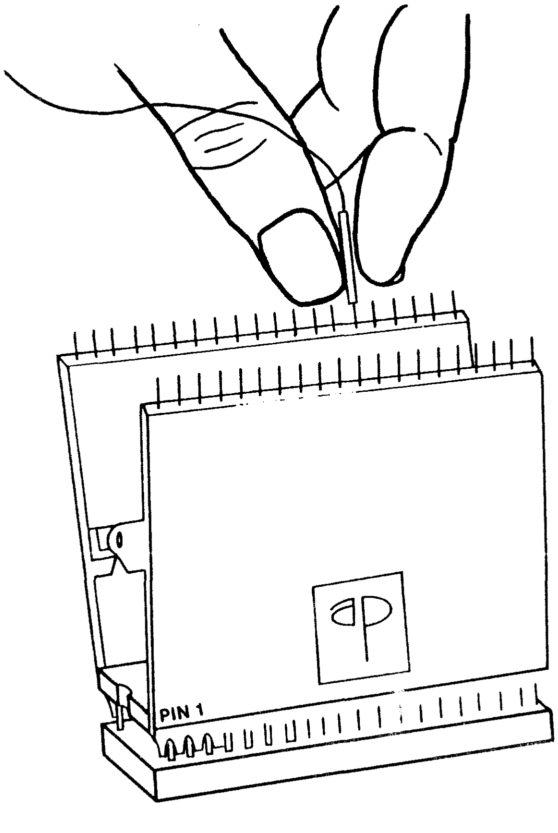


Placing DIP CLIP on Processor

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 section on your processor in the **Target Notes** chapter-- or type **PINOUT** to get an on-line diagram. The **Cable Wiring** appendix (Appendix C) contains a table of the connections for all processors.

Double-check your wiring of the cable, to be sure that it is correct.



Making Connections from Analyzer and Emulator Cables

ATTACH THE RESET WIRE

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

Connecting 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.

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.

An "open collector" 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 "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

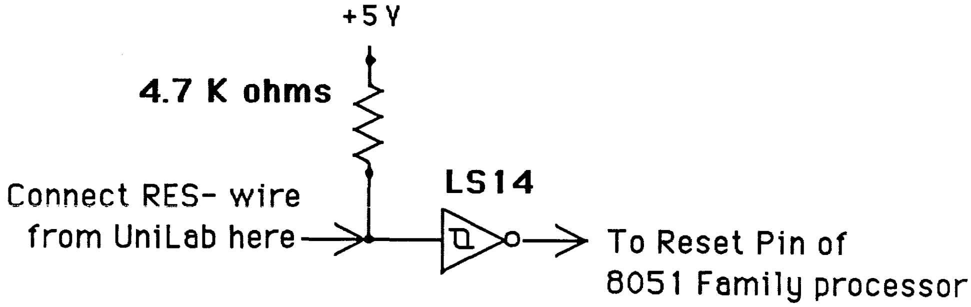
Sometimes the common solution will not work, and you will have to alter your reset circuit. You might have to remove the chip that drives the reset circuit, and connect a pullup resistor in its place.

8051 Family Complication

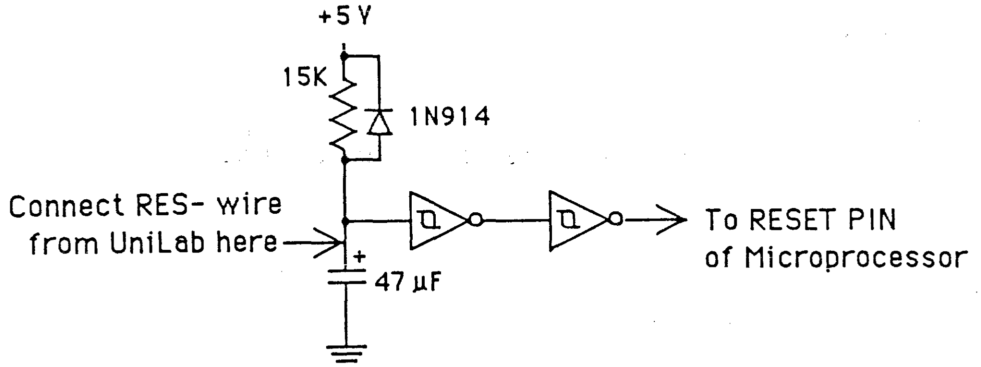
If your processor is in the **8051 family,** the RESET pin of your processor requires a positive going signal. You will have to feed the UniLab's negative going signal through a special inverting circuit, such as the one below.

The members of the **8051 family** are:

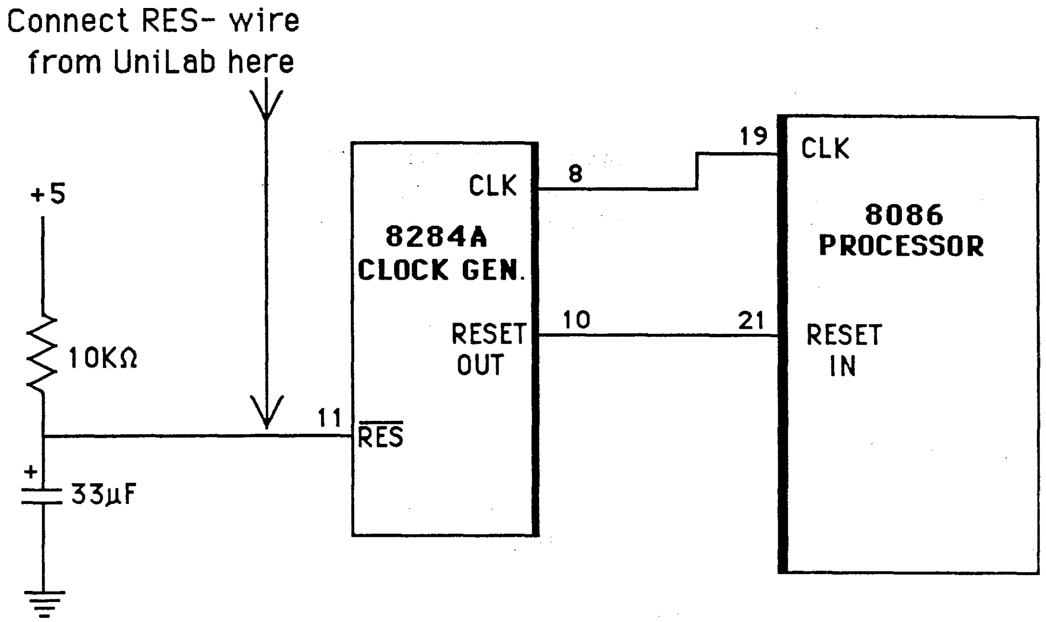
8051, **8052**, **8031**, and **8032.**



Typical Reset Circuit necessary for 8051 family processor



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 special interrupt location. The UniLab software uses this signal for three debugger features:

NMI **RI** and **SI** **SSTEP**

These commands are not supported by all processors. Consult Appendix H to find out whether your software package supports NMI.

If your processor does not support **NMI** then you do not need to have this wire connected. But see the note that appears on the next page about processors that don't support NMI.

Simple Pullup Circuits

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 pin.

Common Complication

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.

Hardware Solution

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.

Simply type in the command **NMIVEC'** to disable the UniLab's use of the NMI vector, and then use **SAVE-SYS** to save the newly configured software.

You can always enable this feature again with **NMIVEC.**

Processors that do not support NMI

On these processors you can make use of the **INT** command to produce a low-going transition on the **NMI-** wire when the UniLab goes into trigger search state.

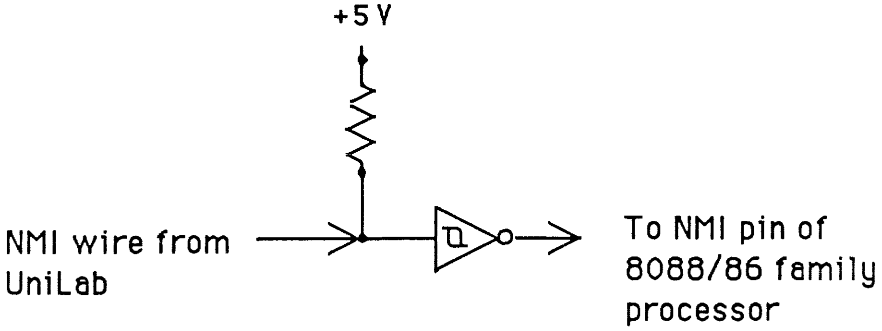
You can then connect the UniLab wire to the IRQ pin of your processor (along with a pullup resistor). This will cause your processor to execute a maskable interrupt. You will have to write your own interrupt routine. You would want to do this if you need to shut down peripheral equipment when some error condition occurs. See the entry on **INT** in the **Command Reference** chapter.

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.



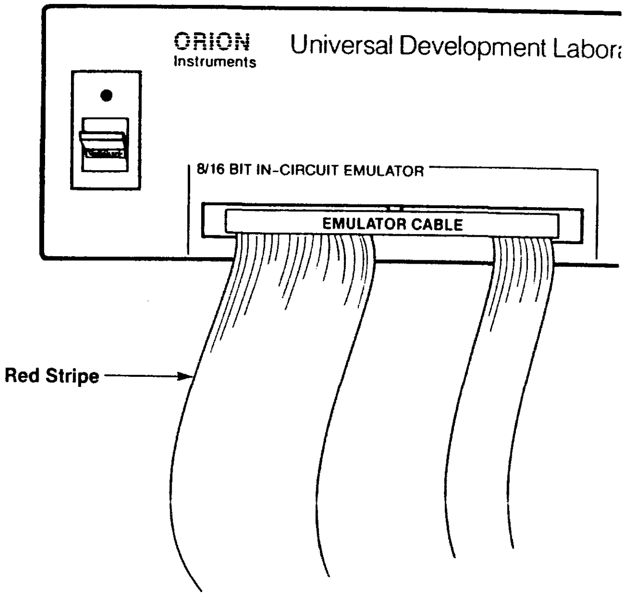
Typical NMI circuit needed for 8088/8086 family processors

]

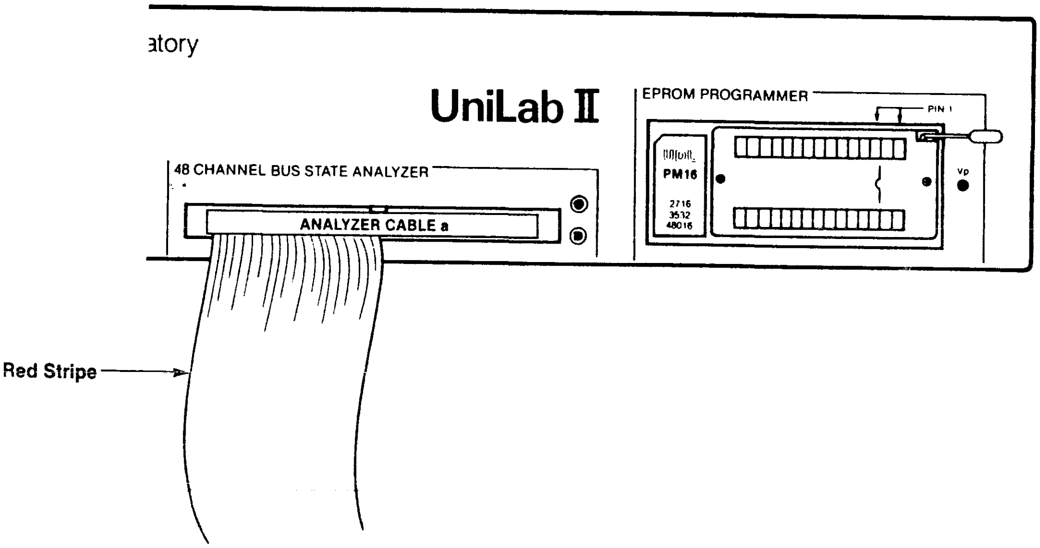
PLUG CABLES INTO UniLab CONNECTORS

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

4. CHECKOUT YOUR SETUP

LOAD A SAMPLE PROGRAM
RUN THE PROGRAM
COMPARE TO SAMPLE TRACE
PLAY AROUND A LITTLE
HOW TO EXIT
GET TO WORK

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 one of the disassembler/debugger packages. If you do not, then a sample program is not included in your software.

Where next?

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

LOAD A SAMPLE PROGRAM

Start up the UniLab program if you have not already done so. Hit 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" (**LTARG**). This will enable memory and load the simple demo program for your target processor.

RUN THE PROGRAM

To watch the program executing, first hit

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 writeup on your processor in chapter 9. See the next page (2-42) to find out how to have the computer compare your trace to the sample trace produced by Orion.

Trouble?

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/Debugger 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 check the contents of your distribution diskette.

If your trace checks out to be okay, then you can be confident that you have connected your UniLab properly and that your target board is working.

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 examining 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.

Using TCOMP

If **TCOMP** detects a difference between your trace and the known good trace that we send to you, it will show you part of the good trace, and then the first differing line of your trace. See also the entries on **TCOMP** and **TMASK** in the **Command Reference.**

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 points at RAM that does not exist.

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 target board. If you do not have RAM at that and lower addresses, then the program will pop garbage data off the stack, and you will not be able to set a breakpoint.

Look at the program as it executes, or look at the listing of it in the debugger 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 the UniLab command <word> <addr> **MM!**. You use that command to poke a new 16-bit word into the address field of the stack pointer initializing instruction. 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 do change the address of the stack, **TCOMP** will indicate a difference between your trace and the standard trace at that point in the program. You will have to visually inspect the trace to determine whether everything is properly connected.

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

PLAY AROUND A LITTLE

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. 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, hit **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.

GET TO WORK

You should take a break, and then use the UniLab some more, to set triggers and examine more traces. You will probably want to follow along with the **Guided Demo** in the next chapter.

Where to go Next

WITH TROUBLE

If you have trouble getting your system running, follow the suggestions in the **TroubleShooting** chapter.

WITH A FUNCTIONING SYSTEM

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

You can take one of four different pathways:

1) Go through the **Guided Demo** chapter first. You can just read through it, but will learn more if you follow it while seated at the computer.

2) Look at the chapter on **Getting Started**, then start using the instrument.

3) Use the UniLab in MENU mode, to gain familiarity with basic commands.

4) Start using the UniLab on a task, with the help of the Command Reference card.

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'.**

Chapter Three: Guided Demonstration

Introduction

This chapter shows you how to use the UniLab. It takes you through the process you must go through when analyzing your software. Before using this chapter, you should have already followed the installation instructions in Chapter Two.

This is not an exhaustive introduction to the UniLab, nor to the menus.

Instead, it illustrates the steps you will follow when testing and debugging any program on any processor.

For purposes of illustration, we use a Z80 processor and a very simple program. However, you will always follow the same basic procedure, no matter which processor you use or how involved your program.

Contents

Overview

This Guided Demo uses the menu mode 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 the instrument ready in steps one and two, double-check your preparations in optional step three, and then work with the program in steps four and five.

These are the steps that you will almost always follow when working with a program, whether you use the menus or commands. The heading of each page tells you what command you could use.

Warning

Though this chapter is laid out as a demonstration that you should follow on your computer, be aware that your traces will look different if you are using any processor besides the Z80.

The trigger specs and breakpoint addresses that you must enter are different from the ones in this text, as well as the name of the **.BIN** file that you read into memory and the locations into which you read it.

If you aren't certain what addresses to use for trigger specs and breakpoint addresses, you should look at the "sample session" in the Disassembler/Debugger writeup for your processor.

If you aren't certain what **.BIN** file to read in, or what addresses to load it into then consult Appendix J.

Where to Go Next

See the **Getting Started** chapter for information on the commands and the special features, as well as more information on the menu system. Or see the **In Detail** chapter for reference material.

Call Up the UniLab Software

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

1. display the opening screen,
2. initialize the UniLab.

And then you are ready for action.

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


 XXXXX disassembler installed - with debugger.

_____________________________________________________________
          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

Hit the function key 10 (**F10**) to get the main menu.

The first thing you must do when working with the UniLab on any program is enable a range of emulation memory. You enable memory to tell the UniLab which addresses it should respond to. You must do this before you load a program (with the exception of the test program loaded by **LTARG,** since that command enables the memory it needs).

```
                    UniLab MAIN MENU

              F1    ENABLE PROGRAM MEMORY
              F2    LOAD OR SAVE A PROGRAM
              F3    EXAMINE OR CHANGE PROGRAM MEMORY
              F4    WATCH PROGRAM EXECUTE
              F5    SET 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
```

Use function key 1 (**F1**) to get to the ENABLE menu from the MAIN MENU.

The Five-Step Procedure

1. Enable a segment of memory The command is: **EMENABLE**

We will be loading a program into the lowest 2K of memory, addresses 0 through 7FF. This range includes the reset address of the Z80 processor: 0000.

Your processor's reset address is probably different-- which means that you need to enable a different range of memory. Type in **LTARG** to find out what range of memory should be enabled for your processor. Note that the UniLab expects all numbers in hexadecimal.

The value of **=EMSEG** will also differ from processor to processor-- and is already set to the correct value for you.

Hit **F2** to enable a range of memory.

Notice that the menu system tells you which command gives the same effect as your menu choice.

This command shows what effect it has on emulation memory.

```
                          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
```

After you load the sample program, hit **F10** to get back to the MAIN menu.

