

ORION **Instruments**

UniLab II[™]

Volume Two
Reference Manual

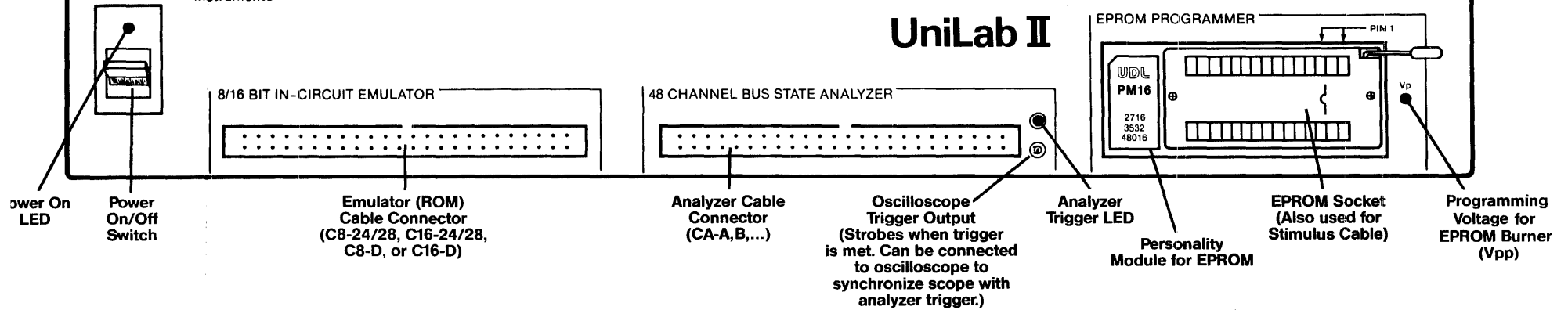
Copyright 1984, 1985, 1986 by Orion Instruments, Redwood City, California
All rights reserved

ORION
Instruments

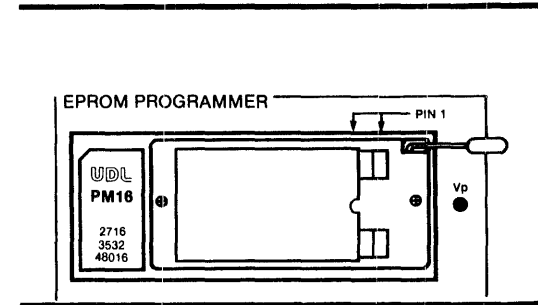
Universal Development Laboratory

UniLab II

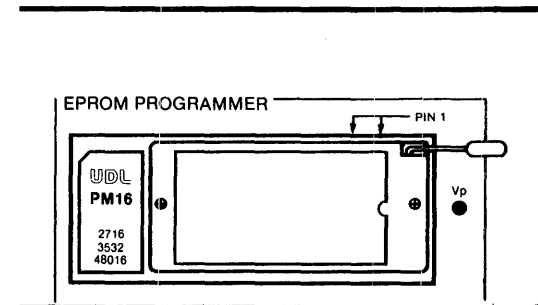
EPROM Clamp
(Down to connect
EPROM in Socket)



24-pin Package is
shifted all the way
to the left



24 Pin EPROM in Programming Socket



28 Pin EPROM in Programming Socket

UniLab Block Diagram

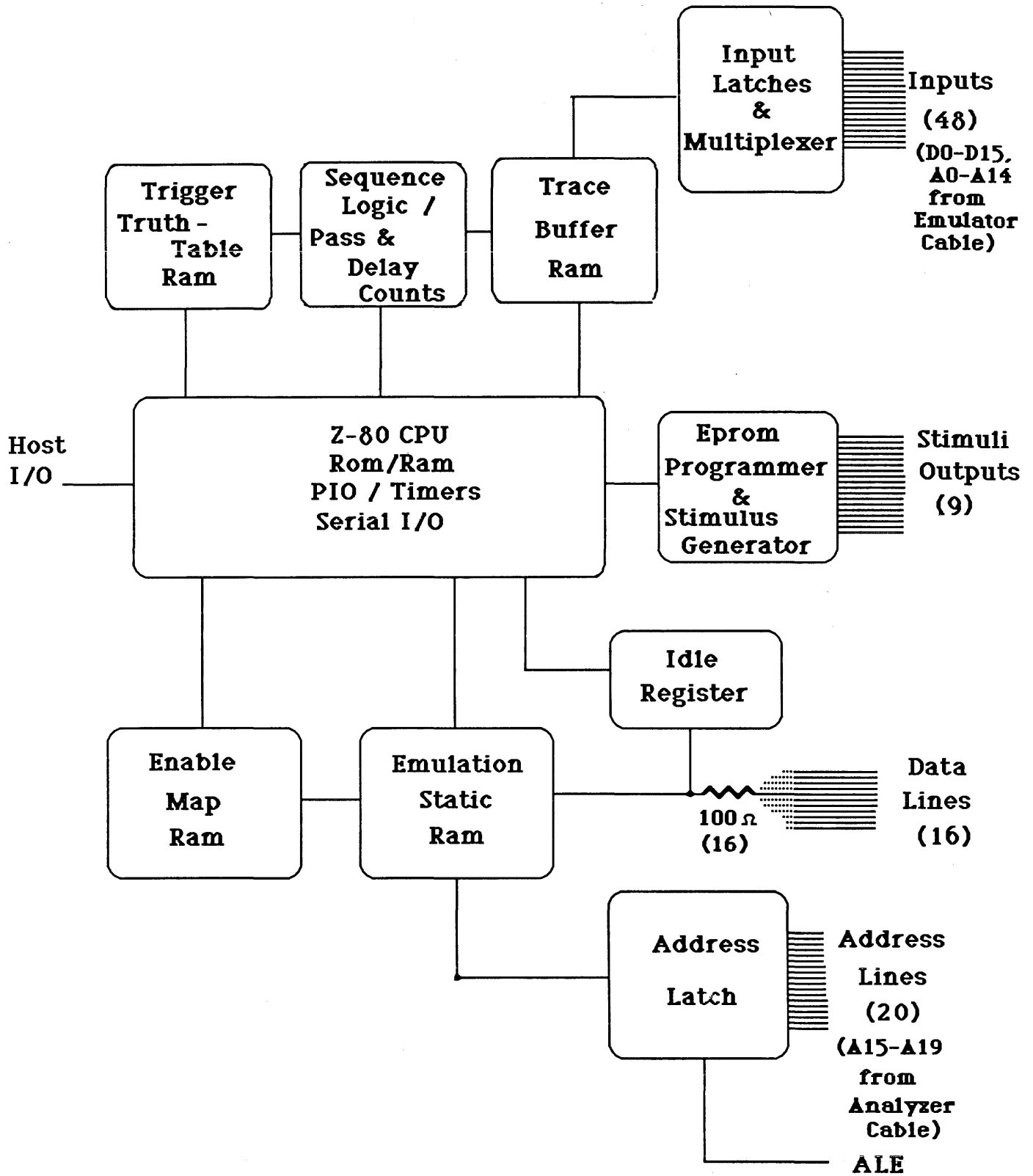


Table of Contents
UniLab Manual

VOLUME ONE
User's Guide

Chapter One: The UniLab IItm Method

Chapter Two: Installing The UniLab

For Me?	2-2
Introduction	2-3
Useful Information	2-7
Quick Step-by-Step	2-9
Detailed Step-by-Step	2-10
1. Connect the UniLab to Host	2-11
Find the Correct Port	
Serial Port of AT	
Connect the Cable	
Turn on the UniLab	
Trouble?	
2. Software Installation	2-14
Install the Software	
On a Hard Disk	
On a Floppy Disk Drive	
Reboot Your Computer	
Start Up the UniLab Program	
Patch Required?	
3. Connect the UniLab to Target Board	2-25
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	
4. Check Out Your Equipment	2-40
Load a Sample Program	
Run the Program	
Compare to Sample Trace	
Play Around a Little	
How to Exit	
Where to Go Next	2-46
Special Note: Display Characteristic Commands	2-47

UniLab is a trademark of Orion Instruments, Inc.