2. Load a program into memory

The command is: **BINLOAD**

From the MAIN menu hit **F2** to get the LOAD menu.

```
              UniLab MAIN MENU

          F2    LOAD OR SAVE A PROGRAM
```

We shall load the short and simple program included on the distribution diskette. 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 what file to load in, or what address to start loading it at.

Use **BINLOAD** to load a binary format file. You will have to specify the starting and ending addresses for this memory load. The UniLab will stop loading when it finds the end of the file, or when it reaches the "Ending address," whichever comes first.

The program for the Z80 gets loaded in starting at address 0000.

With the Z80, the UniLab stops loading after address 31. This sample program is very short-- only 32 bytes.

```
                         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
```

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

3. Examine the program in emulation memory

Memory Dump The command is: **MDUMP**

From the MAIN menu, hit **F3** to get the EXAMINE MEMORY menu.

UniLab MAIN MENU

F3 EXAMINE OR CHANGE PROGRAM MEMORY

The simplest command for examining memory just dumps out the contents of memory, showing you the hex adecimal code and the ASCII interpretation of each byte.

Hit **F1** to "dump" a range of memory. Notice that the command works on 10 hex byte chunks of memory-- it dumps the full range 0 through 2F.

You will probably prefer to use the command that disassembles from memory, rather than just dumping memory. See the next page.

EXAMINE OR CHANGE PROGRAM MEMORY MENU

```
 F1   EXAMINE A BLOCK OF 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
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 03 00 8E 84 F7 A0   <<<<<<<<<.......
```

---

3. Examine the program in emulation memory

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

To see the disassembly of a range of program memory, use **F2.**

The 32 byte program is short enough that we can just disassemble the whole thing-- although we should do it about ten lines at a time, so that it fits on the screen.

```
                         EXAMINE OR CHANGE PROGRAM MEMORY MENU

                       F1   EXAMINE A BLOCK OF 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
                      F10  RETURN TO MAIN MENU


Enter the Starting address:0
Enter the number of lines to disassemble (default=5):10

 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
```

Hit **F10** to return to the MAIN menu.

Familiar? You might have already noticed that this simple program is identical to the Z80 **LTARG** program.

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.

From the MAIN menu, hit **F4** to get the ANALYZER menu.

```
        UniLab MAIN MENU

    F4   WATCH PROGRAM EXECUTE
```

4. Use the Analyzer

Reset the microprocessor, and watch the first cycles

The command is: **STARTUP**

In the ANALYZER menu, hit **F1** to get a trace of the first cycles of the processor as it executes the sample program.

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 numeric key pad.

If you were to look at the rest of the trace, you would see it continues with the series of "INC A" instructions, ending with a "JP 3" instruction at address 29. And then you would see the code at address 3 being executed again.

```
                        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
```

See section one of the **In Detail** chapter to find out more about interpreting the trace.

4. Use the Analyzer

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

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

You can get a sampling of the instructions that the program is executing right now. Hit **F5.** These samples are random selections from the bus.

You hit any key to stop the display of bus samples.

The transcript below confirms what was already pretty obvious from the trace-- the program spends most of its time executing "INC A" instructions.

Note that the disassembler sees one isolated byte (or word, with 16-bit processors), and therefore it fairly often sees only one part of an instruction that takes several bus cycles to be read from memory.

For example, the very first cycle captured by **SAMP** in the transcript below shows a read of DE. But 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, or leave it on while realizing that it can be "fooled."

```
                         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
```

4. Use the Analyzer

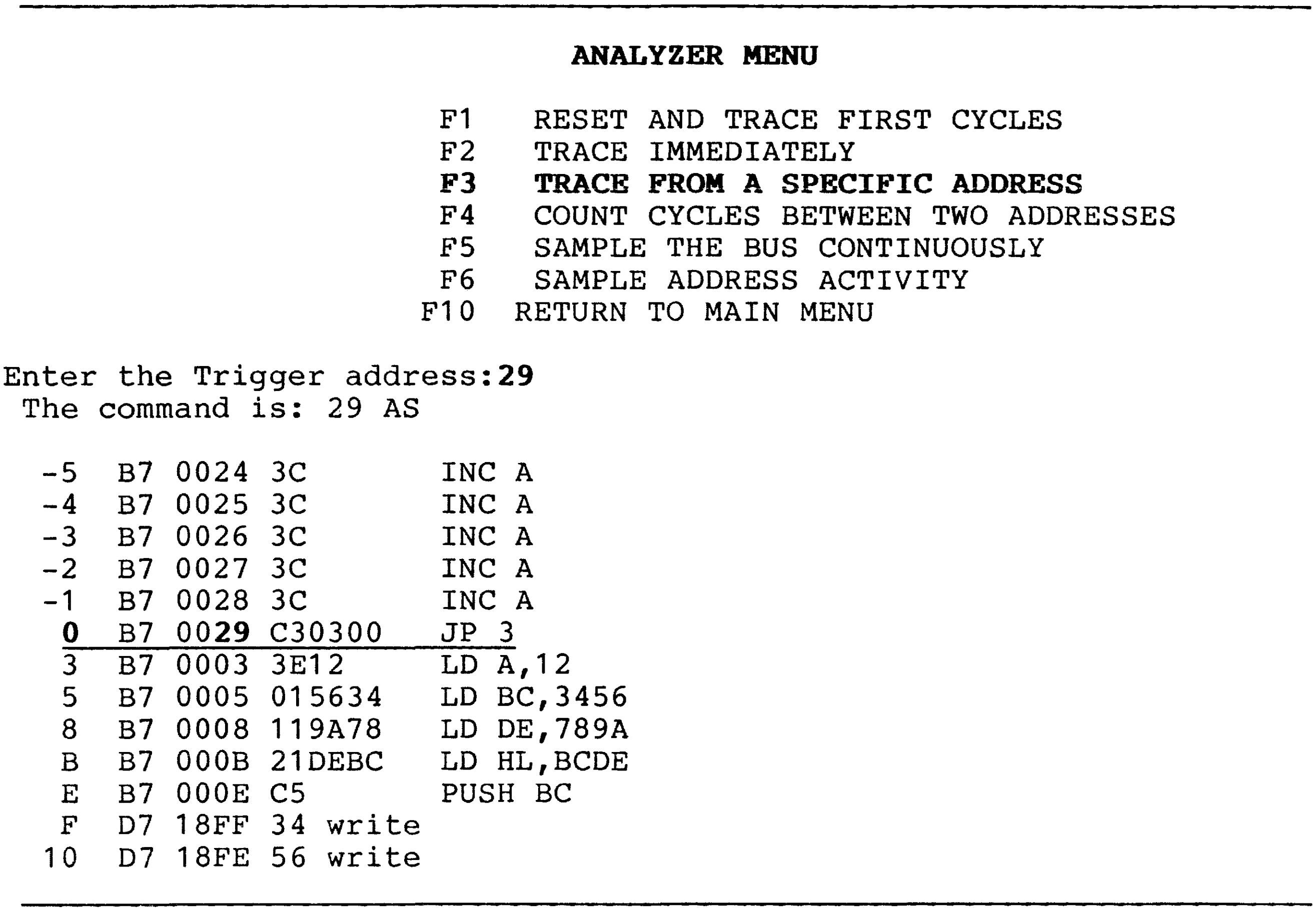
Set a trigger on an address The command is: <address> **AS**

Though the preset triggers of the UniLab are helpful, you will usually set up your own trigger specification.

Use **F3** to set up a trigger that will show the trace starting at whatever address you specify-- in this case, address 29, the address of the jump instruction. (Note that 29 is specific to the testZ80 program. The test program for your processor will have different instructions at different addresses.)

Note that the trigger address is labeled as cycle 0, and that the UniLab shows you the five bus cycles before that address.

As always, the program shows you only a screenful of trace. You must use **PgDn** or **Down Arrow** to see more of the trace 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

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 C30300   JP 3
   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
```

5. Use the Debugger

Set a breakpoint to Establish Debug Control

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

Now hit **F10** to get back to the MAIN menu, and then choose the SET BREAKPOINTS menu with **F6.**

```
            UniLab MAIN MENU

            F6   SET BREAKPOINTS AND SINGLE STEP PROGRAM
```

Whenever you want to use any of the debug features, you must first establish debug control-- which causes special hardware in the UniLab to take control of your microprocessor.

Hit **F1** to set a breakpoint. The example shows a breakpoint set at address **27,** two cycles before the JUMP instruction.

See the **In Detail** chapter for a complete explanation of the debugger and the breakpoint display.

```
                              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)
                      F4   GO TO AN ADDRESS WITH A BREAKPOINT SET
                      F5   GO TO AN ADDRESS AND EXIT THE DEBUGGER
                     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 PC=0027
0027  3C      INC A                          (next step)
```

---

5. Use the Debugger

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

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

F2 lets you set a breakpoint at a new address, and then releases the program from debug control. 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. Press **F2** and enter the value 3 to set a breakpoint at address 3.

That allows us to see the state of the processor immediately after it executes the JUMP instruction.

Of course, you could set this second breakpoint anywhere in the program.

```
                                   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)
                          F4    GO TO AN ADDRESS WITH A BREAKPOINT SET
                          F5    GO TO AN ADDRESS AND EXIT THE DEBUGGER
                         F10   RETURN TO MAIN MENU


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


    Enter the breakpoint address in emulation memory:3
     The command is: 3 RB
AF=2B28 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900 PC=0003
0003  3E12    LD A,12                        (next step)
```

---

5. Use the Debugger

Single-step through code The Command is: **N**

The UniLab single-steps by setting a breakpoint just after the instruction currently pointed to by the program counter. Then it releases your processor.

The processor executes the instruction that the program counter points to, and stops when it hits the breakpoint. Tap **F3** to single-step.

Note that in the transcript below, we begin with the processor stopped at address 3. Then we hit **F3** twice, to step through the next two instructions.

The breakpoint display shows that 12 was properly loaded into the A register by the **LD A, 12** 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)
                        F4   GO TO AN ADDRESS WITH A BREAKPOINT SET
                        F5   GO TO AN ADDRESS AND EXIT THE DEBUGGER
                       F10  RETURN TO MAIN MENU


AF=2B28 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900 PC=0003
0003  3E12     LD A,12                      (next step)

     The command is: N
AF=1228 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900 PC=0005
0005  015634   LD BC,3456                   (next step)

     The command is: N
AF=1228 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFF SP=1900 PC=0008
0008  119A78   LD DE,789A                   (next step)
```

Problems with **N**?

N sets a breakpoint at the address **following** the instruction pointed to by the program counter. If you use **N** when the program counter points at a jump instruction, then the program will not reach the breakpoint. It will seem as if nothing happened.

You're actually releasing the program from debug control with a breakpoint set at a code address that is not executed. Most processors support a command, called **SSTEP,** that allows you to watch jumps and branches.

Summary

We've gone through the process of loading a program, looking at it, and running it. We generated two traces of the program, and then set several breakpoints.

Though the program was very simple, the process you go through will be the same for any program.

More advanced work

Note that so far we have only set a trigger specification on 2 bytes of the 6 bytes of bus activity that the UniLab looks at during each bus cycle. The Guided Demo used the **AS** command to set a trigger specification on the 16 bits of the address input.

You can also:

1) set triggers on any combination of the 6 bytes of the UniLab inputs from the target board,
2) use qualifiers to delay the search for the trigger until after some other bus activity occurs.
3) and produce filtered traces that include or exclude whatever bus cycles you describe.

These capabilities are described in the **Command Language** section of the next chapter, and more completely in the **In Detail** chapter.

Learning

As you use the menus, you will gain familiarity with the UniLab methodology and commands. Most people find that they soon are operating in command mode and making use of the split screen **(F2)**.

But everyone turns to the menus for help with seldom used tasks, such as PROM reading and programming.

Next

See the next chapter, **Getting Started,** for a complete guide to the menu system, and an introduction to the commands and to the special features (such as mode panels and split screens).

Chapter Four:
Getting Started-- The Menus, the Commands, and the Special Features

Introduction

You use this chapter to get an overview of the capabilities of the UniLab software.

Before using this chapter, you should have installed the UniLab system on your personal computer, and have connected the instrument to your microprocessor board.

You would also benefit from looking over the **Guided Demo** chapter before reading this one.

When you require more information than Chapter Four has to offer, the **In Detail** chapter provides you with in-depth information about every aspect of the instrument.

Contents

Overview

The organization of this chapter mimics the stages you will pass through as you start using the UniLab.

When you start working with the UniLab, you will need the prompting and guidance that the menus provide.

After you have gained some familiarity with the instrument, you will find that your work goes faster using commands.

You will soon find that you want to look at several different types of information at the same time. That is when you will need and appreciate the split screen-- though the split screen will be valuable to you from the moment you start using the instrument.

The bulk of the chapter covers these three major topics:

Menus,
Commands,
Special features.

This third section contains a tutorial on the use of function keys, cursor keys and split screens (windows).

When to use: Menus and commands and special features

Menus

Use the menu mode when you first work with the UniLab. The menus help you by guiding your activities. The menus also help you learn, by telling you which command corresponds to the menu selection you have made.

The **Menu Mode** section of this chapter shows each of the menus, and explains what each menu choice does. Consult this section to learn what functions each menu performs, and which commands correspond to each menu choice.

You don't have to exit from menu mode to try out commands. At any time, you can type in most UniLab commands. After a short time, you will be ready to work without menus.

Commands

Use the command mode when you have a passing familiarity some of the UniLab's functions. When you are in command mode you can make use of the full power of the UniLab, and take advantage of the special features that are <u>not</u> available in menu mode, such as split screens, calls to DOS, the mode panels, and text file review.

The **Command mode** section of this chapter introduces you to the use of commands, concentrating on setting trigger specifications. Consult this section to learn the command conventions, and find out how to use UniLab commands to capture information about your target processor program.

This section also includes information about the use of command tails and batch files with the UniLab software.

Special features

You can use the special features as soon as you start using command mode. The UniLab software provides you with:

split screens,
reassignable function keys,
mode panels for easy toggling of options,
a history of the current session,
and other special features.

These easy to use features are completely explained by the third section of this chapter.

1. The Menu Mode

The UniLab software provides a powerful command language that allows control of all facilities from a single context. However, command languages can be intimidating to first-time or infrequent users.

The menu system allows you to gain familiarity with the UniLab-- and to test commands from within the UniLab.

You enter (and leave) the menu system with function key 10 **(F10)**.

Choosing options

You make choices from the menu by pressing a function key.

Whenever you make a menu choice, the program will ask you for any needed parameters, tell you what command you would use, and then execute the command.

The test in the following pages also tells you the command corresponding to each menu choice.

The arrangement of options

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

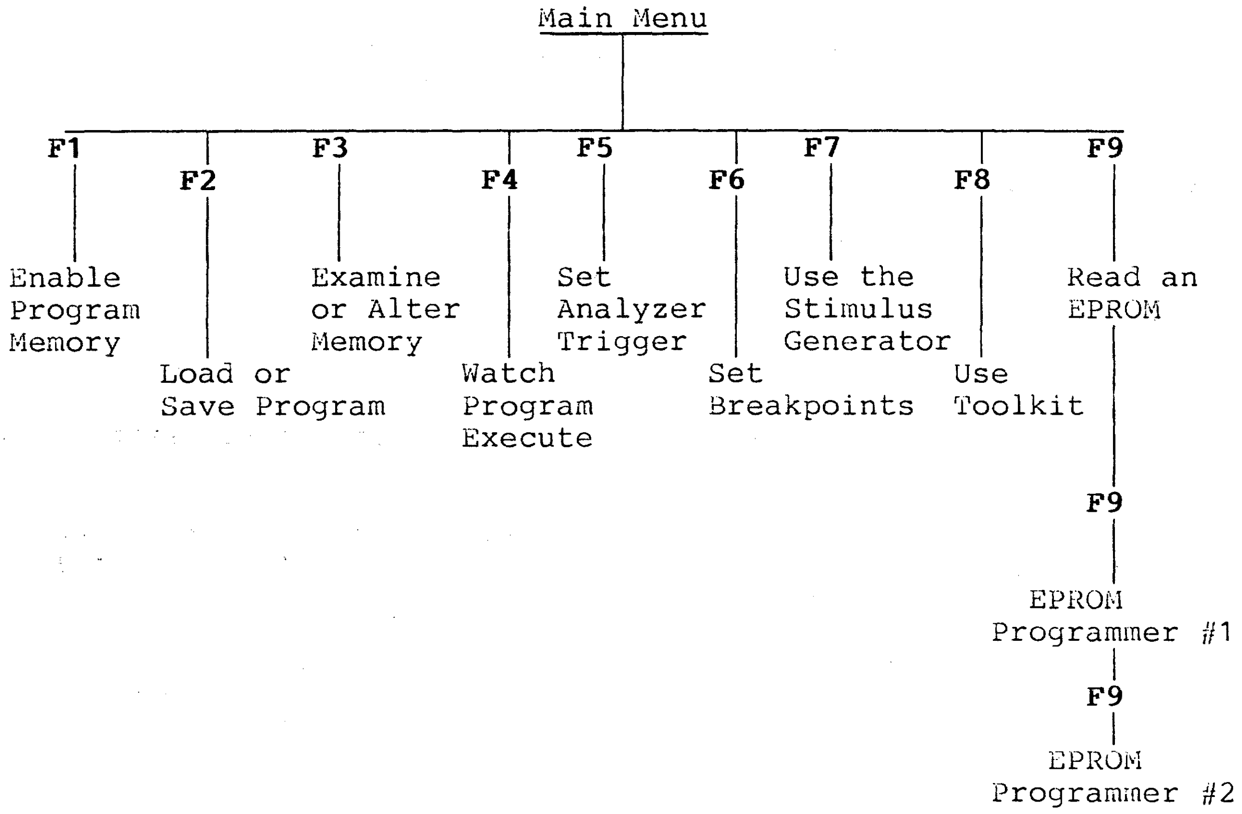
Using commands from inside menu mode

You can use most of the UniLab commands from within the menu mode, though of course you cannot use the function keys to call up commands. Also, the mode panels and windows are not available to you.