Chapter Three: Guided Demonstration

Overview	3-2
Call Up the Software	3-3
Get the MAIN Menu	3-4
The Five-Step Procedure:	
1. Enable Memory	3-5
2. Load a Program	3-6
3. Examine the Program	3-7
Memory Dump	
Disassemble from Memory	
4. Use the Analyzer	3-9
Get the First Cycles of Program	
Sample the Bus	
Set a Trigger on an Address	
5. Use the Debugger	3-13
Set a Breakpoint to Establish Debug Control	
Set Another Breakpoint	
Single Step Through Code	
Summary	3-16

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

Overview	4-2
1. Menu Mode	4-4
2. Command Mode	4-19
Command Tail and Batch Files	4-20
Using the Command Language	4-23
Trigger Specs: Theory and Conventions	4-24
Trigger Specifications: Examples	4-27
3. Special Features	4-33
Function Keys	4-34
Cursor Keys: Traces and Line History	4-36
Windows	4-41
Viewing Textfiles	4-49
Cursor Key Summary	4-51

Chapter Five: On-Line Help

1. Command Reference	5-2
2. Alphabetical Lookup	5-3
3. Reminders	5-4
4. Function Keys	5-5
5. Mode Panels	5-7
6. Help Screens: By Category	5-10

INDEX for volume one

VOLUME TWO
Reference Manual

Chapter Six: The UniLab in Detail

A Guide to This Chapter	6-2
1. Interpreting the Trace Display	6-5
What Each Column Means...Sample Traces...Moving through Trace...Symbolic Names...Toggling Display Options (Mode Panels)	
2. Readying and Loading Memory	6-34
Emulation ROM...Getting Ready...Loading Programs ...Saving Programs	
3. Examining and Altering Memory	6-47
Memory Access...Read...Alter...Optional Assembler	
4. Setting up a Trigger (generating a trace)	6-64
Simple Example...NORMx Words...RESETting...General Purpose Triggers...Real-life Examples...Limits... Filtered Traces...Qualifying Events...Refining	
5. Saving Information	6-90
Screen History...Log File...Printer...Trace Save... Symbol Table...Binary Image...SAVE-SYS	
6. Breakpoints and the Debugger	6-100
Establish Debug Control...Breakpoint Display... Within the Debugger...Trigger-Style Breakpoints ...Exit from Debugger...Disable	
7. Burning Proms	6-125
Personality Modules...Plugging In...Checksums ...Verify...16-bit...Standalone...Macros	
8. Generating Stimuli	6-135
How to do it	
9. Special Keys	6-140
Function Keys...Cursor Keys	
10. Mode Panels-- easy toggling of options	6-146
Analyzer...Display...Log	
11. Windows	6-151
12. Histograms	6-152
When to Use...How to Make a Histogram	

Chapter Seven: UniLab Command Reference

The Categories	7-2
The Commands	7-9

Chapter Eight: Target Notes

	(software order #)
General Information	8-2
1802/4/5/6 (disassembler only).....(DIS-18)...	8-5
6301/3	(DDB-63) 8-7
6500 series where the SYNC output exists..(DDB-65)...	8-10
6500 series piggyback devices w/o SYNC....(DDB-65P)...	8-14
6800/2/8 with external memory at page 0 ..(DDB-68)...	8-18
6801/3	(DDB-681) 8-21
6802 without external RAM at page 0	(DDB-682) 8-24
6805	(DDB-685) 8-25
6809	(DDB-689) 8-30
68000	(DDB-68K) 8-32
68008	(DDB-688) 8-36
68HC11	(DDB-611) 8-38
8048/35/39/40/49/50.....	(DDB-48) 8-40
8051/31/32/52 & 8051P	(DDB-51) & (DDB-51P) 8-45
8085 or 8080	(DDB-85) 8-50
8086/186/286 & 8088/188.....	(DDB-86) & (DDB-88) 8-53
8094/5/6/7	(DDB-96) 8-61
SUPER 8	(DDB-S8) 8-65
Z8	(DDB-Z8) 8-68
Z80 and NSC-800 and HD64180	(DDB-Z80) 8-72
Z8000	(DDB-Z8K) 8-76

Chapter Nine: TroubleShooting

Explanation	9-2
Solutions in Depth:	
Program hangs up on "Initializing UniLab. . . " message . . .	9-3
Program hangs on initialization some of the time, not all of the time	9-5
RS-232 error message "RS-232 Error #XX"	9-6
STARTUP does not work -- never get to see trace, or see trace filled with garbage	9-8
Error message: "NO ANALYZER CLOCK"	9-10
Program runs, UniLab traces, but reads bad data from stack	9-12
Program runs and UniLab traces, but does not disassemble properly	9-13
Program runs, UniLab traces properly, but cannot set a breakpoint-- gives a Debug Control not Established message	9-14
Program runs, UniLab traces properly, but cannot set a breakpoint-- hangs with red light next to Analyzer socket on until key pressed	9-15
Bad Input buffers on the UniLab, as if an IC has been blown.	9-16
Screen flickers when you use PgUp key to look at line history.	9-17

APPENDICES:

Appendix A:	UniLab Command and Feature List
Appendix B:	Sources of Cross Assemblers
Appendix C:	Cabling Chart
Appendix D:	Custom Cables
Appendix E:	UniLab II Specifications
Appendix F:	Writing Macros
Appendix G:	EPROMs and EEPROMs Supported
Appendix H:	Microprocessors Supported
Appendix I:	System Messages
Appendix J:	.BIN files and .TRC files

INDEX for both volumes

Chapter Six: The UniLab in Detail

Contents:

A Guide to This Chapter	6-2
1. Interpreting the Trace Display	6-5
What Each Column Means...Sample Traces...Moving through Trace...Symbolic Names...Toggling Display Options (Mode Panels)	
2. Readyng and Loading Memory	6-34
Emulation ROM...Getting Ready...Loading Programs ...Saving Programs	
3. Examining and Altering Memory	6-47
Memory Access...Read...Alter...Optional Assembler	
4. Setting up a Trigger (generating a trace)	6-64
Simple Example...NORMx Words...RESETting...General Purpose Triggers...Real-life Examples...Limits... Filtered Traces...Qualifying Events...Refining	
5. Saving Information	6-90
Screen History...Log File...Printer...Trace Save... Symbol Table...Binary Image...SAVE-SYS	
6. Breakpoints and the Debugger	6-100
Establish Debug Control...Breakpoint Display... Within the Debugger...Trigger-Style Breakpoints ...Exit from Debugger...Disable	
7. Burning Proms	6-125
Personality Modules...Plugging In...Checksums ...Verify...16-bit...Standalone...Macros	
8. Generating Stimuli	6-135
How to do it	
9. Special Keys	6-140
Function Keys...Cursor Keys	
10. Mode Panels-- easy toggling of options	6-146
Analyzer...Display...Log	
11. Windows	6-151
12. Histograms	6-152
When to Use...How to Make a Histogram	

A Guide to this Chapter

This chapter covers the capabilities of the UniLab II in detail. It's meant primarily as a reference chapter.

Review: What the UniLab does

The UniLab lets you look at the bus activity on your microprocessor control board. The UniLab captures a bus cycle in its trace buffer whenever your microprocessor:

- writes data to memory,
- reads data from memory,
- sends to a port,
- reads from a port,
- or fetches an opcode from ROM.

Capture bus activity

The UniLab can "freeze" this trace buffer at any time, and thus capture a record of bus activity. It then sends this record to your host computer, where you can:

- examine it,
- compare it to previous traces,
- save it,
- or print it.

Each line of the trace display includes the address your microprocessor is fetching from, reading from or writing to, and the data that appeared on the bus. If you have your disassembler enabled, you will also get the assembly language instructions that were fetched from ROM.

See: **Section One: Interpreting the Trace Display**

Program memory

Before you can capture a trace of your program, you have to load it into the UniLab's emulation memory (except when you run the program from a PROM chip-- see page 6-38).

See: **Section Two: Readyng and Loading Memory.**

Once you have the program in emulation ROM, you can look at the program, and change it.

See: **Section Three: Examining and Altering Memory.**

Capture the activity you need to see

You want to look at only a few of the millions of bus cycles that happen each second. You tell the UniLab what cycles you want to see by describing a "trigger event." The UniLab watches for that event on the bus.

See: Section Four: Setting a Trigger (generating a trace).

Record what you did

You can save any trace, any section of memory, or the current symbol table. You can also save the current state of the UniLab software.

While working with the UniLab, you can send all screen displays to the screen and also a file or a printer or both. You can also choose a mode which logs on the printer only the commands that access memory.

See: Section Five: Saving Information.

Look at the Internal State of the Processor

You can set a breakpoint in your program, and then restart the target board. The program will run to the breakpoint, then show you the register display when it stops.

After you have gained debug control you can:

- continue to another breakpoint,
- single step through your program,
- examine and change RAM, emulation ROM, and internal registers,
- or leave debug control.

See: Section Six: Breakpoints and the Debugger.

Save your code to silicon

Once you've completed testing your program, you can program an EPROM or EEPROM with the UniLab. See Appendix G for a list of PROMs that Orion supports.