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.



From the Main menu, **F10** puts you into command mode.

From any of the sub-menus, **F10** puts you into the Main menu.

The UniLab system will present you with the main menu when you enter menu mode by hitting function key 10, or when you type the command **MENU** :

```
                         UNILAB MAIN MENU


                    F1   ENABLE PROGRAM MEMORY
                    F2   LOAD OR SAVE A PROGRAM
                    F3   EXAMINE OR CHANGE PROGRAM MEMORY
                    F4   WATCH PROGRAM EXECUTE
                    F5   SET 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 you will need to use them:

When checking out a program, you need to enable memory **(F1)**, 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 probably need to do both of these at different times.

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)**, but you will need to use commands to make use of the full power of the UniLab.

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 debugging work. But if you want to do it, you must first establish debug control by setting a breakpoint. You can then single-step from that point in the program.

When you are done with your debugging work, you can burn the tested program into an EPROM **(F9)**.

Hit **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

Because you can only put a program into a range of emulation memory that has already been enabled, you will want to check the current status of emulation memory (**F1** or **ESTAT**) before loading in a program.

If the range of memory you need isn't enabled, you will need to correct that (**F2** or <start addr> <end addr> **EMENABLE**). This clears out any previous enable settings.

You might later want to enable another range of memory without clearing out the previous setting (**F3** or **ALSO** <start addr> <end addr> **EMENABLE**).

You can use **SAVE-SYS** <filename> to save the UniLab software after you enable the range of memory that you need for your project. That way, you only need to enable memory once.

The upper four bits of the address are set properly for you, unless you don't have a disassembler software package, or have an abnormal memory map. In the unlikely event that you need to change that setting you can (**F4** or <hex digit> **=EMSEG**). The change does not have any effect until the next **EMENABLE** command. See the first section of the **In Detail** chapter if you need more information about the upper four bits of the address.

You will never need to disable all emulation memory, unless you want to run a program from a ROM chip on the microprocessor board (**F5** or **EMCLR**). Before you can analyze a program running from a ROM, you will need to disable the debugger (with the mode panel choice **SWI VECTOR** or **RSP'**).

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

Hit **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

This sub-menu allows you to load two types of program files into emulation memory.

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 the address in emulation ROM to start and the address to end (**F2** or <from addr> <to addr> **BINLOAD** <file name>). You must stay aware that the program will load until it reaches the <to addr>, or until it reaches the end of file, whichever comes first. If you are not cautious, you might accidentally load only part of your program.

You can just as easily save a range of memory to a disk file (**F3** or <from addr> <to addr> **BINSAVE** <file name>).

Loading and running the simple test program can be a valuable way to test out your UniLab (**F4** or **LTARG**). Watch out, since this command changes the enable status of emulation ROM.

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

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

EXAMINE OR CHANGE PROGRAM MEMORY MENU

F1	EXAMINE A RANGE OF 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
F10	RETURN TO MAIN MENU

Explanation

You can perform all memory operations on RAM after you have established debug control. Otherwise you can only work on enabled emulation ROM.

You can examine memory by performing a hexadecimal dump of some range **(F1** or <from addr> <to addr> **MDUMP).** This command operates on 10-byte blocks of memory, and shows you the numbers stored in memory.

You will probably find it more valuable to see the disassembly of the instructions stored in memory **(F2** or <from addr> <number of instructions to decode> **DM).**

You might want to alter the instructions in program memory. If you know the number that corresponds to an instruction, you can push into memory either a byte **(F3** or <byte> <address> **M!**), or a 16-bit word **(F4** or <word> <address> **MM!**).

If you choose, you can make use of a more heavy-handed way to alter memory, and fill an entire range of memory with the same byte value **(F5** or <from addr> <to addr> <byte value> **MFILL).** This command is useful for putting a bunch of identical instructions into program memory, for testing purposes.

Sometimes when you need to alter your program, you need to move blocks of code from one place to another **(F6** or <start addr source> <end addr source> <start addr dest> **MMOVE**).

For testing purposes, and in the course of verifying PROMs, you might want to find out how two areas of memory differ, if they do differ **(F7** or <from addr> <to addr> <comp addr> **MCOMP).** This command operates on 100-byte blocks.

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

Hit **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

Especially when you first start testing a new piece of software for your microprocessor board, you will want to look at the first cycles that it executes **(F1** or **STARTUP)**.

Another way to explore your program when you first start working with it is to capture the trace of what it is now executing **(F2** or **NOW?)**. You can also use this command to demonstrate to yourself that your processor does not stop when you capture a trace-- while you are examining the "instant replay," the game continues.

One of the simplest and most frequently used trigger specifications tells the UniLab to show you what happens after the program reaches a given address **(F3** or <addr> **AS)**.

While trying to figure out where your program is spending its time, you will find it useful to count the number of bus cycles that occur between addresses **(F4** or <first addr> <second addr> **CYCLES?)**.

You can get a rather rough grained view of your program by displaying a sample of bus activity, randomly snatched from the bus, one per second **(F5** or **SAMP)**.

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

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

Hit **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

The menu mode does not provide you with the full power and flexibility of the command mode-- but the ANALYZER TRIGGER MENU does provide you with some of the most useful trigger commands.

This menu repeats the option from the fourth menu, allowing you to set a trigger on an address **(F1** or <addr> **AS).**

You can also trigger when any member of some range 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 from within the menus.

But the menu 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**).

You can also trigger when the processor goes outside a given range of memory, for either reads or fetches (**F4** or **NOT** <addr> **TO** <addr> **ADR S**). This trigger specification is more useful as a command, since you can, with most processors, limit the trigger event still further by specifying **READ** or **FETCH.** This lets you, by specifying the appropriate addresses, trigger when the processor tries to fetch from outside of ROM, or read from outside of RAM.

You often find that the majority of your trace is filled with the record of some delay or status loop that occurs again and again. You will find it useful to capture a trace that triggers on one address, and excludes some other range of addresses from the trace (**F5** or **ONLY NOT** <addr> **TO** <addr> **ADR AFTER** <trigger addr> **ADR S**).

When you have RESET turned on, your target board starts the target program from the reset address when the analyzer starts watching the bus for a new trigger-- that is, whenever an **S** is issued.

With RESET turned off, the analyzer will search the program "in progress" for the trigger whenever an **S** is issued.

You can toggle RESET on and off (**F6** or **RESET** and **RESET'**).

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

Hit **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)
 F4   GO TO AN ADDRESS WITH A BREAKPOINT SET
 F5   GO TO AN ADDRESS AND EXIT THE DEBUGGER
F10   RETURN TO MAIN MENU
```

Explanation

After you have established debug control, you can read and alter RAM, read and alter internal registers, single step through your program, and perform many other traditional debugging functions. The menu does not provide you with access to all the features available in command mode.

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

From a breakpoint, you can instruct the UniLab 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.

From a breakpoint, you can also step through instructions one at a time, looking at the internal register display after each step (**F3** or **N**). This command will not follow jumps, calls or branches, but processors with a hardware Non-Maskable Interrupt support a **SSTEP** command. See Appendix H.

The **SSTEP** command allows you to follow non-sequential code. See the chart in Appendix H to find out whether your processor supports this and other features that depend upon the Non-Maskable Interrupt.

From a breakpoint, you can also change the program counter and then set a breakpoint, before releasing the processor from debug control (**F4** or <New PC> <addr> **GB**).

You can also change the program counter and then release the program from debug control without a breakpoint set (**F5** or <New PC> G).

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

Hit **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

You can 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.

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

You can also "reset" (change to low signal) a single bit of the stimulus output (**F2** or <bit number> **RESET**).

You might prefer 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.

Hit **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 you with some reference material that you will find invaluable.

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.

Hit **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

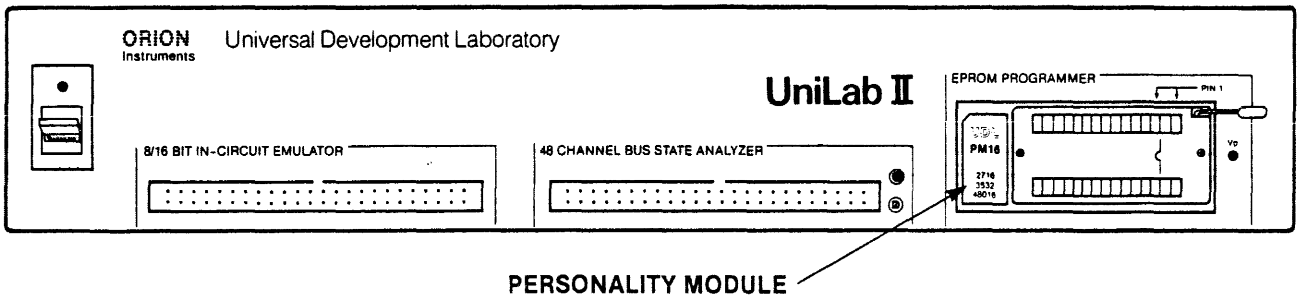
You use this menu to load a program from an EPROM into the UniLab's emulation ROM. Each entry on the menu tells you which **P**ersonality **M**odule (PM) you need to use while reading from the PROM. The PM nestles next to the EPROM PROGRAMMER socket, and alters control signals as necessary for each PROM.

You can perform all EPROM programming and reading from within the menu system. We recommend that you not bother to make use of the commands, though they are available for use in macros.

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

This menu also provides you with access to the PROM PROGRAMMER MENU. Hit **F9.**

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


ORION
Instruments
Universal Development Laboratory
UniLab II
EPROM PROGRAMMER
PIN 1
8/16 BIT IN-CIRCUIT EMULATOR
48 CHANNEL BUS STATE ANALYZER
PM16
2716
3532
48016
VD
PERSONALITY MODULE

Hit **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

You use the two PROM PROGRAMMING menus to program any EPROM that Orion supports.

Hit **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.

Hit **F9** from the first PROM PROGRAMMER sub-menu to get the <u>second</u> 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.

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

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

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

2. The Command Mode

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

Common commands are assigned to some function keys, while 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).

The key functions are covered in the **Special Features** section of this chapter.

The UniLab software understands the command words and hexadecimal numbers. Words must be separated from one another by at least one space. Multiple commands can be strung together on a single line if desired.

Entering command mode

You enter the command mode by pressing function key 10 **(F10)** from the main menu.

Of course, you are normally in command mode when you first start the program.

Most commands can also be executed from within the menu mode. Most of the exceptions are window and pop-up panel display words.

You can re-enter menu mode by pressing **F10** at any time.

Command Tail and Batch Files

The optional information on this page and the two following tells you how to use the batch file facility of DOS and the command tail feature of the UniLab software to automate the process of producing object files and loading them into the UniLab's emulation ROM.

Command tail

The UniLab program allows you to include a "command tail" on the DOS command line, each time you call up the software. You do it this way:

```
C> ULZ80 <UniLab command>
```

The UniLab command 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 start up, you could type in:

```
C> ULZ80 MENU
```

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 DOS text files that contain commands. Their names should end in **.BAT.** The easiest way to make them is to copy from the screen (CONsole) into a file.

For example, if you will always want to load in the contents of a binary file, you could **make** your batch file like this:

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

Notice that you have to finish entering information by pressing **CTRL-Z,** followed by a carriage return.

Once the batch file was made, you would **call** it this way:

```
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
```

You would then be able to alter the source file with the editor. When done editing, you could exit from the editor, which would return you to the batch file. You would not have to touch the keyboard again-- the batch file would assemble the code, erase the old version of the program, link a new version, load the program into emulation memory, and start it executing.

You could work on something else, and avoid the tedium of going through these mechanical steps.

Complex command tails

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 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 command line:

```
                              C> ULZ80  LOADUP
```

By using a macro, you would be able to avoid the limitation on the length of your command tail. Be sure to use the UniLab command **SAVE-SYS** after you make a macro that you want to preserve. See Appendix F and the entry for **:** in the **Command Reference** chapter.

Using the Command Language

Controlling the UniLab with commands doesn't differ much from controlling it with the menu system-- except that you have:

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

But nothing fundamental changes. You approach a task in the same way.

Whether you use commands or the menu mode, you:

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

The big difference is that you make up any trigger specification you want, as explained in the next few pages.

The mode panels, split screen, and several other features are described in the third section of this chapter, **Special Features.**

Saving the emulation settings

You can avoid repetition-- you don't have to enable memory every time you start up the UniLab software. Instead, use

SAVE-SYS <filename>

to save the current state of the system after you have enabled the proper range of memory.

After that, call up the UniLab software by using the new name. Orion recommends that you always save the system to the same name, rather than keeping several versions of the UniLab software. You can always retrieve the original, unaltered verstion from the distribution diskette.

Trigger Specs: Theory and Conventions

The UniLab allows very complex triggers to be defined, using all 48 analyzer inputs. For more details consult sections 1 and 4 of Chapter 6.

The groupings

Every bus cycle, the UniLab software reads six bytes of inputs from the target system's bus: two bytes of address, two bytes of data, 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.

The names

Each of the groupings of inputs is referred to using the same descriptive name used to label 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. **LADR** and **HADR** each label one byte of the address inputs.

The **HDATA** column appears on the trace only with processors that have an 8-bit data bus. 16-bit processors show both the **HDATA** and the **DATA** byte under the **HDATA** column.

Setting a trigger

To set a trigger on a single value, you first clear out any previous definitions with **NORMT** or **NORMM** or **NORMB**, 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 data on 16-bit processors, or on anything you like with an 8-bit processor.
<8-bit value> **MISC**	to trigger on anything you like. (Usually target system inputs and outputs.)

Starting the analyzer

After you have defined a trigger, you start the analyzer with the single letter command **S**. You will usually enter the **NORMx** word, the trigger specification and the start command all on the same line:

NORMT 100 ADR S

to see a trace of what happens after address 100 appears on the bus.

Triggering on ranges

You can trigger on a range of values by using <value> **TO** <value> instead of a single value. For example:

10 TO 1A DATA

You can trigger outside a range by preceding the range with the keyword **NOT**.

Examples

The next several pages show several of the most useful ways to put together trigger spec commands, and explains a bit about each trigger spec. These examples are mostly drawn from the **Command Reference** chapter of the manual.

Notation conventions

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

Command conventions

Many commands must be preceded by one or more parameters. The command can be entered in upper or lower case. Commands that require a file name, such as **SYMSAVE**, **BINLOAD**, etc., will prompt you for the file name if you do not enter it.

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.

Other number bases

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

Spaces

Spaces are used to separate commands. It doesn't matter how much "white space" (blank spaces) you use, as long as you do use at least one space. The absence of space can cause the UniLab software to misinterpret your intentions.

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.

Trigger Specifications: Examples

Simple triggers

Trigger when one condition on one grouping is met.

Trigger on a value

1205 AS
clear out the previous trigger, set up a new trigger on address 1205 appears and start the analyzer (abbreviated version).

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

NORMT 12 DATA S
after clearing all previous settings with NORMT, 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 range with the CONT grouping

NORMT 70 TO 7F CONT S
triggers when any one of a range of values appear on the CONTrol lines of the UniLab. Because of the mixed nature of this grouping, this trigger spec requires C7=0, C6-C4 = 1, and A16-A19 = any value.

Trigger outside a range

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

"AND" triggers: Trigger when several conditions are true

Trigger when one condition holds on one grouping and an independent condition is met on another grouping.