See: Section Seven: Burning PROMs.

"Mock up" peripheral inputs

Sometimes you need to see how your microprocessor board responds to an input from a peripheral device. The stimulus generator of the UniLab allows you to produce any 8 bit signal you want-- or toggle individual lines.

See: **Section Eight: Generating Stimuli.**

Make use of special features and shortcuts

The UniLab makes full use of the function keys of your personal computer, including **ALTERed**, **SHIFTed**, and **CTRLed** function keys, and the keys of the numeric key pad.

Some of the function keys are pre-assigned to help screens (see On-Line Help chapter) or to commands. The others are left available for you to assign as you please.

See: **Section Nine: Special keys.** See also Chapter Four.

Function key 8 has a special effect--it gives you access to the pop-up panels, where you can easily change many options, including display and logging features.

See: **Section Ten: Mode Panels.**

Function key 2 also is special-- it splits the screen, giving you the ability to look at different parts of your trace at the same time, or to examine a textfile while looking at a breakpoint display, or . . .

See: **Section Eleven: Windows.** See also Chapter Four.

The Software Graphical Performance Measurement option gives you the ability to generate histograms of your target program's activity.

See: **Section Twelve: Histograms.**

1. Interpreting the Trace Display

Introduction

This section covers the trace display-- the record of bus activity that the UniLab captures for you.

The trace examples show a Z80 processor and an Intel 8096 processor.

Why you care about the trace display:

You want to find the bugs in your system. Bugs cause undesirable behavior in your system, which you can track down by looking at the record of bus activity on your board-- the trace display.

Contents

1.1	Feature Summary	6-6
1.2	The Trace: What Each Column Means	6-8
1.3	Sample Traces	6-10
1.4	Moving through the Display	6-17
1.5	Symbolic Names in the Display	6-21
1.6	Toggling Display Options	6-28

-- Interpret the Trace --

1.1 Feature Summary

While you are examining a trace, you can turn these options on and off:

<u>Option</u>	<u>Mode Panel</u>	<u>Commands</u>
Disassemble code	Yes	DASM DASM'
Substitute symbolic names for numbers	Yes	SYMB SYMB'
Show CONTrol column	Yes	SHOWC SHOWC'
Show MISCellaneous column	Yes	SHOWM SHOWM'
Binary number base for MISC	Yes	2 =MBASE
Fixed header	Yes	HDG HDG'
Stop display after each screen	Yes	PAGINATE PAGINATE'
Define symbolic names	NO	IS SYMFILE SYMLOAD
Show source lines in trace	NO	SOURCE SOURCE'

Mode panels:

Commands:

1. ANALYZER modes

DISASSEMBLER
SYMBOLS

DASM DASM'
SYMB SYMB'

2. DISPLAY modes

MISC COLUMN
CONT COLUMN
MISC # BASE
PAGINATE
FIXED HEADER

SHOWM SHOWM'
SHOWC SHOWC'
=MBASE
PAGINATE PAGINATE'
HDG HDG'

-- In Detail --

-- Interpret the Trace --

You can look at any portion of a trace you want:

<u>Feature</u>	<u>Cursor key</u>	<u>Command</u>
Show trace from top	HOME	TT
Show next step of trace	Down Arrow	none
Show next page of trace	PgDn	TR
Show trace from step <n> (resets default to n)	none	<n> TN
Show trace from step <n>, with no effect upon the default	none	<n> TNT
Dump trace buffer from UniLab	none	TD

You can save and compare traces (details in **Saving Information**):

<u>Feature</u>	<u>Command</u>
Save a trace to a file	TSAVE <file name>
Compare last <n> cycles of saved trace to current trace	<n> TCOMP <file name>
Compare saved trace to <u>result</u> of current trigger specification	<count> SC <file name>

-- Interpret the Trace --

1.2 The Trace: What Each Column Means

The header line of the display labels all but one of the columns:

cy# **CONT** **ADR** **DATA** **HDATA** **MISC**
(unlabeled column)

Each column displays a different piece of information:

cy# shows you what cycle you are looking at, relative to the trigger event. The trigger event is always labeled as cycle zero.

This column starts with an **f** when you produce a filtered display.

CONT shows you what the UniLab sees on the control inputs, and on the upper four bits of the address inputs.

The UniLab uses four of its inputs, labeled as

C7, C6, C5, and C4

to determine whether the bus cycle is a fetch, read, or write. The first digit of the **CONT** column shows those four inputs as a hexadecimal digit. The disassembler needs this information, but you can ignore it-- except when you are trouble shooting the wiring of the connection from your UniLab to your target board.

The second digit shows the four highest bits of the 20 bit address inputs to the UniLab, labeled as

A19 through A16.

While working with most 8-bit processors, these wires are not attached to anything, and so float high, at logic level one. The Z80, 8085, and NSC-800 processors don't follow this general rule-- they have one of these upper four wires connected to a processor pin. See the explanation on page 6-13.

You can use the mode panel (hit function key 8) to hide this column.

ADR shows the first 16 bits of the address bus, A0 through A15. See the Disassembler Note below.

The highest four bits, A16 to A19, appear as the right-hand digit in the **CONT** column.

-- In Detail --

-- Interpret the Trace --

DATA shows you what data was put on the bus. Depending on the type of the cycle, that "data" is either a data value or a machine language instruction. The data is 8 bits or 16, depending on the processor.

The center (unlabeled) column shows the disassembled instructions. Data reads and writes are also identified.

This column appears when you are working with the disassembler enabled, as you usually will.

HDATA shows you what values the UniLab reads on D8 through D15. This column only appears with 8 bit processors.

The UniLab doesn't use the full 16 bits of data input when working with processors that have an 8 bit wide external data bus. That makes these 8 inputs available to you for gathering more information about the outputs of other chips or ports on your board.

MISC shows you what values the UniLab reads on M0 through M7, the **MISC**ellaneous inputs. These wires are always available for you to connect anywhere you want on your board.

The number base is normally binary, but you can change it with the mode panel (**F8**). You can also use the mode panel to hide this column.

Disassembler note

With a processor-specific disassembler enabled, each line of the trace shows a complete assembly language instruction, no matter how many bytes it takes. On those lines that show an instruction which takes more than one cycle to fetch from memory, the **cy#** column contains the cycle number of the first fetch, and the **ADR** column contains the address of the first byte of the instruction (the first word on 16-bit processors).

The **MISC** and **HDATA** columns show only the state of those inputs during the last cycle of the instruction. Use the mode panel (**F8**) to turn off the disassembler if you want to see the state of these inputs during every bus cycle.

-- Interpret the Trace --

1.3 Sample Traces

This section shows two sample traces and explains the first few lines of both in detail.

The 8-bit processor example shows the Orion test program for a Z80 processor. The 16-bit processor example shows a trace of the test program for the Intel 8096.

A trace of the test program for your processor appears in the **Target Notes** chapter, and also in the Disassembler/Debugger writeup for your processor.

-- In Detail --

6-10

-- Interpret the Trace --

An 8 bit processor: Z80 trace

The following display shows a trace of the test program for the Z80 microprocessor. The test program was first loaded into the UniLab from disk with **LTARG**, and then started up with **STARTUP**. The **STARTUP** command captures a trace of the bus cycles starting at the reset address-- for the Z80, address 0000.

cy#	CONT	ADR	DATA		HDATA	MISC
0	B7	0000	310019	LD SP,1900	11111111	11111111
3	B7	0003	3E12	LD A,12	11111111	11111111
5	B7	0005	015634	LD BC,3456	11111111	11111111
8	B7	0008	119A78	LD DE,789A	11111111	11111111
B	B7	000B	21DEBC	LD HL,BCDE	11111111	11111111
E	B7	000E	C5	PUSH BC	11111111	11111111
F	D7	18FF	34	write	11111111	11111111
10	D7	18FE	56	write	11111111	11111111
11	B7	000F	C1	POP BC	11111111	11111111
12	F7	18FE	56	read	11111111	11111111
13	F7	18FF	34	read	11111111	11111111
14	B7	0010	3C	INC A	11111111	11111111
15	B7	0011	3C	INC A	11111111	11111111

2B	B7	0027	3C	INC A	11111111	11111111
2C	B7	0028	3C	INC A	11111111	11111111
2D	B7	0029	C30300	JP 3	11111111	11111111
30	B7	0003	3E12	LD A,12	11111111	11111111

This simple program is an infinite loop. It first initializes several registers, starting with the stack pointer.*

Then the program pushes a value on the stack and pops the same value, to demonstrate the working stack. Notice the cycles associated with memory reads and writes. These show you register and memory locations, just when you most want to know them.