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.
```

"OR" triggers

Trigger when one condition or another is met by one of the groupings of UniLab inputs.

NORMT 3 DATA ALSO 7 DATA S
triggers when the data is either 3 or 7

NORMT 43 DATA ALSO 20 TO 3E DATA S
triggers when the data is either 43 or between 20 and 3E hex.

NORMM NOT 12 DATA ALSO NOT 34 TO 56 DATA S
triggers when the data is not either 12 nor between 34 and 56.

"OR/AND" triggers

Trigger when any one of several conditions holds true on one grouping, and another condition holds true on another grouping.

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

Filters: Only show. . . .

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.

The x**AFTER** commands are especially useful for finding what part of the program is causing bad data to be written to some address or some range of addresses. The second example below will show you all access to address 1200, and then the first byte of the next instruction.

When you filter the trace, the cycle numbers will be marked with "f" for filter.

One cycle

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 ten and thirty.

Two cycles each trigger

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

Qualifiers: Start searching for trigger after. . . .

Don't start to look for the trigger until after some other condition has been seen on the bus.

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.

If the first qualifier appears on the bus and the very next cycle is not the second qualifier, then the UniLab will start looking for the first qualifier again.

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

Add a second qualifying event

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 there is a conditional jump to 1500 at address 235. If that jump is not taken, the UniLab will start looking for address 235 again.

Qualifiers and 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.

ONLY READ AFTER 1235 ADR S
show only reads from RAM, starting after the code at 1235 is fetched. NOTE: **READ** is a processor specific macro. It is not defined on some processors.

Exclude a loop from trace

Another very useful spec: trigger on one address, and filter out a status loop from your trace.

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. Special Features: Function keys, Cursor keys, and Windows

This section tells you more about how to use the function keys and the cursor keys. The function keys allow you to call commands with a single key stroke.

For example, you use **F2** to split the screen, **F8** to enter the mode panel.

The cursor keys are used to move about on the screen--between windows or within the mode panels.

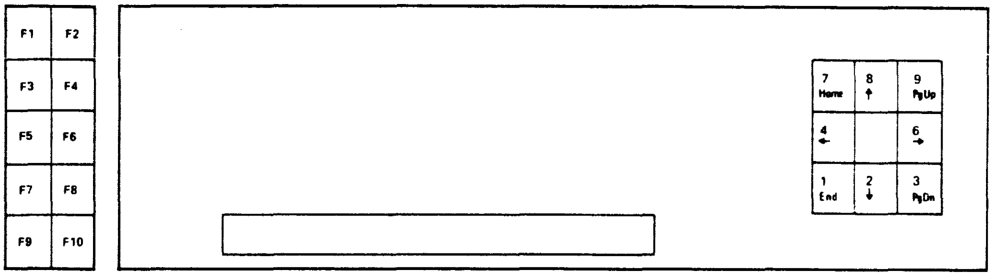
Function Keys

This section tells you more about the features that have been pre-assigned to function keys, and explains how to reassign the function keys.

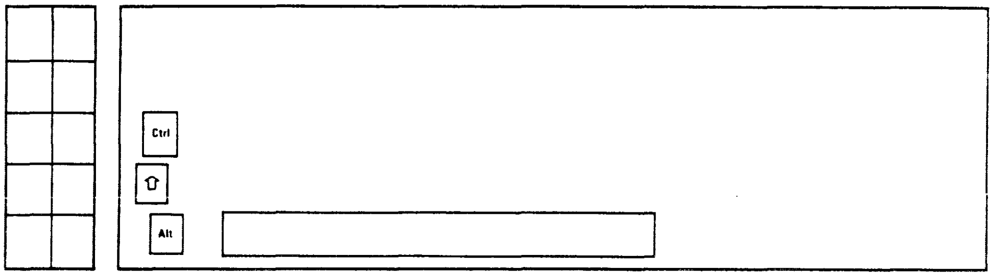
As you see from the chart on the next page, many of the forty functions have been left unassigned. These keys can be assigned by you to any command you choose, as long as that command does not require parameters. For example, to assign the command **SAMP** to ALT-F5, you would type in **5 ALT-FKEY SAMP.** You can also assign to a function key a macro that you have defined. See Appendix F to learn how to define a macro.

The four commands that assign commands to function keys are:

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



Function Keys and Cursor Keys



ALT, SHIFT and Control Keys

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	Ascii display - Shows ascii values for keys.
----	F5	F6	----
----	F7	F8	Set new window split size
----	F9	F10	----

Function Key assignments when Alt key held down

List Function Key assignments for Alt	F1	F2	----
----	F3	F4	----
----	F5	F6	----
----	F7	F8	----
----	F9	F10	----

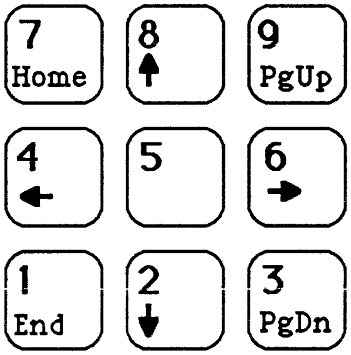
Function Key assignments when no other key held down

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 NMI pulse to target to get breakpoint.
Restore window split to Default sizes.	F5	F6	Single Step - Execute next instruction. Will follow jumps and branches. May be same as NMI.
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.

Cursor Keys

The cursor keys on the computer will be used to move around to see different parts of the trace. Since you will want to go forward and backward, the keys you will be using are these:

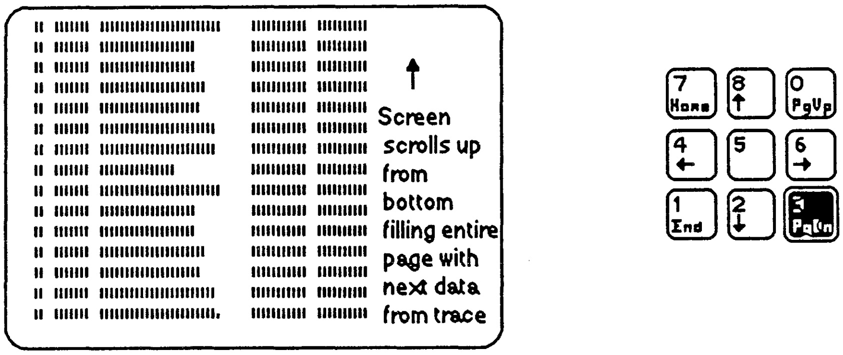
Up Arrow	(also labelled with an 8)
Down Arrow	(also labelled with a 2)
PgUp	(also labelled with a 9)
PgDn	(also labelled with a 3)



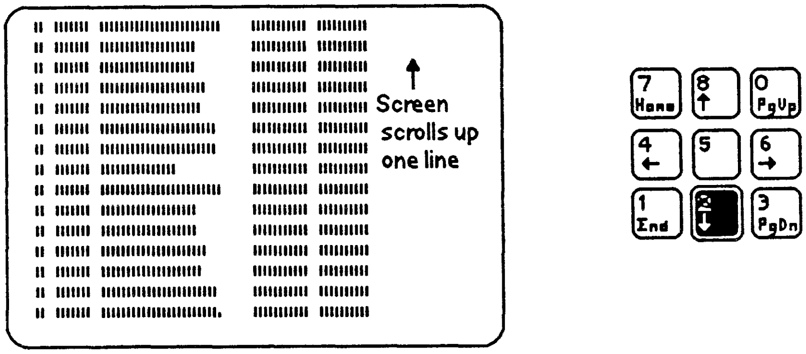
Moving through the trace

Use **LTARG** followed by **F9 (STARTUP)** to get a trace of the first 170 cycles of the sample program's operation. Or use whatever trigger spec you want to generate a trace of your program.

Now that you have a display of the first part of the trace, press the **PgDn** cursor key to display the next screen full of data:


Screen scrolls up from bottom filling entire page with next data from trace
7 Home
8 ↑
0 PgUp
4 ←
5
6 →
1 End
2 ↓
3 PgDn

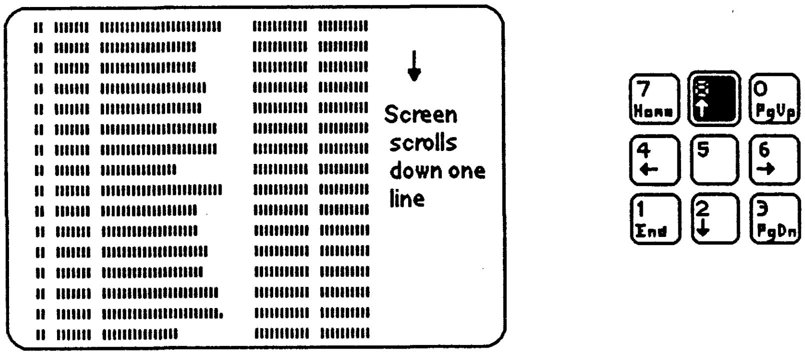
Press **PgDn** again to see the next portion of the trace buffer. Do not press **PgDn** more than twice for now-- you would come to the end of the trace. That gets you a whole new screen of information. If you want to see just one or two more lines press the **Down Arrow** cursor key:



One additional line is shown, and everything else is scrolled up by one line. Press the **Down Arrow** 4 or 5 times to see a few more lines.

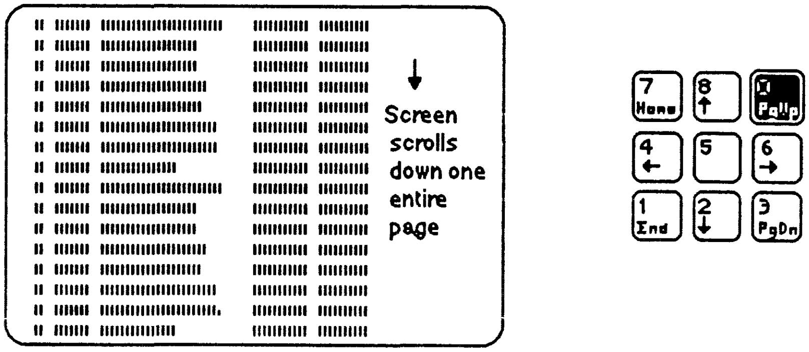
The history mechanism

The **Up Arrow,** of course, must scroll back one line. Even though it appears to be just the opposite of the **Down Arrow,** it is remarkably different. First, try it once:



It seems as if you are looking back through the trace. But in fact you are looking back through the record of everything that has appeared on your screen.

Now do a **PgUp** to see more of the trace buffer before this line:



You will notice that the screen display backed up a whole page, but the display looks a little funny. There is a break in the display in the middle of the page with lines that look like the heading for the trace display:

cy# Adr Data etc.

a blank line and a line that looks like the prompt at the end of the last display

PgDn or <trace resume> Home, etc.

When you scroll back, you are really looking 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 will be explained later). 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.

This means that 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.

Summary: Line history and trace listing

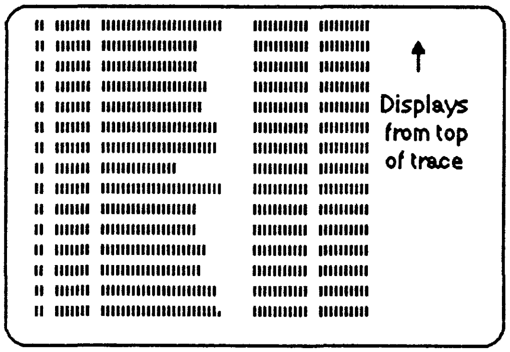
Going forward with the **Down Arrow** or the **PgDn** cursor keys always shows new analyzed data from the trace buffer, employing the resident disassembler if it is turned on.

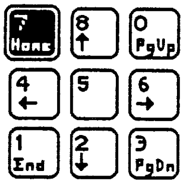
Going backward with the **Up Arrow** or the **PgUp** cursor keys will show the history of the current session with the UniLab.

See trace from top

Going backward with the Up Arrow or the PgUp keys will always show data that was scrolled off the top of the screen. It is recorded history rather than newly analyzed data. A different cursor key shows the trace listing from the top.

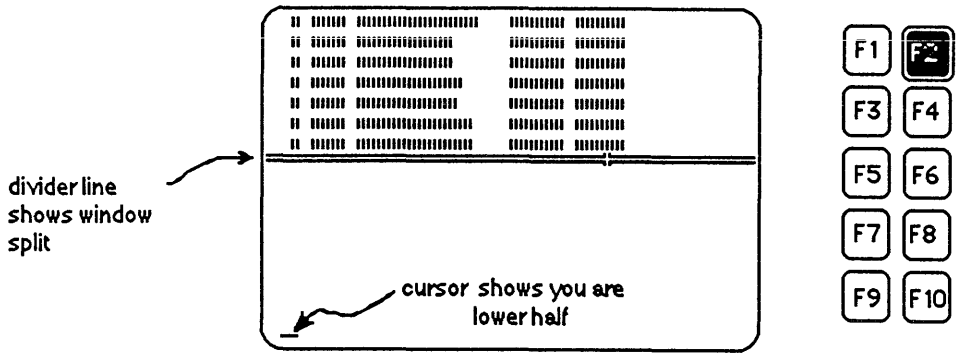
The **Home** key shows the trace starting from the very beginning, the top of the trace buffer. Press the **Home** key:





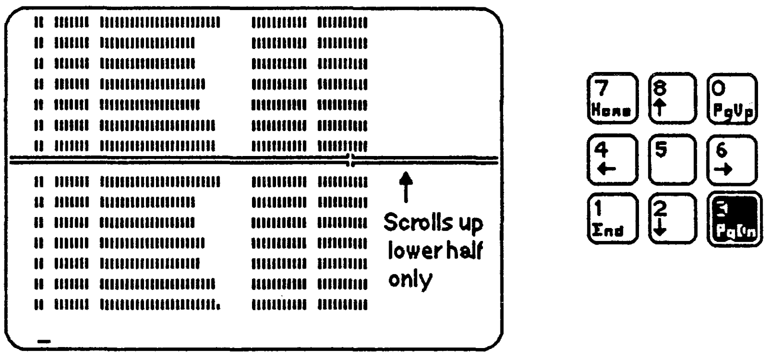
Windows

Now you're probably eager to hit the **End** key, but please don't yet. It won't do anything until we use one of the function keys. Press **F2:**


divider line shows window split
cursor shows you are lower half
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10

You should see a display like the one shown above. The lower portion of the screen has been blanked out, and the upper section still shows your last action.

Notice the horizontal dividing line, and the flashing cursor in the bottom line of the screen. Press the **PgDn** key:

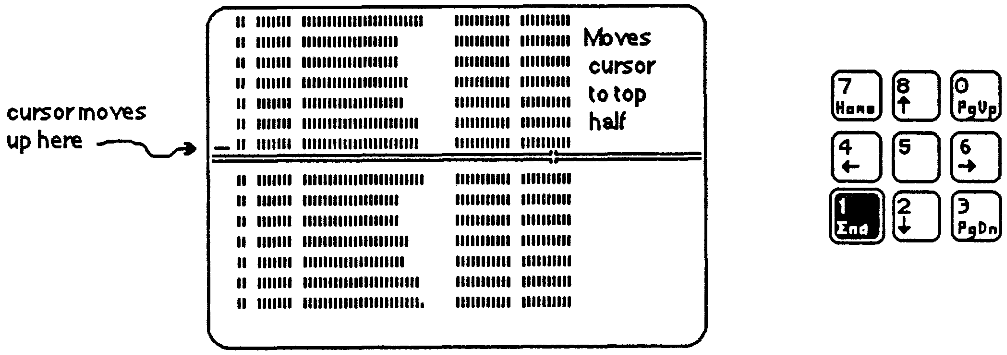


The trace listing scrolls up the screen, but it stops when it fills only the lower half of the screen. Hit **PgDn** again, and notice that the trace listing scrolls by in the lower half of the screen only.

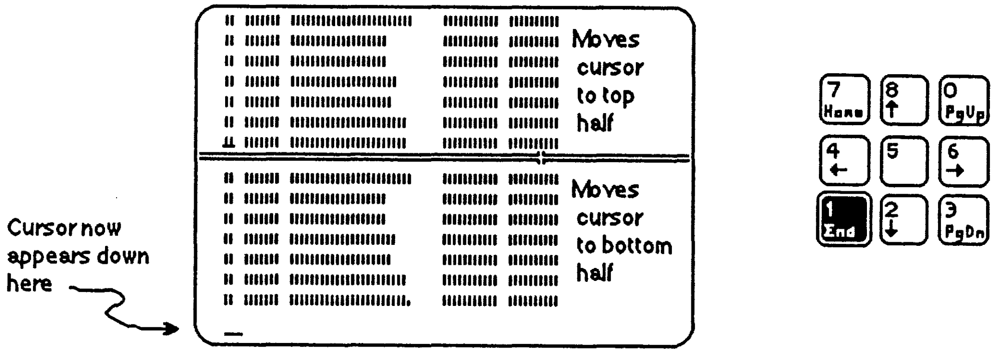
Moving from window to window

The display is now set up for two viewing windows. Hit the **RETURN** key four or five times. Type the word **ESTAT.** All of your commands and their actions are displayed only in the lower window.

Now you can use the **End** key. Press it once and notice that the cursor goes into the upper window:



Press it again:



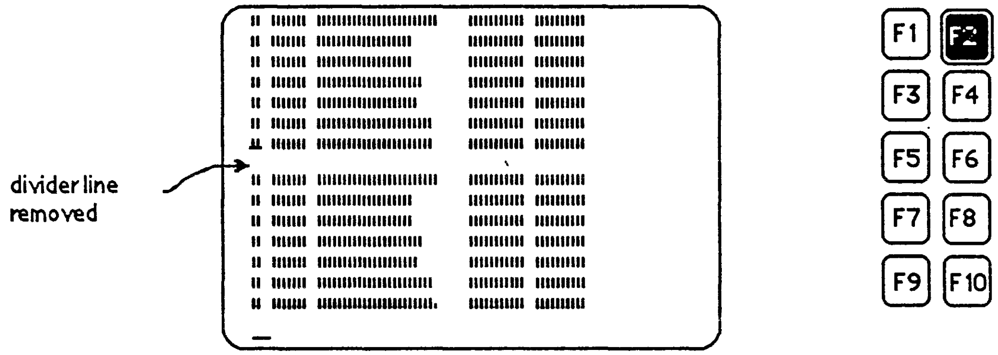
The **End** key moves you back and forth between the two windows. You can do this at any time. It is especially useful if you have a portion of the trace that you want to refer to while you are doing other operations on the screen. Try the cursor keys in both windows. If you know other UniLab disassembler or debug commands, go ahead and use them.

A note on history and windows

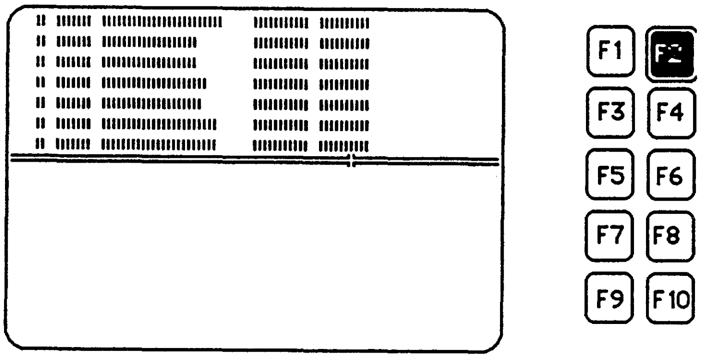
Here is an important difference in the way the line history works in the upper and lower windows. Lines scrolled off the top of the upper window are not recorded. If you want the trace recorded, do it from the bottom window or from the full screen mode. Line History can be viewed in either window.

Leaving split mode

How do you get out of this? Hit the **F2** key again (the same one that got you into the split mode):



The divider line is erased. If you scroll now, the entire screen will be used for the display. Press **F2** again to get a split screen back on the display:

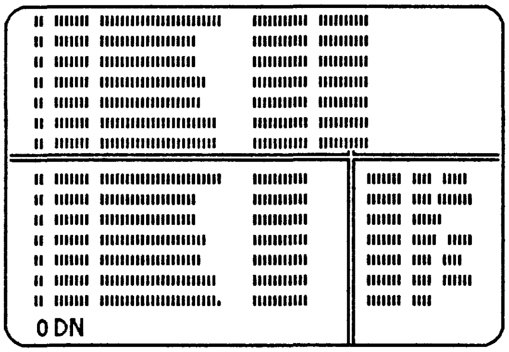


Now press the **Home** key to bring the top of the trace into the lower window. If you have done any other command since you executed **STARTUP (F9)**, hit **F9** again, then press **Home** so that you will have a trace that starts from the reset address.