After that come a series of "increment register A" instructions. The last command in the program, at address 29H, is an unconditional jump back to address 3, so that the program goes back to the second instruction.

* You must have a working stack for debugger commands to work.

-- Interpret the Trace --

An examination of the first two lines

This section dissects the first two lines of the Z80 trace. For the sake of simplicity, the **HDATA** and **MISC** columns (which were not attached to anything on the board) are not displayed.

```
cy#  
0 B7 0000 310019   LD SP,1900  
3 B7 0003 3E12     LD A,12
```

The first line of the display starts with cycle zero, which means that this cycle was the "trigger event."

The UniLab trace buffer captured a **31** on bus cycle 0, **00** on bus cycle 1, and **19** on cycle 2. These hexadecimal numbers were then translated by the disassembler into **LD SP,1900**.

The second line is labeled as cycle three, which lets you know that the Z80 microprocessor required three bus cycles to read the first instruction from ROM.

```
CONT  
0 B7 0000 310019   LD SP,1900  
3 B7 0003 3E12     LD A,12
```

The **CONTROL** column shows two different types of information. The high four bits of the byte (nibble) shows the control inputs, **C4** through **C7**. The low nibble shows the highest four bits of the address inputs, **A16** through **A19**. Both nibbles are important to the proper functioning of the disassembler and emulation ROM. You will only have to pay attention to this column if you suspect that the wires carrying these signals are improperly connected.

The high nibble is used by the processor-specific disassembler to distinguish between cycle types.

If the wires that carry the control signals have been incorrectly connected, then the disassembler will not work properly. The disassembler needs these control signals to classify each bus cycle as a fetch or read or write.

The first two lines both show the microprocessor fetching an instruction from ROM. With the Z80, **B** in the control column always indicates an instruction fetch, **D** marks the write cycles, and **F** marks a read.

-- In Detail --

-- Interpret the Trace --

CONT

```
0 B7 0000 310019 LD SP,1900
3 B7 0003 3E12 LD A,12
```

The low nibble carries information that is used by the emulation ROM. It is useful while troubleshooting, but otherwise is only for the curious. If you are curious, read on.

If the value of this nibble matches the value set by =EMSEG then the UniLab's emulation ROM will check whether the address is enabled.

The emulation ROM will put data on the bus only when the low 16 bits of the address fall into an enabled range (EMENABLE) and the number on inputs A16 through A19 match =EMSEG.

Every line of the Z80 test program display will have 7 as the upper four bits of the address. Three of the address inputs to the UniLab, A18, A17, and A16, are left to float high.

A19, however, is connected to the MREQ pin of the microprocessor. This "active low" output of the Z80 goes low when the processor Memory is REQuired. The Z80 needs memory access on all bus cycles, except when it is writing to or reading from a port.

Thus, these four inputs to the UniLab are usually 0111, which is hexadecimal 7. When the MREQ signal goes high, the 7 becomes F-- for example, when the Z80 is addressing a port address rather than memory.

If you have a different processor, your UniLab's inputs will be connected differently.

ADR

```
0 B7 0000 310019 LD SP,1900
3 B7 0003 3E12 LD A,12
```

The first line shows address 0000, the reset address for the Z80. The Z80 starts executing code from this address whenever it receives a reset signal. The second line shows address 0003, since the first instruction occupies bytes at addresses 0, 1, and 2.

-- Interpret the Trace --

DATA
0 B7 0000 310019 LD SP,1900

The first byte of the first instruction is 31 hex, which decodes as a command to load an immediate value into the stack pointer. The stack pointer of the Z80 holds a 16 bit value.

DATA
0 B7 0000 310019 LD SP,1900

That immediate value is 1900. Notice that the two bytes appear on the bus in reverse order, following the Intel convention, rather than the one adopted by Motorola.

DATA
0 B7 0000 310019 LD SP,1900
3 B7 0003 3E12 LD A,12

The second instruction loads an immediate value into the A register. This register of the Z80 only holds an eight bit value.

DATA
0 B7 0000 310019 LD SP,1900
3 B7 0003 3E12 LD A,12

The whole instruction only takes up 2 bytes, since the Z80 only needs one byte of data for the A register.

-- In Detail --

-- Interpret the Trace --

An 8096 trace-- 16-bit processor

The display below shows the trace of the test program for the 8096. The trace is shown only to highlight the difference between an 8-bit trace and a 16-bit trace.

Notice that there is no **HDATA** column in the trace, and that instead the UniLab shows 16-bits of data for each bus cycle.

The 8096 has a full 16-bit external data bus. With each bus cycle, the UniLab records a 16-bit word of either opcode or data.

A brief discussion of the trace appears on the following page.

cy#	CONT	ADR	DATA			MISC
0	FF	2080	A1004118	LD	SP,#4100	11111111
2	FF	2084	A100A01C	LD	AX,#A000	11111111
4	FF	2088	A100B01E	LD	BX,#B000	11111111
6	FF	208C	A100C020	LD	CX,#C000	11111111
8	FF	2090	A100D022	LD	DX,#D000	11111111
A	FF	2094	CA1C	PUSH	[AX]	11111111
C	EF	A000	A000	read		11111111
D	CF	40FE	A000	write		11111111
B	FF	2096	CE1C	POP	[AX]	11111111
F	EF	40FE	A000	read		11111111
10	CF	A000	A000	write		11111111
E	FF	2098	071C	INC	AX	11111111
11	FF	209A	071C	INC	AX	11111111
12	FF	209C	071C	INC	AX	11111111
13	FF	209E	071C	INC	AX	11111111
14	FF	20A0	071C	INC	AX	11111111
15	FF	20A2	071C	INC	AX	11111111
16	FF	20A4	071C	INC	AX	11111111
17	FF	20A6	071C	INC	AX	11111111
18	FF	20A8	071C	INC	AX	11111111
19	FF	20AA	071C	INC	AX	11111111
1A	FF	20AC	071C	INC	AX	11111111
1B	FF	20AE	071C	INC	AX	11111111
1C	FF	20B0	E7E5FF	LJMP	2098	11111111
1F	FF	2098	071C	INC	AX	11111111
20	FF	209A	071C	INC	AX	11111111

This simple program functions just about the same as the Z80 test program. Both are infinite loops that first initialize several internal registers, next push and pop a value, and then go through a series of "increment A" instructions.

The last command of the 8096 test program is a jump back to the first "increment A" instruction.

-- Interpret the Trace --

CONT
0 FF 2080 A1004118 LD SP,#4100

Remember that you can usually ignore the CONT column. But if you want to pay attention to it, notice that for the 8096, **F** marks a fetch, **C** marks a write and **E** marks a read cycle.

CONT
0 FF 2080 A1004118 LD SP,#4100

All four of the high address inputs to the UniLab, A19 through A16, float high. They are not attached to anything on the target board.

ADR
0 FF 2080 A1004118 LD SP,#4100

The reset address for the 8096 is 2080. Contrast this to the Z80, which has a reset address of 0000.

DATA
0 FF 2080 A1004118 LD SP,#4100
2 FF 2084 A100A01C LD AX,#A000
4 FF 2088 A100B01E LD BX,#B000

The opcode A1 decodes as a load of an immediate value into an internal register.

DATA
0 FF 2080 A1004118 LD SP,#4100
2 FF 2084 A100A01C LD AX,#A000
4 FF 2088 A100B01E LD BX,#B000

The last byte of the 4 byte instruction tells the 8096 which register to load the immediate value into.

DATA
0 FF 2080 A1004118 LD SP,#4100
2 FF 2084 A100A01C LD AX,#A000
4 FF 2088 A100B01E LD BX,#B000

The two-byte immediate value appears on the bus as the second and third bytes of the instruction. As with the Z80, the bytes appear on the bus in reverse order.

-- In Detail --

1.4 Moving through Your Trace Display

When the UniLab sends its trace buffer to the host machine, the host displays it starting from either cycle 0 or whatever cycle number was last set with <n> **TN**.

You will sometimes see everything you needed to know in the first screenful of the trace.

But much of the time you will need to look at a different part of the trace.

A small but complete set of commands moves you through the trace buffer.

Dumping the trace

Usually the UniLab will automatically dump the trace into the host computer. But if the trace buffer in the UniLab does not fill (especially when producing a filtered trace) then you will need to manually dump the trace to the host, with **TD**.

Look at next line of trace

Use the Down Arrow key (number 2 on the numeric key pad) to see the line of the trace that follows the "current" line.

The current line is usually the last one that you displayed on the screen. However, refer to the discussion of the history mechanism on the second page following.

Look at the next screen of trace

Use the Pg Dn key (number 3 on the numeric key pad) to see the next screenful of the trace, starting from the "current" line (or use the command **TR**).