Showing disassemble

Type **0 DN** followed by a return. You should see a display like below. **DN** is a special word that means disassemble from memory and display it in the right window.

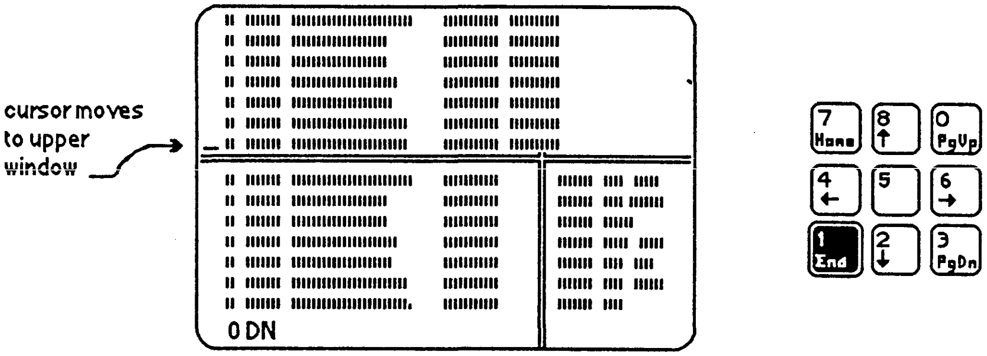
DN works just like **DM** command, except that it always fills up the right hand window, and so does not need the number of lines as a second parameter.



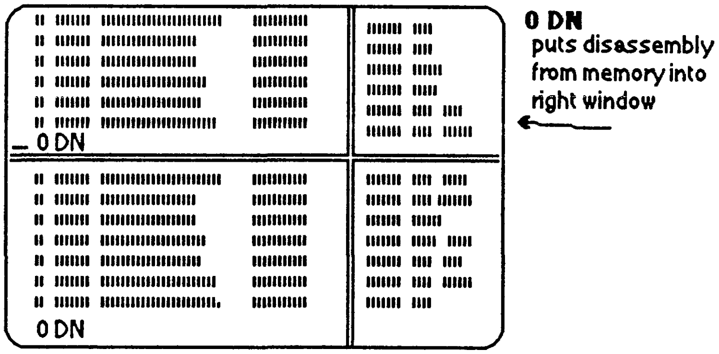
0 DN
puts disassembly
from memory into
right window

You should now see in the lower two windows a trace of the program after the target was reset (in the big left-hand window) and a listing of the disassembled code (in the little right-hand window). This right hand window is used only by the **DN** command. You can see a disassembly starting from any address with <addr> **DN**.

You can also use the upper right hand window. First press the **End** key to move the cursor up there:



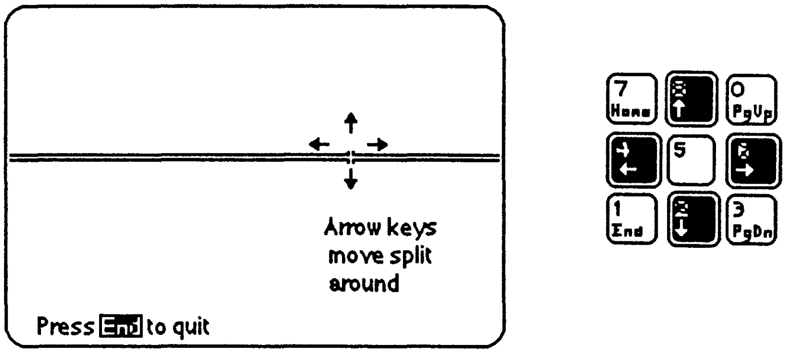
Use **DN** again. This time it will display in the upper righthand window:



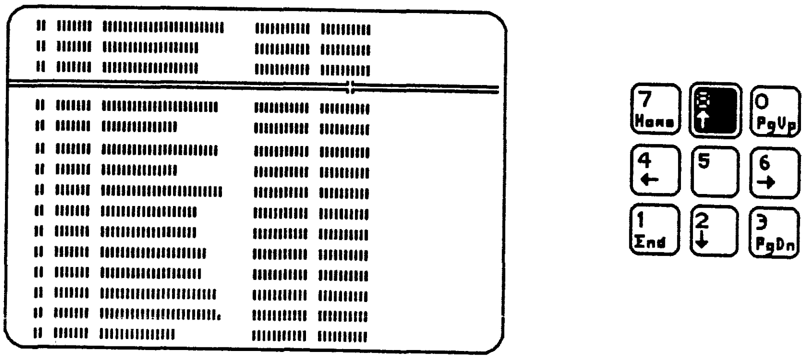
You can also use **DN** with the full screen.

Changing window size

If you would rather have a larger lower window, you can change the size of the split by hitting the **F8** function key while holding down the **Shift** key (the hollow arrow on the left side of the keyboard, not the cursor **Up Arrow** key). You will see this display:

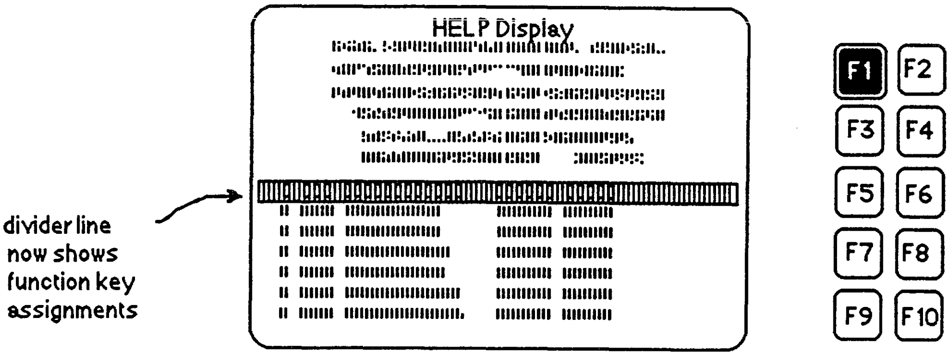


Use the cursor arrow keys to move the split up or down, and the right or left. Move the split up to make the top half a little smaller than it was. Hit the **End** key when you want to exit this size-setting mode. No other keys will have any effect while you are setting the window partition. Try displaying the trace in the new window sizes:

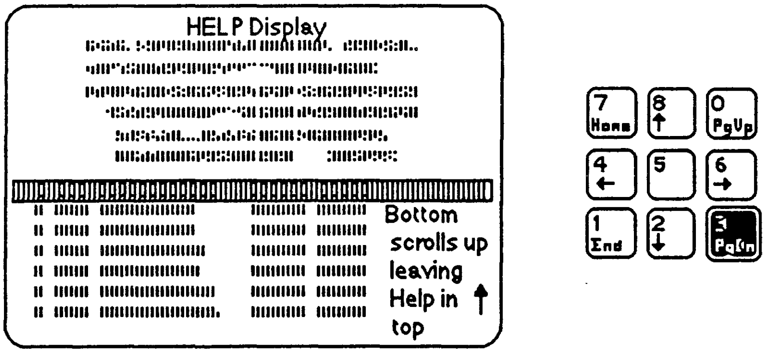


Split screens and help displays

When in the split screen mode, the HELP screen displays will automatically change the size of the window so that the text of the help display fills the upper window. Press function key **F1** to see this:

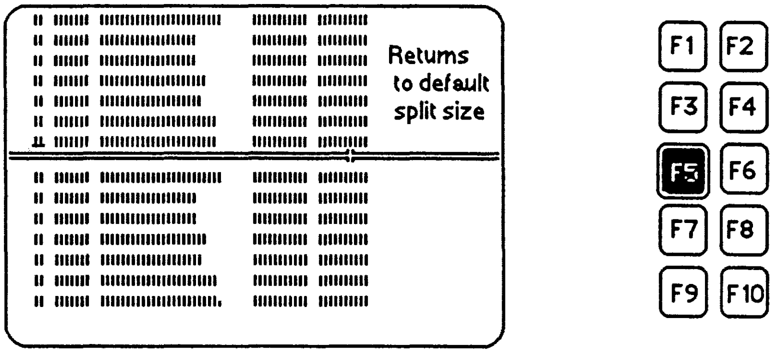


Press the **PgDn** key to see the trace listing presented in the lower window only:



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, tap **F5** and the window split will return to the 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 **Shift-F8** again to change the split size again.

Viewing 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 allows you to view source code in a convenient manner, so you don't have to print out lots of listings, or leave the UniLab program, enter a text editor, and come back again.

This is done with the **TEXTFILE** command. **TEXTFILE** always uses a split window. The text always appears in the top window. If you have a small text file (source code for your cross-assembler or a long .BAT file), open it in the UniLab environment by typing

TEXTFILE <name>

where <name> is the full pathname of your file. If it's called **MYFILE** and is on drive B, then specify **B:MYFILE** just like you would in DOS.

This should be a DOS textfile, not a file that is to be used by a text editor with embedded formatting commands. Most text editors have the capability of saving files as text only, or as "non-document" files.

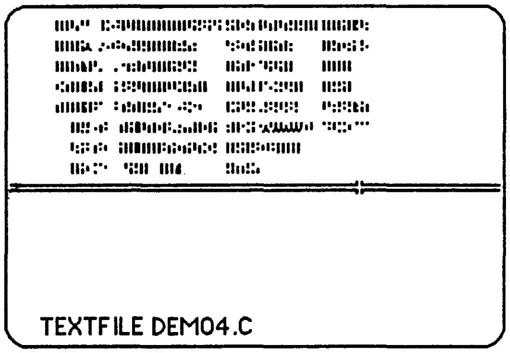
If you try to use **TEXTFILE** to open a file that is not text, you will get the " Not a DOS TextFile" message.

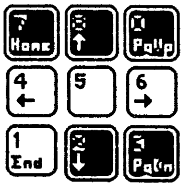
After a few seconds (or more if it's a large file), you'll see the first few lines of the text file in the top window.

Cursor key assignments when viewing a text file

PgDn	shows the next screen full of the textfile
Down Arrow	shows the next line of the textfile
PgUp	shows the previous screen full of textfile
Up Arrow	shows the previous line in the textfile
Home	shows the top of the file

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.) 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

Cursor Key Summary

There are no specific keys or commands to change the way the cursor keys work-- their purpose changes as the task you are working on changes.

The **PgDn** key, for example, can sometimes be used to move the trace display and other times can give you more **WORDS** or scroll down in a text file. However, there should be little confusion as to which function is currently active.

Viewing the trace buffer

Mainly you use the cursor keys to view the trace buffer. If you ever get into a mode where you want **PgDn** to go down in the trace buffer rather than show more **WORDS** or more of a glossary definition, press the **Down Arrow** key. This will get you back into full control of the trace with the cursor keys.

If you are in a window viewing a text file, and want to see the trace buffer, you must first enter a trace view command such as **TR** or <cy#> **TN**. After that the cursor keys will again work with the trace.

Other uses

Anytime you use the analyzer in a window where you are looking at a text file, the cursor keys will be re-assigned to work with the trace display. The analyzer is usually started with a command like **STARTUP**, <trig spec> **S**, or <adr> **AS**. There is then no way to go back to the text file-- you have to type in the command **TEXTFILE** to gather the file in again.

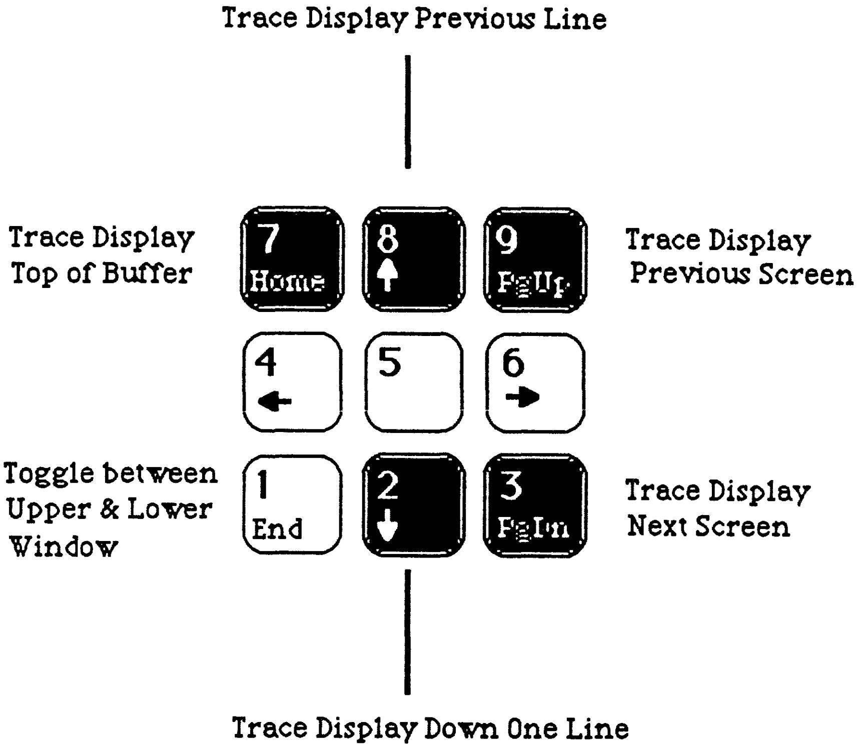
You can, however, keep a text file open in one window (usually the top is preferable), and view the trace in the other window. You can move the cursor back and forth with the **End** function key, and the cursor keys will keep the appropriate function for the window you are in

The cursor keys have special uses in the setting of the window size, and in the use of the pop-up panels. When in either of these modes, no other keys will be effective. These special modes should introduce no confusion.

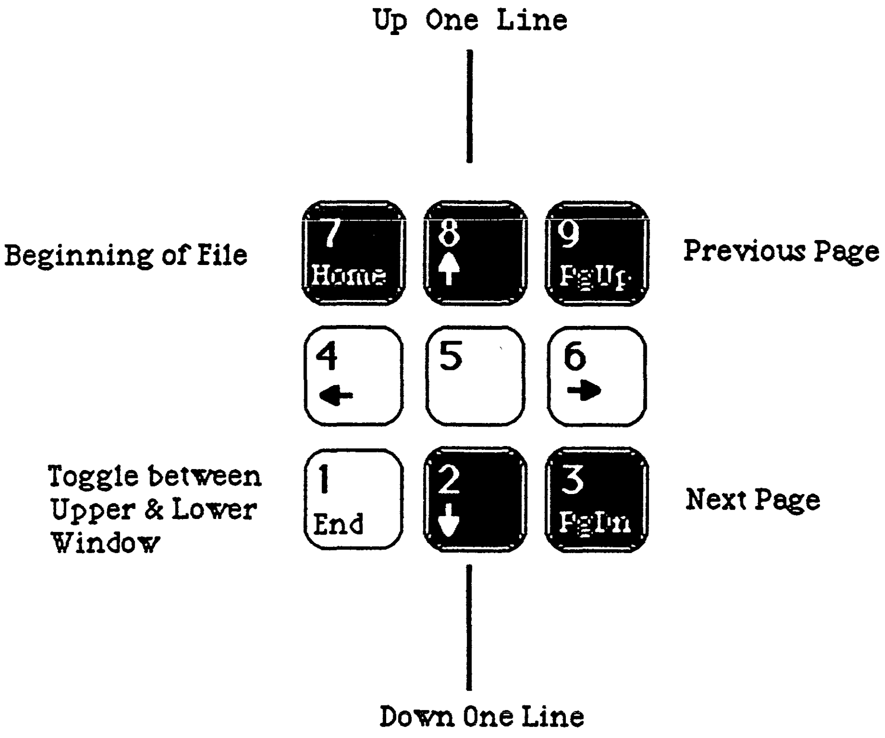
Cursor Key Chart

The next two pages summarize the uses of the cursor keys in various different contexts.

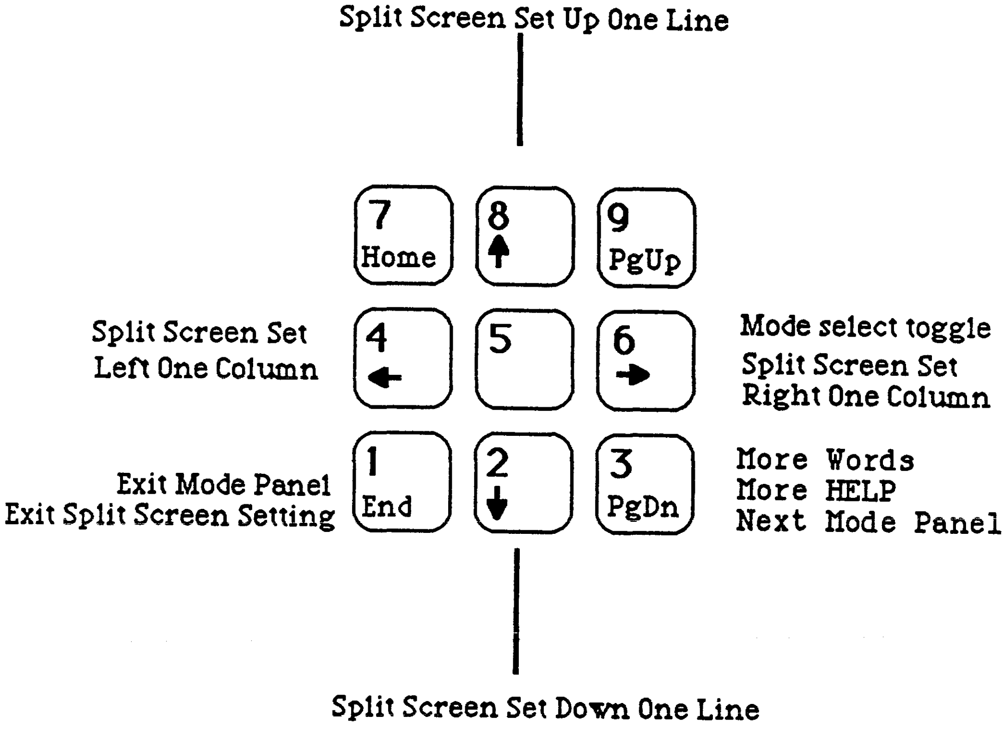
Cursor Key Assignments for Viewing Trace Buffer Display


Trace Display Previous Line
Trace Display Top of Buffer
7 Home
8
9 PgUp
Trace Display Previous Screen
4
5
6
Toggle between Upper & Lower Window
1 End
2
3 PgDn
Trace Display Next Screen
Trace Display Down One Line

Cursor Key Assignments for Viewing Text Files



Other Cursor Key Uses


Split Screen Set Up One Line
7 Home
8
9 PgUp
Split Screen Set Left One Column
4
5
6
Mode select toggle Split Screen Set Right One Column
Exit Mode Panel Exit Split Screen Setting
1 End
2
3 PgDn
More Words More HELP Next Mode Panel
Split Screen Set Down One Line

Chapter Five:
On-Line Help

Introduction

The UniLab software provides you with extensive on-line help. The help facilities give you the assistance you need to avoid confusion and to solve problems.

The Menu system, demonstrated in chapter three and fully mapped out in chapter four, gives you help while you gain familiarity with the instrument. The menus help you get to work with the UniLab right away.

The command **MESSAGE** will give you information on the most recent updates and additions.

Other On-Line Help includes:

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

Contents

1. Command Reference

The on-line version of the command reference includes the definition of the commands and features in the UniLab software. Type in

HELP <command>

to get the information on your screen.

To use the **HELP** feature, you must have the DOS variable **GLOSSARY** properly set, 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.

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.
          Also a heavy-handed way to push a byte into memory.
          See also MM, M, MM!, and M!, for more elegant ways to
          manipulate memory.
          Note that the <from> and <to> addresses must be in the
          same 32K block.
     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.

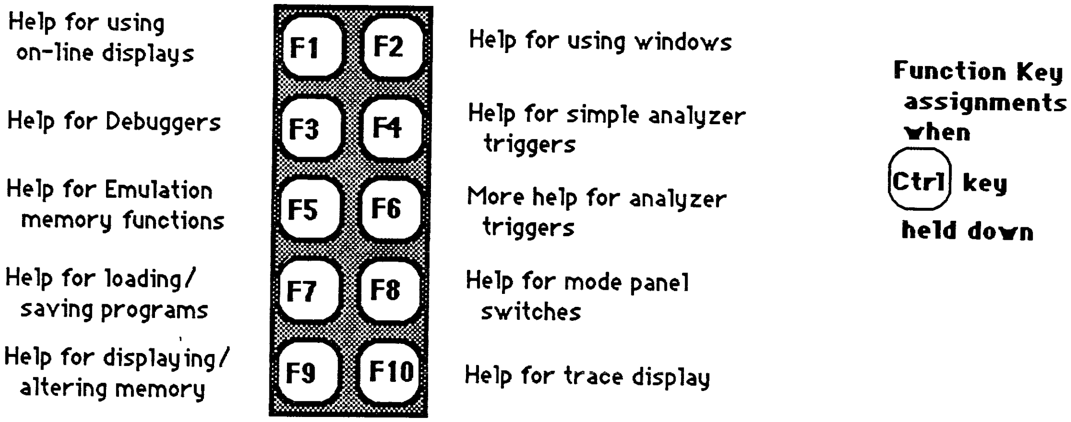
Altogether, you can have forty features assigned to the function keys. Each function key can be assigned four commands:

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.


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 Ctrl key held down

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	Ascii display - Shows ascii values for keys.
----	F5	F6	----
----	F7	F8	Set new window split size
----	F9	F10	----

Function Key assignments when Alt key held down

List Function Key assignments for Alt	F1	F2	----
----	F3	F4	----
----	F5	F6	----
----	F7	F8	----
----	F9	F10	----

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 NMI pulse to target to get breakpoint.
Restore window split to Default sizes.	F5	F6	Single Step - Execute next instruction. Will follow jumps and branches. May be same as NMI.
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.

5. Help for the Mode Panels

You use the mode panels to easily turn features on and off.

On-line help for the Mode Panels makes them as easy to understand as they are to use.

Hit **F8** or type **MODE** to get into the mode panels.

While in a mode panel hit **F1** to get the help display for the current option.

The display that you get for each option includes the name of the command that the mode panel replaces. Type in

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 MISC COLUMN 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 CONT COLUMN 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 MISC #BASE 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 PAGINATE 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 FIXED HEADER 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 debugger 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 hit **F1**) gives you general information on commands that give you help with the UniLab software.

If you hit 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

The 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.
1=HELP F2=SPLIT F3=N F4=NMI F5=DEF F6=SSTEP F7=TSTAT F8=MODE F9=STARTUP F10=ME
```

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
     Debugger 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**

Help for Windows

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: Debugger Commands **CTRL-F3**

Note that this help screen includes information on the processor specific debugger that you are using.

Help for The Debugger

You use the debugger 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 **Establish Debug Control** with **NMI** (not supported on all processors) 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. You use **SSTEP** (available on your processor if **NMI** is) to follow jumps and branches.
All commands for reading and changing memory work on RAM when stopped at a breakpoint. **CTRL-F9** tells you more about memory.

[PROCESSOR SPECIFIC INFORMATION FOR Z80]

m n OUT writes m to port n. **n INP** reads port n.
EINT reenables 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 **NORMx** 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> fo 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 debugger, 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, hit **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 (hit **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.

READ THIS FIRST

IS SHEET REPLACES CABLE WIRING INSTRUCTIONS OF THE INSTALLATION CHAPTER

Connecting the UniLab to your 8031 target board with an In-Place (tm) Emulation Module

Introduction

The In-Place (tm) Emulation Module is the new way to connect the UniLab to your target board: you simply remove the microprocessor from your board and put the module in its place.

Please read all instructions before attempting to install your new module, as incorrect procedures may cause damage to the UniLab.

Power Supply

The module can even supply power to your target, if your board takes 5 Vdc power and draws less than 1 amp.

Power is normally provided to the 8031 from the target system, but, the UniLab can also provide power to the 8031 and the target system by placing a jumper on pins 11 & 12 of the 12-pin jumper (refer to diagram on EM31.06).

WARNING: If your target board requires more than 1 amp of power,or does not take TTL voltage, then you will need a separate power supply. When you use your own power supply you must keep the +5V jumper disconnected-- otherwise damage to the UniLab may result.

INTERNAL/EXTERNAL RAM: If your board has external RAM, the UniLab normally uses RD7 (pin 17) and WR6 (pin 16) of the 8031 as inputs to clock external RAM reads and writes. If you are not using external RAM and are using these pins for some other function, these UniLab clock inputs can be disabled by changing the jumpers on the 12-pin jumper header on the Emulation Module: The 8031 Emulation Module normally comes with these inputs hooked up, with a jumper on pins 5 & 6 and a jumper on pins 9 & 10 for WR6 and RD7 respectively. To DISABLE the UniLab inputs, REMOVE the jumper on pins 5 & 6 and place it on pins 6 & 8; and REMOVE the jumper on pins 9 & 10 and place it on pins 7 & 9.

SPECIAL FEATURE: The UniLab is also able to generate an interrupt on INTO (pin 12) of the 8031. The 8031 Emulation Module normally comes with this capability disabled. By REMOVING the jumper on pins 1 & 2 and placing it at pins 1 & 3 of the 12-pin jumper header, you will be able to interrupt the 8031 from your target board OR the UniLab.

The Three Step Connection Process:

1) Remove microprocessor from board

With the module connection, the UniLab still runs all the target code on your microprocessor. The module just moves your processor a centimeter or two away from the board.

2) Plug module into microprocessor socket

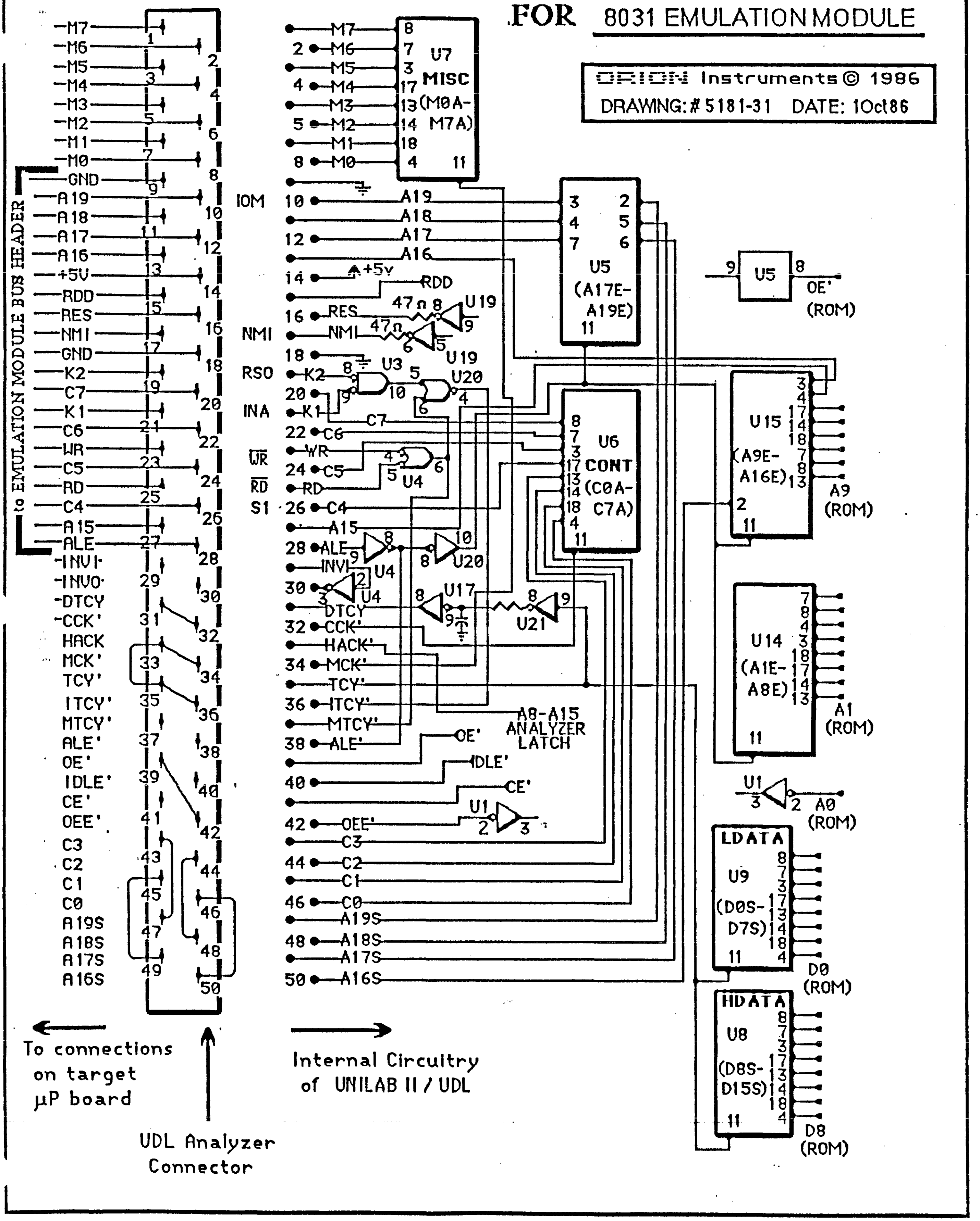
Put the module into the microprocessor socket on the target board. Double check that you have oriented the module correctly, so that pin one of the processor on the module lines up with pin one of the processor socket.

3) Plug cables into UniLab sockets

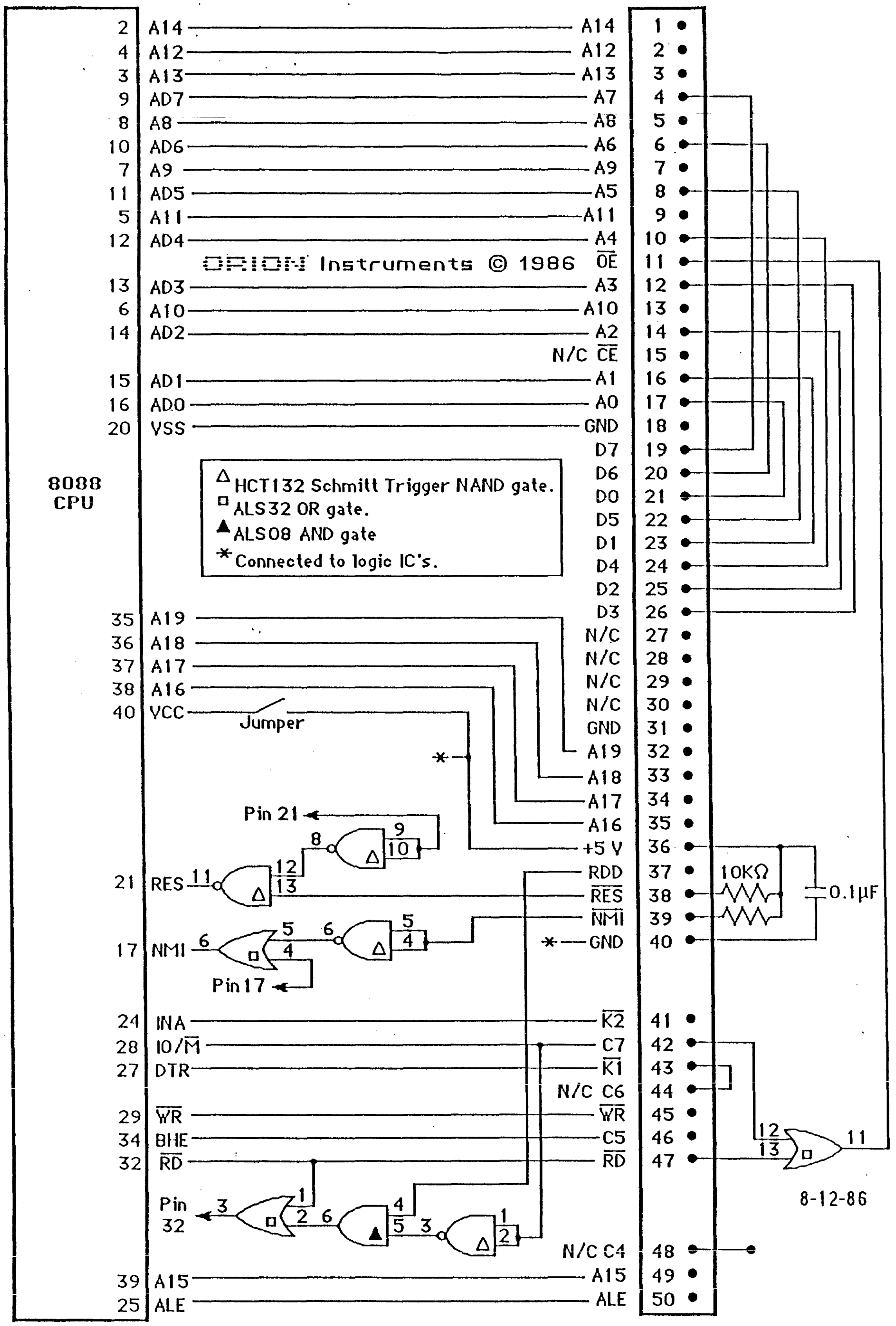
Plug the 50-pin connector labeled "Emulator" into the left socket on the UniLab, plug the "Analyzer" connector into the right socket. Both connectors must be plugged in with the plastic "key" on the upper surface, and the red edge of the cable to the left.