-- Interpret the Trace --

Look at the trace, starting from cycle number <N>

Use one of the two commands: <n> **TN** or <n> **TNT**.

TN will also reset the default cycle number that **T** displays from (normally -5).

Use **TNT** to look at a particular cycle of the trace, without changing the default used by **T**.

To look at the trace starting from the top

The HOME key (number 7 on the numeric key pad) shows you the trace from the top (or use the command **TT**).

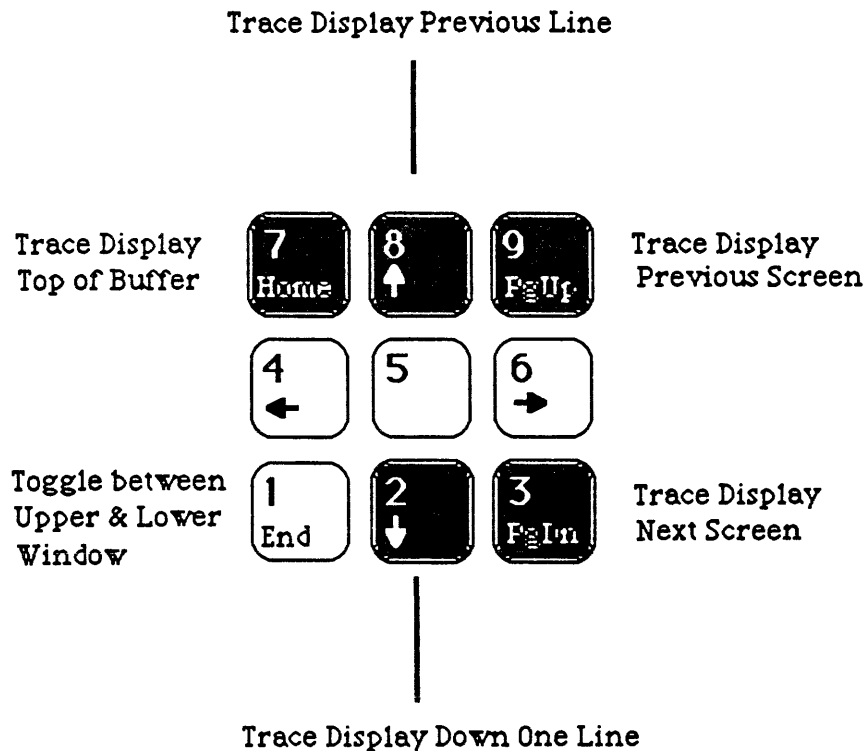
The trace and the "history" mechanism

Everything that goes by on the full screen or the lower window gets saved by the history mechanism of the UniLab. This handy feature allows you to review your past actions and past traces.

The PgUp key (number 9 on the numeric key pad) shows you one screen full of history.

The Up Arrow key (number 8 on the numeric key pad) shows you one line of the history.

Cursor Key Assignments for Viewing Trace Buffer Display



-- Interpret the Trace --

The trace, the history, and the Down Arrow

The UniLab "remembers" the cycle number of the last line of trace buffer you saw. Whenever you use one of the trace commands that start displaying from the "current" cycle number (PgDn, **TR**, or Down Arrow), the UniLab will normally start the display from after that last line.

However, if you use the Up Arrow and PgUp keys to look at the history of your session, you can end up with the cursor sitting on a line of trace display. The UniLab will temporarily call that cycle the current cycle.

If that is what you want, then you don't need to worry. But if instead you want to start displaying from the line of the trace buffer that you **last** displayed, first hit ENTER to get the cursor to a blank line, and then use any display command.

1.5 Symbolic Names in the Trace Display

Most people find it convenient to assign symbolic names to numbers. For example, LOOP.START conveys more information than address 2098. You will find your traces easier to read if you have symbolic names assigned to important addresses, ports and data.

You can load in the symbol table that your assembler generated, and have the same symbolic names that your source program assigned.

Or you can assign symbolic names one by one, using the IS command to give names to numbers.

You should not use a symbol name that is identical to a UniLab command. That would prevent you from using the command because the new interpretation of the name takes precedence.

Entering **SYMB'** tells the UniLab to ignore the symbol definitions.

STEP TO CREATE A SYMBOLIC TRACE FROM INTEL ASSEMBLER:

- ① USE ASM51 <FILENAME> DEBUG TO GENERATE DEBUG LIST FILE
- ② USE RLS1 <FILE.OBJ> TO <FILE.ABS> IREF(GENERATED)
THIS WILL GEN. A LIST FILE WITH .MSI EXTENSION
- ③ SYMFILE <FILENAME.MSI> TO ~~DEVELOPMENT SYS.~~
- ④ USE SYMTYPE TO FIND INTEL DEVEL. FORMAT
- ⑤ USE SYMLIST TO MAKE SURE TABLE IS IN

-- Interpret the Trace --

Choosing symbol file formats

Enter the **SYMTYPE** command to get the menu of predefined symbol table formats:

SYMBOL FILE FORMAT MENU

```
F1  2500AD SOFTWARE
F2  2500AD SOFTWARE (ABBREVIATED)
F3  ALLEN ASHLEY
F4  MANX AZTEC C
F5  AVOCET
F6  OTHER FIXED FORMAT
F10 RETURN TO COMMAND MODE
```

At this point you can select the desired format from the menu. If the format you require is not on the menu, see the subsection on **SYMFIX** on page 6-27 and in Chapter Seven.

Enter **SAVE-SYS** to make the selection permanent. (You can still change it again with **SYMTYPE**, then save the system again.)

If you don't use **SAVE-SYS**, the format you choose will only be used during the current session with the UniLab.

Load symbol table from file

After choosing the symbol file format, use:

```
SYMFILE <filename>
```

to load symbols in from a file. You will be prompted for the file name if you do not include it on the command line.

SYMFILE clears out the symbol table before loading the file. You can load in several symbol files, by using **SYMFILE+** to load each additional file.

Define individual symbols

A single symbol can be defined at any time with:

<n> IS <name> .

For example if you enter **1234 IS DELAYLOOP**, then DELAYLOOP will be displayed instead of the 1234, whenever 1234 occurs on the trace display.

You can also use DELAYLOOP in trigger specs, or to set breakpoints. For example:

DELAYLOOP AS

Toggle symbol translation on and off

To turn the symbol translation feature on for the trace display with **SYMB** or the Mode Panel (function key 8). Use the Mode Panel or **SYMB'** to turn the symbol translation off.

Note that enabling translation of the symbols will not change anything unless you have some symbols defined.

You can greatly improve readability of a hex trace by identifying the crucial subroutines and storage areas. If you are programming in a high-level language you can identify the run-time routines for improved readability.

Redefine a symbol

You can redefine a symbol at any time, simply by using with **IS** to define it again (only the most recent definition will be found). You cannot clear out only one symbol definition, but you can forget an entire symbol table with **CLRSYM**.

Save a symbol table as a file

You can save an existing symbol table as a named file with **SYMSAVE**, and reload a previously saved table from disk with **SYMLOAD**.

-- Interpret the Trace --

Setting the size of the symbol table

You can allocate up to hexadecimal 80 K (128K decimal) to the symbol table.

The size of the symbol table is set by giving the command:

<hex # of Kbytes> =**SYMBOLS**

then saving the newly altered UniLab software with **SAVE-SYS**. You must exit the program with **BYE** and start it again.

The size of the symbol table is allocated when the program starts up, and cannot be changed on the fly.

Use the command **?FREE** to find out how many bytes are allocated to the symbol table and to the line history. That display appears in decimal base, not hexadecimal.

Symbol example

The trace printout below shows a disassembled trace with symbol translation.

First eight symbol names were entered by hand:

1900 IS Init.Stack
3 IS Start.Loop
29 IS End.Loop
10 IS First.IncA
3456 IS Init.BC
789A IS INIT.DE
BCDE IS INIT.HL
28 IS LAST.INCA

And then **F9** was pressed, to get a trace of the startup:

cy#	ADR	DATA	
0	0000	310019	LD SP,INIT.STACK
3	START.LOOP	0003 3E12	LD A,12
5		0005 015634	LD BC,INIT.BC
8		0008 119A78	LD DE,INIT.DE
B		000B 21DEBC	LD HL,INIT.HL
E		000E C5	PUSH BC
F		18FF 34	write
10		18FE 56	write
11		000F C1	POP BC
12		18FE 56	read
13		18FF 34	read
14	FIRST.INCA	0010 3C	INC A
15		0011 3C	INC A
.		.	.
.		.	.
.		.	.
2A		0026 3C	INC A
2B		0027 3C	INC A
2C	LAST.INCA	0028 3C	INC A
2D	END.LOOP	0029 C30300	JP START.LOOP
30	START.LOOP	0003 3E12	LD A,12
32		0005 015634	LD BC,INIT.BC
35		0008 119A78	LD DE,INIT.DE
38		000B 21DEBC	LD HL,INIT.HL
3B		000E C5	PUSH BC

-- Interpret the Trace --

After these symbols have been loaded in, you can set a trigger or a breakpoint using the symbolic name:

LAST.INCA AS

The last example, below, shows breakpoint displays with these same symbols defined:

RESET END.LOOP RB resetting

AF=2B28 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFD SP=1900 PC=0029
END.LOOP 0029 C30300 JP START.LOOP (next step) ok

SSTEP NMI

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

N

AF=1228 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=FFFF IY=FDFD SP=1900 PC=0005
0005 015634 LD BC,INIT.BC (next step) ok

-- Interpret the Trace --

Reading other symbol table file formats

If the format you need is not included in the **SYMTYPE** menu, use **SYMFIX** to describe the format of other fixed length format files.

Fixed length format

The **SYMFIX** command is used to define parameters for any symbol file format which uses fixed length records. The 6 parameters for the **SYMFIX** command are as follows:

- a = offset from start of record to start of name field.
- b = 1 if address is 4 ASCII digits or 0 if 16-bit binary.
- c = offset from start of record to start of addr field.
- d = 1 if binary address has most significant byte first.
- e = pad characters used to fill between symbols.
- f = record length in bytes

Examples

The format of the 2500AD global symbol table is:

0 0 21 1 0 24 SYMFIX

The format for the ALLEN-ASHLEY symbol table is:

0 1 D 0 21 12 SYMFIX

Variable length format

If you have a variable length format symbol file, use the **AVOCET** choice in the menu if the format is NAME followed by VALUE.

Use the **MANX AZTEC C** choice if the format is VALUE followed by NAME.

-- Interpret the Trace --

1.6 Toggling Display Options On and Off

You can alter the way the trace gets displayed on your screen. Depending upon what you need, you can do everything from displaying only machine code to displaying source code lines in your trace.

You can change these options, either from the mode panels or with commands, as detailed in the following pages:

Disassemble code

Substitute symbolic names for numbers

Show CONTROL column

Show MISCellaneous column

Binary number base for HDATA and MISC

Fixed header

Stop display after each screen

Show source lines in trace.

Disassembly

When you don't want or don't need to see the assembly language instructions that each opcode represents, you can turn off the disassembler and then look at the same trace again. The disassembler remains off until you turn it on again.

Mode Panel:

1. ANALYZER modes
DISASSEMBLER
 SYMBOLS
 RESET

Command:

DASM DASM'

The display below shows the first fourteen cycles of the Z80 test program, with the disassembler on and with the disassembler off.

The LD A,12 instruction is underlined and the **PUSH BC** instruction is highlighted in both traces. The disassembled display has been extended so that cycle numbers will match up.

DISASSEMBLER ON

cy#	ADR	DATA	
0	0000	310019	LD SP,1900
3	<u>0003</u>	<u>3E12</u>	<u>LD A,12</u>
5	0005	015634	LD BC,3456
8	0008	119A78	LD DE,789A
B	000B	21DEBC	LD HL,BCDE
E	000E	C5	PUSH BC
F	18FF	34	write
10	18FE	56	write
11	000F	C1	POP BC
12	18FE	56	read
13	18FF	34	read
14	0010	3C	INC A

DISASSEMBLER OFF

cy#	ADR	DATA
0	0000	31
1	0001	00
2	0002	19
3	<u>0003</u>	<u>3E</u>
4	<u>0004</u>	<u>12</u>
5	0005	01
6	0006	56
7	0007	34
8	0008	11
9	0009	9A
A	000A	78
B	000B	21
C	000C	DE
D	000D	BC
E	000E	C5
F	18FF	34
10	18FE	56
11	000F	C1
12	18FE	56
13	18FF	34
14	0010	3C

-- Interpret the Trace --

Translate symbols

When you load a symbol table or define a symbol, symbol translation gets turned on. However, you may want to turn symbol translation off, to see the numeric values more easily.

Turn symbol translation on and off with the mode panel or with the commands:

Mode Panel:

1. ANALYZER modes
DISASSEMBLER
SYMBOLS
RESET

Commands:

SYMB SYMB'

-- Interpret the Trace --

Show or hide MISCellaneous column

When your MISC wires (M0 through M7) are connected to signals on your board, you will want to see the signals displayed. Otherwise that display just clutters up the screen.

This also hides the HDATA column on 8 bit processors.

Turn it off and on with the mode panel or with the commands:

Mode Panel:	Commands:
2. DISPLAY modes	
MISC COLUMN	SHOWM SHOWM'
CONT COLUMN	
MISC # BASE	
PAGINATE	
FIXED HEADER	

Show or hide CONTROL column

When you are troubleshooting, you will need to see the CONT column.

Most of the time it represents needless clutter-- unless you need to routinely see the full 20-bit address.

Turn it off and on with the mode panel or with the commands:

Mode Panel:	Commands:
2. DISPLAY modes	
MISC COLUMN	
CONT COLUMN	SHOWC SHOWC'
MISC # BASE	
PAGINATE	
FIXED HEADER	

-- Interpret the Trace --

Change the MISCellaneous display number base

When you have the MISC inputs connected to a port or a register, you will probably want to display that column in octal or hexadecimal, rather than in binary.

This feature alters the display base of the HDATA column at the same time.

Alter this variable with a command, or toggle between binary and octal with the mode panel:

Mode Panel:	Command:
2. DISPLAY modes	
MISC COLUMN	
CONT COLUMN	
MISC # BASE	<n> =MBASE
PAGINATE	
FIXED HEADER	

Stop after each screenful of trace

You usually want the display to stop after each screenful of display. But sometimes, when you are sending data to a file or a printer, you might want to have the whole trace scroll on by.

Turn this option off and on with the mode panel or with the commands:

Mode Panel:	Commands:
2. DISPLAY modes	
MISC COLUMN	
CONT COLUMN	
MISC # BASE	
PAGINATE	PAGINATE PAGINATE'
FIXED HEADER	

Have a fixed header for the display

This little extra allows you to have a fixed header on your lower display window, if you want.

Turn it off and on with the mode panel or with the commands:

Mode Panel:	Commands:
2. DISPLAY modes	
MISC COLUMN	
CONT COLUMN	
MISC # BASE	
PAGINATE	
FIXED HEADER	HDG HDG'

2. Readyng and Loading Memory

Introduction

This section covers emulation memory--

how to tell the UniLab what addresses to emulate,
how to load information into emulation ROM,
and how to save the data in emulation ROM.

The UniLab controls your target processor by emulating program memory. When the processor tries to fetch instructions or data from an address that has been **emulator enabled** (**EMENABLE**), the UniLab's emulation ROM responds on the bus.

Remember that the UniLab replaces your ROM rather than your microprocessor, and then watches the bus while the processor runs.

Contents

2.1	Feature Summary	6-35
2.2	What is Emulation ROM	6-36
2.3	Getting Ready	6-38
2.4	Loading Programs	6-42
2.5	Saving Programs	6-46

2.1 Feature Summary

<u>Feature</u>	<u>Menu</u>	<u>Commands</u>
Report Emulation STATUS	Yes	ESTAT
Choose 64K Segment	Yes	<hex digit> =EMSEG
Enable 2K blocks within 64K segment	Yes	<addr> EMENABLE
Load from an Intel HEX format file	Yes	HEXLOAD
Load from a binary file	Yes	<from addr> <to addr> BINLOAD
Save a block of memory to disk	Yes	<from> <to> BINSAVE
Enable minimal memory and load test program	Yes	LTARG
Load from a ROM	Yes	Commands are available, but the use of the menus is recommended

Commands:

Menus:

ENABLE PROGRAM MEMORY MENU

ESTAT	F1	DISPLAY CURRENT STATUS OF EMULATION MEMORY
<addr> EMENABLE	F2	ENABLE ONE BLOCK OF EMULATION MEMORY
ALSO <addr> EMENABLE	F3	ADD ANOTHER BLOCK OF MEMORY
=EMSEG	F4	SET A16-A19 MEMORY SEGMENT BITS
EMCLR	F5	DISABLE ALL EMULATION MEMORY
	F10	RETURN TO MAIN MENU

LOAD OR SAVE PROGRAM MENU

HEXLOAD	F1	LOAD INTEL HEX FILE
<from> <to> BINLOAD	F2	LOAD BINARY OBJECT FILE
<from> <to> BINSAVE	F3	SAVE A BLOCK OF MEMORY TO DISK FILE
LTARG	F4	LOAD A SAMPLE PROGRAM
	F10	RETURN TO MAIN MENU

PROM READER MENU

<from> <to> RPROM	F1	READ 2716/48016 - use PM16
<from> <to> R2532	F2	READ 2532 - use PM16
<from> <to> R2732A	F3	READ 2732 - use PM32
<from> <to> RPROM	F4	READ 2764 - use PM64
<from> <to> RPROM	F5	READ 27128 - use PM64 (PM56 for 27128A)
<from> <to> RPROM	F6	READ 27256 - use PM56
<from> <to> R27512	F7	READ 27512 - use PM512
	F9	Go to Prom Programmer Menu
	F10	RETURN TO MAIN MENU

See also Appendix G for more info on EPROMs.