ANALYZER CABLE CONFIGURATION
FOR 8031 EMULATION MODULE

ORION Instruments © 1986
DRAWING: # 5181-31 DATE: 1Oct86

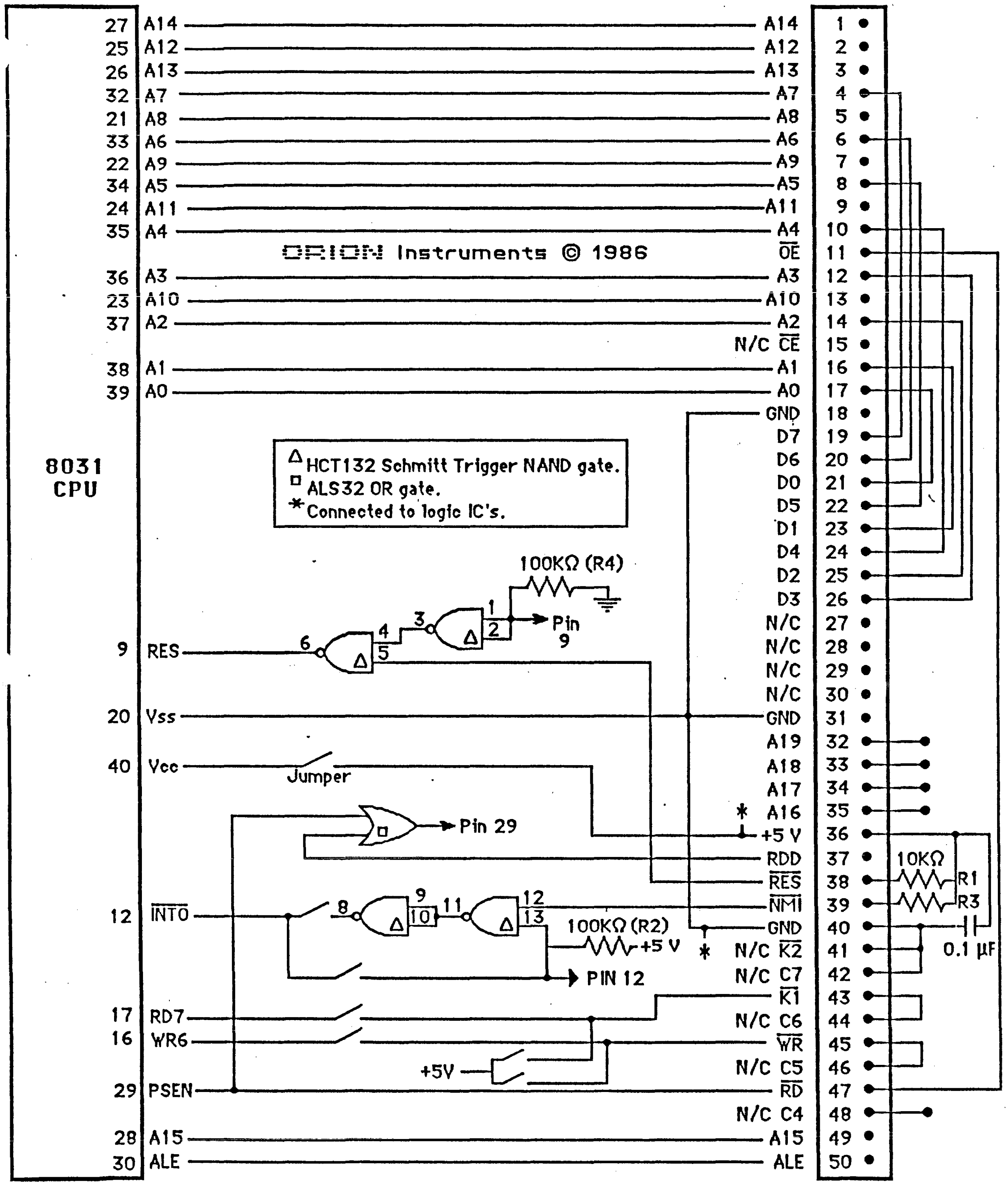


8088 MIN EMULATION MODULE SCHEMATIC

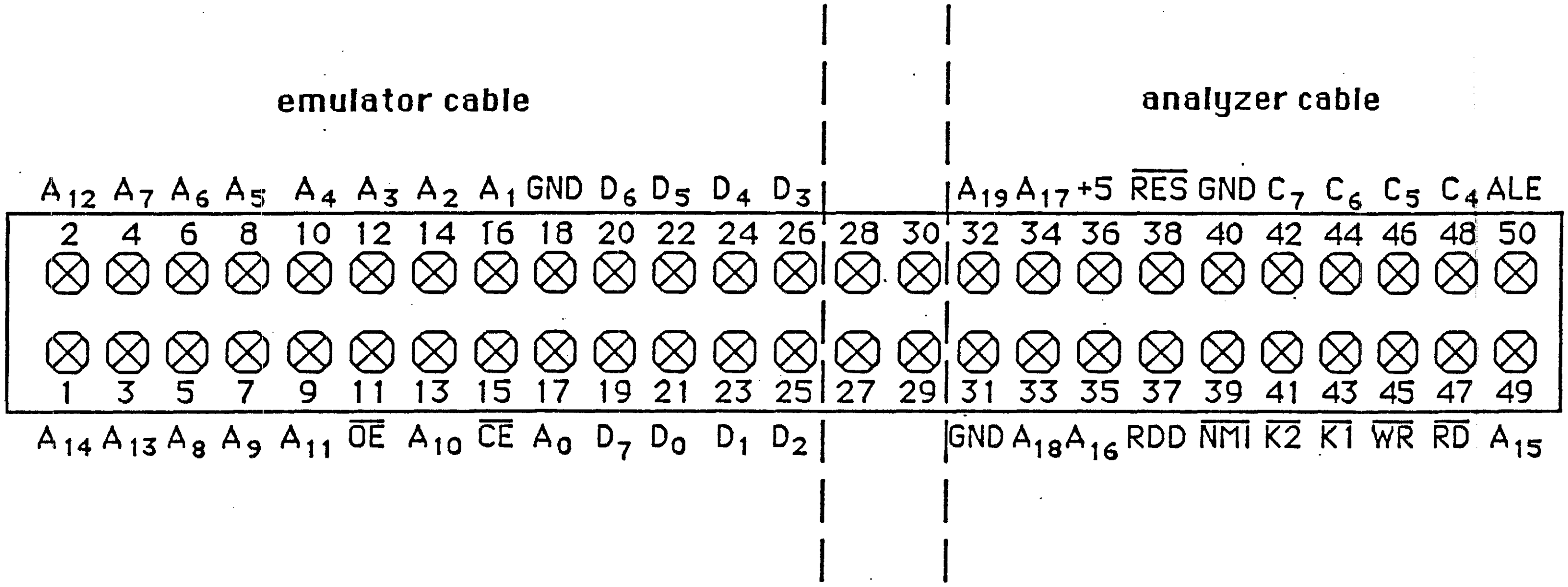


Title: EM88MN Schematic | Rev: A
DWG#: 5125 | drawn by: PAB | appv'd:
DATE: 12Aug86 | ORION INSTRUMENTS, INC

8031 EMULATION MODULE SCHEMATIC



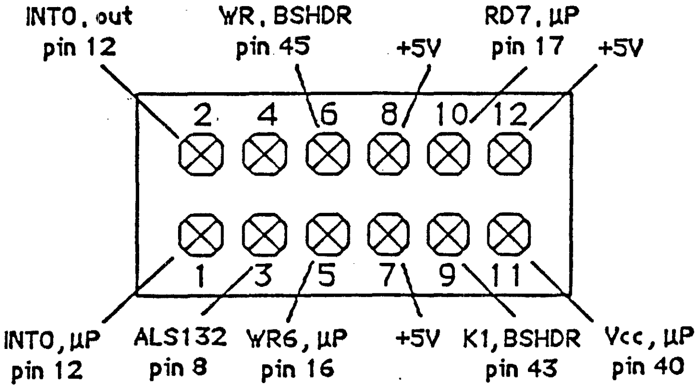
Title: EM31 Schematic		Rev: A
DWG#: 5128	drawn by: PAB	appv'd:
DATE: 1Oct86	ORION INSTRUMENTS, INC	



A_{12}	A_7	A_6	A_5	A_4	A_3	A_2	A_1	GND	D_6	D_5	D_4	D_3			A_{19}	A_{17}	+5	$\overline{RES}$	GND	C_7	C_6	C_5	C_4	ALE
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49
A_{14}	A_{13}	A_8	A_9	A_{11}	$\overline{OE}$	A_{10}	$\overline{CE}$	A_0	D_7	D_0	D_1	D_2			GND	A_{18}	A_{16}	RDD	$\overline{NMI}$	$\overline{K2}$	$\overline{K1}$	$\overline{WR}$	$\overline{RD}$	A_{15}

50-PIN EMULATION MODULE BUS HEADER

ORION Instruments © 1986



12-PIN JUMPER HEADER

Title: 50-pin EM BUS HEADER		Rev: A
DWG#: 5180A	drawn by: PAB	appv'd:
DATE:12Aug86	ORION INSTRUMENTS, INC	

ORION

User Updates from Orion

This Newsletter is designed to provide Orion customers with important update information about the use of their Orion purchase. It contains product facts from our engineering and marketing staff as well as contributions and user-hints from our customers.

We invite you to share your personal user "discoveries" and techniques that might be passed along to other Orion customers involved in microprocessor development. Send your material to the attention of Joyce Waterhouse, the *Orion Express* Editor. Naturally, we'll give you credit for your contributions.

Target Hardware Debugging with your UniLab

Orion's UniLab benefits go far beyond conventional in-circuit emulators, and are particularly useful in situations where the target processor system is not yet working.

A conventional emulator depends on the emulation processor to send traces back to the host system; therefore, if the target processor is disabled for any reason (such as a bus short), then you won't be able to look at register or trace memory displays. As a result, an oscilloscope is the only tool that can be used to fix the system.

Only a Microprocessor Clock Needed

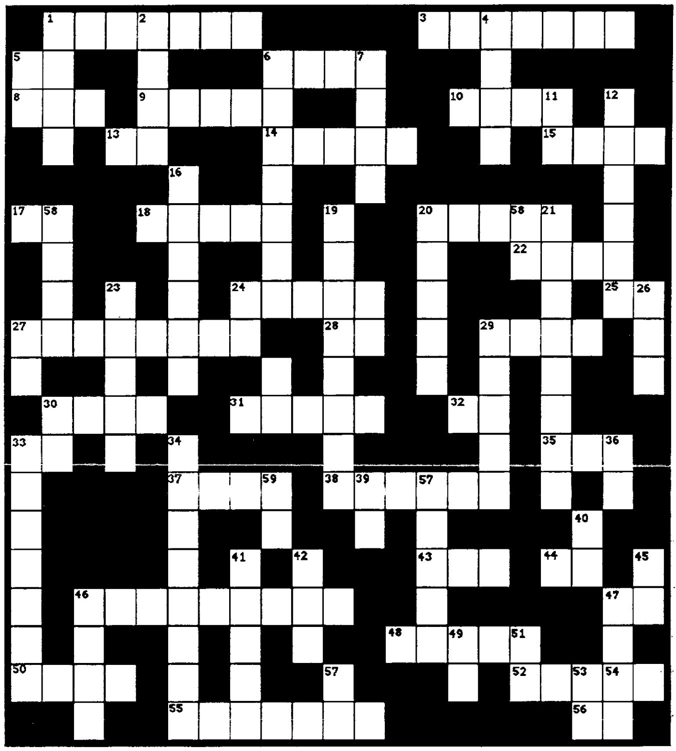
Since the Orion UniLab has its own independent hardware for handling traces and emulation memory loading, it uses the target processor *only* for debug operations. For UniLab to function, all that is needed is for the target processor to have a clock. If a clock is not available, UniLab cannot tell what is happening on the bus. If this is the case, a "no target clock" message will be displayed. An oscilloscope then becomes your only alternative.

Locating Bus Problems

If a clock is present in the target processor, either UniLab's ROM emulation or its bus state analyzer can be used to deduce the problem. For example, to find address line shorts, use the MFILL command to fill the beginning of the target system's memory with the "no-op" instruction opcode.

(continued on page 2)

CrossLab



All words are in the UniLab™ Glossary.

Across

1. Key to Library
3. You can see all the symbols with this word
5. Word used to indicate range
6. Display symbols
8. Display trace, but don't change default starting point
9. Select automatic initialization of target when analyzer starts.
10. Establish communications with UniLab and Host
13. Display byte contents of memory
14. Move up to next qualifying event level
15. Turn on filter to capture only selected bus cycles
17. Macro for conditional execution, used with THEN
18. From a new beginning, please
20. Check time optimization
22. Trigger spec used to focus on ram variable values
24. Multiplexed display
25. Throw filter out
27. Symbolic append
28. Can you wait just a thousandth?
29. It's all this, but this is essential
30. Do it at BP
31. Take this byte and shove it
32. A name ____ a name ____ a number
33. Double print
35. Won't go on fire
37. Categorical additive
38. Miscellaneous is extraneous
43. Restore startup patch
44. Needs a place to start, a place to stop
46. Adjust tints
47. Two bites, no chocolate inside
48. What's happenin', analyzer?
50. What it is, target?
52. Just the points of interest are tested
55. Format translator

Down

1. Can't tell a read from a write without this
2. Where are you, program?
4. Waiter, selections please
5. Once more, from the top
6. "____ me ____, and I'll never stop"
7. Same as 30 across
11. Set span
12. Compliments of the host
16. Dos, take a memo
19. The pulse that invigorates
20. Don't forget the lines – Where did we fail?
21. May the source be with you
23. Elbow room?
24. What's next?
26. Poking point
27. Startus Interruptus
29. Don't disassemble
30. Next is ones and zeroes
33. A0, 55, and 4 are normal
34. Make Macros permanent
36. On you mark, get ready… wait
40. Resume to break
41. The middle point, before and after
42. Opposite
45. Abbreviation for UniLab storage
46. Use symbols
47. Can't filter on this
49. Upload trace buffer
51. Rerun trace
53. NORMT ADR S
54. Impatient tester
57. Around and around
58. For extensible keyboard
59. UniLab acknowledge
60. Byte to the max

(Answers to Puzzle Inside)

Target Hardware Debugging

(continued from page 1)

If a Z80 is used, for example, fill the memory with 0's. At **STARTUP**, the very first cycle is the most interesting one. A trace will be seen of the first instructions being executed. This means that the emulation ROM, when Address 0 is fetched, gives a 0 on the data bus.

If the first instruction is not fetched correctly, ignore the rest of the trace and work mainly with that first instruction. If there is a short to Plus on one of the data lines, you will get, for example, 04 instead of 0. This would indicate that data bit D2 is shorted high. Experiment by inserting different values in location 0 and initiating **STARTUP**. Observe what happens on the bus when location 0 is fetched.

If 0 is fetched properly, the NOP instruction is executed at successive addresses. A trace shows a 0 being fetched from Location 0, then another 0 from Location 1, and another from Location 2, etc., in a conventional counter sequence. A break in that sequence indicates one of the address lines is shorted.

Here's an example: if address line 4 is shorted, you will count 0 through F hex address, but when the address line should be going to hex address 10, the trace will show a 0 instead. This is because the line which has to go high to make it address 10 is shorted to ground. Note that if address bit 4 had been shorted to Plus, the very first cycle of the **STARTUP** trace would have been 10 instead of 0.

Solutions by Deduction

The process of tracking down bus shorts is a matter of deduction. By putting different values into the emulation ROM and observing the trace, you can track down the problem. If the counting starts out correctly, the triggers can be set at later points in the count. Looking at the **STARTUP** trace works well for the least significant part of the address, however, higher bits can be checked quickly using the AS command to trigger on the address of interest. For example, **1000 AS** will show the trace as it passes from FFF to 1000 to confirm that all is functioning correctly.

It is important not to look past the first bad instruction. Quite often the first bad instruction will be the very first instruction. If address 0 cannot be fetched it may be because of a high short. Another subtle type of bus short occurs between lines–either between one data line and another, or between address lines. This type of problem is more difficult to deduce.

Let's say, if data bits D0 and D1 were shorted together, then operation would appear normal when the memory is filled with 0's. But, if a 1 were fetched, it may come out as a 3. Or, because of the short, it may come out as a 0. The problem can be deduced by trying **STARTUP** with various values stored in location 0 and observing what gets fetched (ignore all but the very *first cycle*).

The principle holds true with address shorts. Watch the address sequence. As soon as there is an unusual occurrence, some type of a short or open is generally indicated. It may be necessary to look at different addresses to get the complete picture–for example, to determine if one address bus is shorted to another. Jump commands can be put in memory to exercise the address bus.

Testing for Malfunctions

Basic system malfunctions can also be tested. For example, if debug has not established control, and if there is a RAM in the circuit, a short program can be devised using the on-line assembler that writes to location in RAM and then reads from that location. You will see data being written on the analyzer trace, and then read back. If the read-back data doesn't agree with what's being written, a RAM malfunction is indicated.