-- Loading Emulation ROM --

2.2 What Is Emulation ROM ?

You use the UniLab to watch the execution of a program on your microprocessor board. Microprocessors usually run a program that is loaded into ROM or RAM. While using the UniLab, you load the program into emulation ROM.

32K or more of emulation memory

The standard UniLab contains 32K bytes of 195 ns static RAM which functions as emulation ROM. An optional expansion board can expand this capacity up to 128K bytes. This RAM appears to the target system as ROM, and cannot be altered by the target microprocessor.

Cable Connections

Most of the data and address lines are connected by plugging the emulator cable into a single PROM socket in the target system, as explained in the **Installation** chapter (two sockets for a 16 bit data bus).

Many ROMs can be emulated with one connection socket, but the sockets of emulated ROM must be empty to prevent bus contention.

Using the Emulator without the Analyzer

The analyzer cable must be hooked up for the emulator to operate properly. In the unlikely event that you want to use the emulation ROM without using the analyzer, you must still connect the analyzer cable from the UniLab to your board.

The UniLab must see the full address bus to emulate properly, and some of the address signals are picked up by the analyzer cable.

20-bit addresses

Since the UniLab accepts a 20 bit address input, it can handle target systems with up to 1 megabyte of active memory, or even more if you connect chip select logic to the A19 input to the UniLab.

Emulate throughout a 128K region

Any combination of 2K byte memory segments can be enabled, as long as they are all in the same 128K region. That is, you can emulate 2K chunks scattered throughout the range C0000 through DFFFF, since that forms one 128K region.

However, you would not be able to emulate some memory within the range 10000 to 1FFFF and other memory in the range 30000 to 3FFFF, since those two 64K segments are not contained within the same 128K region.

The general rule-- all emulated memory must have the same value for the upper three bits of the full 20 bit address. You set this value with =EMSEG. You rarely need to change this value.

Watch out

Since the 32K UniLab emulates 128K of address space in only 32K of physical RAM, each physical location represents four emulation ROM addresses. This can cause problems if you are not aware of it.

For example, the four addresses 00000, 08000, 10000 and 18000 all refer to the same physical memory location.

If you try to enable both **0 TO 7FF** and **8000 TO 87FF**, then you will find that both sections of memory always contain the same data. Those two ranges of emulation ROM both refer to the same RAM locations in the 32K UniLab.

-- Loading Emulation ROM --

2.3 Getting Ready. . .

Before you can start working on your program, you have to enable the emulation ROM and load your software into the UniLab's emulation memory.

When you enable a section of memory, you are telling the UniLab what addresses you want it to respond to.

The minimal memory necessary

The **LTARG** command enables a 2K section of memory and loads in a simple test program (on some packages, such as the 8096, the **LTARG** command enables several 2K sections). If you are in doubt about what memory to enable for your processor, type in **LTARG**, and note the values of **=EMSEG** and **EMENABLE**.

For example, the Z80 test program sets up the variables as:

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

In general, you have to enable the reset address for your processor. The only exception occurs when you want to analyze a program running from ROM chips instead of emulated ROM.

The exception: Running a program from ROM chip

When you want to run a program entirely from ROM on your target board you must first use **EMCLR** to clear emulation memory, and then use the Mode Panel option **SWI VECTOR** (SoftWare Interrupt **VECTOR**) or the command **RSP'** to disable the debugger.

If you don't disable the software interrupt vector under these circumstances, then the UniLab will give you an error message when you try to start the analyzer. When the debugger is enabled, the UniLab writes information into the reserved area when you start the analyzer. If the reserved area is not being emulated, you will get the message "Debug Control not established."

You can use ROM chips for some of your program, and use emulation memory for the rest of it.

No matter what else you do, you must always either emulate the reserved area (see appendix H) or disable the debugger.

-- In Detail --

The high four address bits

Address bits A16 to A19 are set with <hex number> **=EMSEG**, where hex number is the desired digit for A19-A16. With this command you tell the UniLab which 64K segment of memory you want to emulate, out of the possible 1 megabyte that a processor with a 20 bit address bus can access.

The value of **=EMSEG** is initialized to the correct value. You will probably never have to alter it.

The UniLab uses **=EMSEG** to decide whether to put data on the bus. When the processor tries to fetch information from ROM, the UniLab first checks whether the upper four bits of its address inputs match the value of **=EMSEG**.

If the UniLab finds that the upper four bits match, then it checks whether the lower 16 bits of the address are enabled.

On many 8 bit processors these inputs just float high, so that the **=EMSEG** value is usually F (1111 binary).

Emulating two 64K segments

You can emulate address in 128K, as long as the two 64K segments are neighboring segments (that is, only A16 differs). Use **=EMSEG** to set the values of **A16** through **A19** that the UniLab's emulation ROM will respond to.

For example, 4 is 0100 in hexadecimal,
while 5 is 0101, so that you could give this enable
command:

4 =EMSEG 0 TO 7FF EMENABLE
ALSO 5 =EMSEG 1000 TO 17FF EMENABLE

-- Loading Emulation ROM --

The other 16 bits

You can enable the UniLab's memory 2K at a time. The memory that you enable will be in the 64K segment that you last set with **=EMSEG**.

You are telling the UniLab what range of addresses on A0 through A15 you want it to respond to.

To enable ROM from address 0 to 17FF you type:

0 TO 17FF EMENABLE

You could instead enable locations F800 to FFFF by entering:

F800 TO FFFF EMENABLE

The **TO** is necessary to indicate an address range. If you enter <16 bit address> **EMENABLE** the single 2K segment which includes that address will be enabled. For example,

1000 EMENABLE

will enable the 2K segment 0 to 17FF.

Enabling several areas

Each **EMENABLE** statement usually clears out the previous settings. However, if you use **ALSO** you can have the UniLab respond to both the previous setting and the new one.

For example, to enable 0 to 17FF and F800 to FFFF, you type:

0 TO 17FF EMENABLE ALSO F800 TO FFFF EMENABLE

Be careful when enabling several areas with a 32K UniLab. The 128K area within which you can enable 2K blocks gets mapped onto the 32K of the UniLab. This means that 0000 addresses the same memory location as 8000. 2000 refers to the same location as A000.

Keeping track

Every time you issue the **EMENABLE** command, the system will display the complete resulting memory enable status. If you want to see this enable status display without changing enables, just enter **ESTAT**.

-- In Detail --

6-40

-- Loading Emulation ROM --

Saving the settings

Once you have them set properly for a project, you will want to can save the enable settings of emulation memory. The contents of emulation memory can also be saved, with the **BINSAVE** comman.

You enter:

SAVE-SYS <filename>

after you enable the memory to save the current state of the UniLab software, including the emulation memory settings.

From then on, when you start the program by typing in the filename, the proper area of the UniLab's memory will already be enabled.

-- Loading Emulation ROM --

2.4 Loading the Target Programs into Memory

You load in your program after you have enabled a section of memory large enough to contain your program.

You can load opcodes into memory from a disk file, by hand, from a ROM chip or from an Orion test program.

Loading from disk files

The UniLab software provides you with four different ways to load a program from a file on disk. Depending on your assembler or compiler, you will chose one of these methods.

1. If you compiled or assembled the code into a binary file on a disk, then load it with

`<from addr> <to addr> BINLOAD <filename>`

The filenames usually end in **.BIN**, **.COM**, or **.TSK**. You will be prompted for the file name if you do not include it on the command line. You can use the DOS command **EXE2BIN** if your assembler produces a **.EXE** file.

The program will be loaded starting at the address you gave.

You save memory to a file with the **BINSAVE** command.

2. Read Intel-format HEX object files from a disk with

`HEXLOAD <file name>.`

You will be prompted for a file name if you do not include it on the command line. The addresses will automatically be converted to the correct ones for the host image of the target program. This method is much slower than **BINLOAD**.