User Definable Macros

The macro capabilities of the UniLab are a great help in automating deductive testing. For example, if you are having trouble fetching the first byte, a macro can be made which enters a value to location 0, then does **STARTUP** every time "P" is entered, like this:

```
: P  0 M! STARTUP ;
```

It is also possible to effectively use DOS calls in a macro. The following example sets up the macro name DOS-DIR, which allows you to display the DOS directory when invoked from UniLab:

```
: DOS-DIR " DIR \ ORION \*.* " <DOS> ;
```

To implement other DOS calls in this manner, simply place the desired DOS call within the quotation marks and change the macro name.

Another example:
To format a disk on drive B, set up the macro name FORMATB like this:

```
: FORMATB " FORMAT B:\S " <DOS> ;
```

Most of the things that debug can do can be implemented with short macros. Debug will function only if the program execution in the target system is working, and generally the stack must also be functioning properly. Stack operation can be determined by writing a PUSH instruction followed by a POP instruction and observing the trace.

In Summary

The debug capabilities of the UniLab, of course, require that the target system be functioning properly. However, the basic ROM emulation and analyzer functions are independent of the target system. By using the UniLab tools and your powers of deduction, you should be able to find, quickly, most types of problems in the target system.

Program Performance Analysis Option

A new analysis package from Orion is now available, affording you all the tools you need in a single instrument for beginning-to-end development and optimization of microprocessor code.

UniLab II users can now add comprehensive Program Performance Analysis (PPA) capability to their present system. Program Performance Analysis is used to evaluate and investigate any section of code, from a single byte to a complete program. UniLab hardware tallies every program event of interest in real time at full speed, and the results are automatically displayed in both tabular form and as bar graph histograms.

Address displays show both the number and percentage of accesses to user-specified address ranges, or to named subroutines. To evaluate the performance of particular subroutines, time domain analysis precisely measures elapsed time between entry and exit points. The combination of these techniques eliminates sampling errors associated with statistical approaches.

Invisible bugs and inefficient code are quickly uncovered by comparing PPA-generated data with expected program performance. Complete screen set-ups for recording and displaying data as well as results can be saved to disk or printed out for future reference, or for comparison with other runs. The PPA upgrade for UniLab is available at a list price of $1250.

UDL users are not forgotten. Orion's UniUp package, selling for $1985, will upgrade your system to Unilab performance. And for even greater capability, the Program Performance Analysis upgrade package can also be added.

Support Services Options to extend your warranty are available for $700, and will cover *both* your system and any upgrades.

Orion's Rental and Lease/Purchase Plans

Rental and lease/purchase plans are available for all Orion products. These plans have been devised to further augment our policy of full customer service.

The rental plan assists the customer who has only a short term requirement for a microprocessor development system, or wishes an extended trial before he buys. The lease/purchase plan is ideal for the customer who requires a longer-term rental or an extended payment plan.

	Rental Plan	**Lease/Purchase Plan**
Term	3 Month Minimum	12 months
Renewable	Month-to-Month after 1st 3 Months	Yes, for an additional 12 Months
Purchase	50% of rent paid applies up to 50% of purchase price	1 additional payment of 11% of total configuration price after 12 Month lease
Rate	10% per month, payable in advance. Initial 3 months payable in advance	11% per month, payable in advance. Ist and last month's payment required at signing of lease
Credit	Approval Required (or automatic bill with credit card)	Approval Required (or automatic bill with credit card)
Taxable in CA	Yes	Yes
Warranty	Included. Return to Orion. Cables: 90 day warranty only	Included. Return to Orion. Cables: 90 day warranty only
Application Engineering Support	Included	Included
Updates	Included	Included
Support Services Option	$500/Year (After Purchase)	30% Discount ($350 Net for UniLab) for 12 Months after Purchase

Ask Dr. D.

Questions from Orion Customers

Q. I'm experiencing erratic trace displays with my UDL (high-speed model). Any ideas? Signed A.E.

A. Dear A.E.,
Recently we have seen a few problems apparently caused by degradation of parts used in the high-speed models. Standard speed units are not affected. Call our Applications Engineering Group at (415) 361-8883 and they'll get the problem solved. Signed Dr. D.

CrossLab Solution

How did you do?

	C	A	T	A	L	O	G						S	Y	M	L	I	S	T	
T	O			D				S	Y	M	B				E					
T	N	T		R	E	S	E	T			P			I	N	I	T		M	
	T		M	?				A	F	T	E	R			U		O	N	L	Y
					T			R			X								O	
I	F			N	O	R	M	T		S			T	H	I	S	T		A	
	K				F			U		T			C			R	E	A	D	
	E		?		I		S	P	L	I	T		O				X		N	O
S	Y	M	F	I	L	E	+			M	S		M		D	A	T	A		R
I			R		E					U			P		A		F			G
	B	P	E	X			M	F	I	L	L			I	S		I			
D	#		E		S					U					M		L	O	G	
C					A	L	S	O		S	H	O	W	M	'		E		W	
Y					Y						O		O					R		
C					E		N		N				R	S	P		G	B		E
L		S	E	T	-	C	O	L	O	R			D						M	M
E		Y			S		R		T			T	S	T	A	T			I	
S	A	M	P		Y		M			L				D		T	M	A	S	K
		B			S	Y	M	T	Y	P	E							S	C	

Put a UniLab or UDL on Your VAX or PDP-11

Users of Digital Equipment Corp. (VAX and PDP-11) can now connect UniLab or UDL instruments directly to their DEC computers. The DEC-compatible software has been developed by Compu-Mech, Inc., one of Orion's long-time distributors. For more information, contact Compu-Mech at 5242 Angola Road, Toledo, Ohio 43615. Telephone (419) 535-6702.

WESCON OptiLab Winner

Visitors to Orion's booth at the recent WESCON show in Anaheim were entered in a drawing to win a complete OptiLab work station, including an ACS Turbo XT computer. The OptiLab product includes program performance analysis capability along with the integrated instrument set for microprocessor development. The winning entry was submitted by Mr. Keith Vennel, Director of Engineering at Micro Concepts Corporation, 1630 South Sunkist Street, Anaheim, CA 92806.

Terry Zimmerman, Orion's Vice President of Marketing, telephoned Keith with the announcement shortly after the close of the show. It appears Keith's good fortune was timely since he tells us that Micro Concepts builds stand-alone, in-circuit testers and currently is involved in projects using 68000 and NSC 800 microprocessors. Both are supported by the Orion OptiLab toolbox.

Our congratulations to Keith Vennel of Micro Concepts.

ORION
Instruments

Orion Sales Hotline: (800) 245-8500
(in California (415) 361-8883)

ORION
Instruments

News Briefs

- UniLab now supports these symbol file formats: 2500AD; Allen Ashley; Intel; Microtek; Manx Aztec C; Avocet; and a generic fixed format.
- New disassembler debug packages now available for Zilog Super 8, NEC 78310/12, and Motorola 68HC11 microprocessors.
- The 6305 microprocessor is now supported by the 6805 disassembler debug package.
- Movable overlays and line-by-line assemblers are now included in all disassembler debug packages (except Z8000).

ORION

ORION
Instruments

702 Marshall Street
Redwood City, CA 94063

Address Correction Requested

First Class Mail

Newsletter for Users of Orion UniLab and UDL

ORION

Orion Express Rolls On

This Newsletter is designed to provide Orion customers with important and useful information to help you get the most from your Orion purchase. It contains product facts from our engineering and marketing staff as well as contributions from our customers.

We invite you to share your experiences, questions, and "discoveries" with other Orion users through this Newsletter. Please mail your material to the attention of Editor, *Orion Express.* Naturally, we'll give you credit for any contributions. **ORION**

UniLab Macros for the Power User

The UniLab command set can be easily extended by creating custom commands specific to your needs. The basic techniques are to use combinations of commands to form Macros, and to use PC-DOS's "command tail" feature to pass these macros to the UniLab from the keyboard or from a DOS batch file.

For example, if you switch back and forth often between the UniLab and a cross-assembler to make progressive changes in your code, you may have found yourself entering commands like this each time you re-enter the UniLab system:

0 7FF BINLOAD TARGET.BIN (load new binary file from cross assembler)

SYMFILE TARGET.SYM (load new symbol table from cross assembler)

STARTUP (Start the target system up from reset)

42F RB (get a breakpoint at address 042F, reset still on from **STARTUP**)

NMI (single step)

NMI (single step again)

There are a couple of ways you can automate much of this procedure.

Automate your procedures

If you wanted to automate only the file loading part, you could make a macro called **LOAD-FILES** like this:

```
:  LOAD-FILES  0  7FF  BINLOAD  TARGET.BIN
SYMFILE  TARGET.SYM  ;
```

Now, every time you want to load in the binary file and symbol file with these names you just type **LOAD-FILES.**

Suppose you change the name of the binary file for different versions. You can construct another macro a little differently to make it prompt you for the file name when you enter **LOAD-FILES1**:

```
:  LOAD-FILES1  0  7FF  [COMPILE]  BINLOAD
SYMFILE  TARGET.SYM  ;
```

Using the command **[COMPILE]** immediately before **BINLOAD** causes **BINLOAD** to ask for the file name when the macro is executed, rather than when the macro is defined (compiled). You could also have preceded **SYMFILE** with **[COMPILE]** and then a macro called **LOAD-FILES2** could prompt you for both file names.

More time savers

The second part of your procedures could also be automated by a macro command. For purposes of this demonstration, let's define a macro and name it **RUN.**

```
  :  RUN  STARTUP  42F  RB  NMI  NMI  ;
```

Now, you can just type **LOAD-FILES RUN** to do everything.

One point: If you wanted to set the breakpoint at another address, you would have to retype in all of the commands that are in the **RUN** macro separately. If **RUN** were defined *without* the 42F address, then you could supply the address as a parameter to the macro when it was executed, like this:

```
  :  RUN+  STARTUP  RB  NMI  NMI  ;
```

This allows you to execute **42F RUN+** or **174 RUN+** , or any address you want.

Putting these two together produces a macro which can save you a lot of keystrokes:

```
  :  LRUN  LOAD-FILES  RUN+
```

(continued on page 2)

(continued from page 1)

Create your own custom UniLab

Now, if you save your customized UniLab system to disk with **SAVE-SYS**, your new commands will be permanently added to the regular command set of your system. You can get rid of them by typing **FORGET LOAD-FILES.** (Using **FORGET** will delete every new macro made since **LOAD-FILES** was defined.)

There is another technique which can take the place of some of these commands, or can be used to extend them even further.

If you had defined these commands and saved the system, you could enter the system and execute the commands immediately by entering from DOS:

```
C>ULZ80  42F  LRUN
```

Now this "command tail" will be executed by the UniLab after it boots up.

You also could have entered the comands separately:

```
C>ULZ80  0  7FF  BINLOAD  TARGET.BIN
   SYMFILE  TARGET.SYM  482  RUN+
```

(Note: Type as one line on screen or in batch file.)

Using the command tail lets you make very useful batch files. You may already have a batch file to use with your cross-assembler. By adding an additional line like the one above, you can quickly get from DOS into your editor, then to your cross-assembler to assemble a new file, and finally to the UniLab, executing commands automatically. A real time saver.

Now, if you end the command tail with **BYE**, then you would automatically exit from the UniLab, and be back in DOS, or the next line of your executing batch file.

The power of DOS and the UniLab macro capability gives you great flexibility in setting up your own development environment. So have fun and let us know what especially useful macros you create. See the article below for advanced programming information. **ORION**

New Programmer's Guide

A comprehensive new UniLab Programmer's Guide is available for advanced users of the UniLab and OptiLab systems. This 8½ by 11-inch, 45-page document enables programmers familiar with the Forth programming language to write advanced macros utilizing the UniLab's operating system. Differences from the PADS Forth system are noted, and special words are explained which are not covered in the UniLab manual. Material covered:

- File and Editor Commands
- Accessing the UniLab trace buffer directly
- UniLab String Package
- Changes to the PADS nucleus

Examples of automated test routines are included, as is a source listing of the UniLab Forth nucleus. The manual, affectionately given Part Number "PROGO", is priced at U.S. $45.00, including airmail postage, and is available from stock. **ORION**

Ask Doctor D

Each month our product gurus select interesting questions asked by users for inclusion in this column. Maybe some of Doctor D's answers can help you too.

Q. I understand the UDL and UniLab need four overlay bytes in ROM space. I have code at your overlay area. Is there anything I can do?

A. All Disassembler/Debug (DDB) packages now have an **=OVERLAY** command to relocate the area that our DEBUG program needs. If you type **HO** or press **Ctrl-F3**, the help screen for DEBUG will show the current location for the overlay area. One common mistake in changing the overlay area, is moving up too high. The amount of space needed varies from processor to processor, but a good rule of the thumb is to keep the low byte of the overlay area the same or less than the low byte of the default overlay address.

For instance, if the overlay area is currently set to FFB2, then don't try to change it to FFC4. This might make the overlay area cross a page boundary, or try to use non-existent emulation memory. In some processors it could even overwrite the reset and interrupt vector area. It is better to move it to FEB2 or FF20, allowing the overlay area to remain on a single page of memory.

Send your questions to Doctor D. We'll publish the ones of most general interest, but if you'll include your phone number, we'll give you a personal call back. Write today! Right now!

Do You Have the Latest and Greatest?

Here's a listing of the latest revisions of popular Orion software. Check to see which revision you have by typing the command: ".DDB" while in the Disassemble/Debug (DDB) package. If you don't have the latest, read below to see how you can get updated.

PROCESSOR	LATEST REVISION
1802	October 23, 1986
6301	May 4, 1987
65P	November 12, 1987
6502	October 13, 1986
6800	December 18, 1986
6801	May 4, 1987
6802	December 18, 1986
6805	January 13, 1987
6809E	November 5, 1986
68000	January 14, 1987
68008	October 15, 1986
68HC11	May 4, 1987
8048	November 11, 1986
8051	January 29, 1987
8051P	January 29, 1987
8085	May 6, 1987
8086	February 5, 1987
8088	December 18, 1986
8096	October 15, 1986
SUPER 8	May 5, 1987
Z-80	May 4, 1987
Z-8000	October 21, 1986
78312	May 4, 1987

SYSTEM SOFTWARE	PREVIOUS RELEASE	LATEST RELEASE
UDL MSDOS	2.42	UDL2.5 (Nov. 25, 1986)
UniLab II	3.2	UNI3.3 (April 1987)
OptiLab	3.21	OPT3.3 (April 1987)

How to Obtain Latest Releases:

Of course, if you haven't purchased a particular package from Orion, call your local Orion Sales Representative or our Sales Hotline to place your order. If all you need is an update, each is available for U.S. $50.00 including airmail postage. Subscribers to Orion's Support Services Option receive updates free of charge. The Support Services Option costs just $500/year and gives you free updates plus unlimited free Applications Engineering telephone support, and free factory repair of your unit in case of failure. Ask for full details.

New Convenience and Power for Your System

Emulate with a MicroTarget™

Many customers have wanted to test their software before they actually have their own target hardware up and running. Orion's new MicroTargets allow you to do just that. The MicroTargets are completely functional circuit boards built around the most popular microprocessor types. They include RAM, Parallel I/O, and in many cases timer chips.

Even after your own target hardware is up, the MicroTargets can provide a ready means of verifying that your Orion system is functioning properly. MicroTargets can be purchased separately, and include the microprocessor itself and a schematic of the circuit. You may find that the MicroTarget is all the hardware you need for some smaller projects.

Available MicroTargets are:

63PO1 64180 6502 68PO5 68000 8031 8051P 8085 8086MIN 8086MAX 8088MIN 8088MAX Z8 Z80.

Prices range from $150 to $300.

Emulation Modules Speed Hookup

As you probably know, Orion's technique for performing emulation functions uses an actual microprocessor in your target circuit. This technique, which we call In-Place Emulation, eliminates the timing and signal errors often experienced with conventional emulators due to long cables between processor and target. The result is that Orion allows your circuit to run at full speed while you observe the performance in real-time.

Orion's new line of In-Place Emulaton Modules™ make connecting your Orion system to your target circuit easy. Just remove your microprocessor from its socket and plug-in the Emulation Module! The cabling is all done for you, and all necessary connections are automatically made. Of course, if you have a soldered-in microprocessor or extremely limited physical space where the Emulation Module won't fit, you can still connect to your circuit with ROM cables and jumpers – another advantage of using the versatile Orion approach.

Emulation Modules are available for these processors:

6303R 63PO1 64180 6502 65C02 6805E2 14-6805E2 68PO5WO 68POV07 6809E 6809 68000 80188 8031 80031 8032 8048/P 8050 8051P 87P50 80C51VS 8085 8086 (MIN/MAX/C) 8088 (MIN/MAX/C) Z8 Z80.

Prices range from $190 to $390.

Call the Orion Sales Hotline, 800-245-8500 for more information on these useful new products.

Newsletter for Users of Orion UniLab and UDL

First Class Mail

ORION
Instruments
702 Marshall Street
Redwood City, CA 94063

Address Correction Requested

ORION
Instruments

News Briefs:

- NEW and HOT! Full support for 68HC11 now available in stock! Save thousands over the competition!
- Z80 support now includes Alternate Register Display/ Alter feature. See inside for update info.
- True single stepping, breakpoint on trigger condition, and auto-breakpoint features now implemented for 6805, 8048, 8051, Z8, and SUPER 8 processors.
- NEC latest processor, the 78310/12 is now supported by Orion.

ORION

Index for Volume One

The full index can be found at the end of Volume II.