If your assembler will only make Intel Hex files, you can still use the UniLab command **BINSAVE** to make a binary format file. Just load the hex file the first time, and then use

`<from> <to> BINSAVE <filename>`

to save the memory as a binary image. From then on you can use

`<from> <to> BINLOAD <filename>`

to load the program into memory.

-- Loading Emulation ROM --

3. Download Intel hex format programs from another computer system with **HEXRCV**, if your PC has two serial ports. The sending computer must support the XON/XOFF protocol.

After you type this command, your PC will accept hex code through its second port until you press a key or the PC receives an end of file message.

While this method is useful for interfacing with existing systems, it makes more sense to use your personal computer for program development and avoid the bottleneck of program downloading. See Appendix B for a partial list of assemblers and compilers for the personal computer.

4. If your assembler or compiler can assemble directly into memory at a specified location other than the origin, you can instruct it to leave the object code in some unused area of host memory (C000 to E000 is free in most systems).

Then when you enter the UniLab program you can download from your host's memory to the UniLab's emulation ROM with

<fromadr> <toadr> <targadr> **MLOADN**.

Note that the first two addresses refer to RAM in your host machine, the third address is in UniLab emulation ROM.

Hand enter Code

You can also hand enter a program, poking machine language instructions into memory. We recommend this only to those suffering from computer nostalgia.

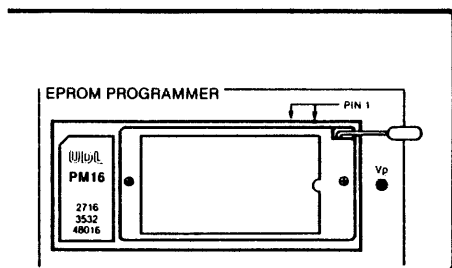
You hand-enter a short program by using the memory patching commands of the UniLab system. Type in <address> **ORG**, where the address is the start of the target program, then enter <byte> **M** or <word> **MM** for each byte or word of the program.

-- Loading Emulation ROM --

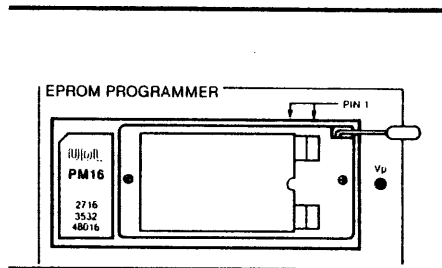
Read a program from ROM

The UniLab software also allows you to read a program from ROM. We support all of the most popular EPROMs-- see Appendix G: EPROMs Supported.

Read a program from a ROM by first placing the chip into the UniLab's programming socket. Hit function key 10 to get the main menu, and then function key 9 to get the PROM reader menu.



28 Pin EPROM in Programming Socket



24 Pin EPROM in Programming Socket

PROM READER MENU

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

(Press the Function Key to select item):

Watch out

Avoid leaving any PROM in the socket after you read it or program it.

2764s and up will sometimes erase location zero when you turn on the UniLab.

Loading the sample program

Or instead, you can load a test program, with **LTARG** (Load **TAR**Get memory).

LTARG enables emulation memory and loads a simple test program. Chapter 9 contains a trace of the test program for each microprocessor, along with examples of debugging operations.

The separate Disassembler/Debugger writeup on your processor covers the test program and the debugging operations more completely.

-- Loading Emulation ROM --

2.5 Saving Programs

You can save a program for later use with **BINSAVE** as described below. There are at least five situations in which you will want to save a program:

- 1) You have changed the program since you loaded it in, by moving sections of memory or poking in an opcode.
- 2) You have loaded a program using **HEXLOAD**, and want to be able to use **BINLOAD** instead.
- 3) You have loaded a program from a ROM.
- 4) You have "hand assembled" a program.
- 5) You want to make a macro that tests equipment by loading and running a test program.

When you have completed your design, you can "save" a program to ROM with the PROM programmer menu. See section 7.

Saving with BINSAVE

Any area of emulation memory can be saved to disk as a named file. Type in

<from address> <to address> **BINSAVE** <file name>.

The addresses refer to emulated memory. If you leave off the filename, then you will be prompted for a name.

3. Examining and Altering Memory

Introduction

After loading a program into emulation memory, you can immediately run it. However, you often want to look at the program first, to refresh your understanding of the code, or to verify that you have loaded in the correct file.

And as you work on the program, you will want to look at portions of your code, and perhaps alter the code.

You can also examine and alter RAM, but only after you have established debug control (see subsection 3.2 and section 6).

Contents

3.1	Feature Summary	6-48
3.2	Memory Access	6-49
3.3	Read from Memory Disassemble Peek, dump or compare	6-52
3.4	Alter Memory Fill Move Poke	6-57
3.5	On-Line Assembler	6-61

-- Examine and Alter Memory --

3.1 Feature Summary

All memory access commands work both on emulated ROM and on target RAM. However, to access RAM you have to first establish debug control. See the next two pages.

Feature	Menu	Command
Examining Memory		
Dump a range of memory	Yes	<start> <end addr> MDUMP
Disassemble a range of memory	Yes	<start> <# of lines> DM
Disassemble into right hand window	NO	<start> DN
Compare two ranges of memory	Yes	
	<start> <end>	<comparison addr> MCOMP
Look at one byte	NO	<addr> M?
Look at one word	NO	<addr> MM?
Altering Memory		
Fill a range of memory with one byte value	Yes	<start> <end> <byte> MFILL
Alter a single byte	Yes	<value> <addr> M!
Alter a single word	Yes	<value> <addr> MM!
Move a range of memory to a different place	Yes	<start> <end> <new start> MMOVE
Set up the address for subsequent M and MM commands	NO	<addr> ORG
Store one byte and update ORG	NO	<byte> M
Store a word and update ORG	NO	<word> MM
Optional		
On-line assembler	NO	<addr> ASM
Assemble code from FORTH file	NO	<addr> <from scr> <to scr> ASM-FILE

Command:

Menu:

EXAMINE OR CHANGE PROGRAM MEMORY MENU

MFILL	F1	EXAMINE A RANGE OF MEMORY
DM	F2	DISASSEMBLE FROM MEMORY
M!	F3	CHANGE ONE BYTE
MM!	F4	CHANGE ONE WORD
MFILL	F5	FILL A RANGE OF MEMORY WITH ONE VALUE
MMOVE	F6	MOVE AN AREA OF MEMORY
MCOMP	F7	COMPARE TWO AREAS OF MEMORY
	F10	RETURN TO MAIN MENU

-- In Detail --

3.2 Memory Access: Emulation ROM and RAM

The commands that access memory have two complications:

- 1) When you access emulation ROM, you will cause the program to stop.
- 2) You cannot access RAM unless you have first established debug control.

Access to emulation ROM

When you read from or change emulation ROM, the UniLab has to take control of the memory chips that emulate ROM. While this is going on, your processor will not be able to read instructions from ROM-- which causes your target program to crash.

Be aware that you have to restart the target program after examining emulation ROM, because you and the processor cannot look at program memory at the same time.

Access without crashing

While at a breakpoint, you can examine emulation ROM without crashing your target system. Your processor does not need access to memory while you have debug control.

-- Examine and Alter Memory --

Access to RAM

Normally, all the commands that read and write memory perform their work on emulation ROM. However, you can examine or alter RAM once you have established debug control. See section 6 of this chapter to learn how to establish debug control.

If you try to read or write RAM without first establishing debug control, the attempt will fail, and you will get two messages:

- 1) a "not enabled" message, informing you that you are trying to access an address that the UniLab is not emulating,
- 2) a "Debug Control not established" message, when the UniLab software tries to access the RAM.

Since you are trying to access RAM, not emulated ROM, the address you specify has not been enabled-- that is the meaning of the first message.

The second message tells you that the routines which alter RAM will not work, because you have not established debug control.

Successful Access to RAM

If you first get debug control, then you can access RAM. You will still get the "not enabled" message, to remind you that you are working on RAM, not on emulated ROM.

Processors with RAM and ROM in the Same Address Space

With most processors, RAM occupies one range of memory, and ROM another range.

However, some processors allow you to have RAM and ROM at the same addresses at the same time, such as:

the **8051** family,
the **Z8** family,
and the **64180**.

If you have one of these processors or the others that allow ROM and RAM to simultaneously occupy the same address, you will not be able to access RAM that occupies the same addresses as emulated ROM until you have established debug control and have issued the command **TRAM**. This command tells the UniLab to try to access RAM rather than emulation ROM.

It would be wise, after you've used **TRAM**, to issue the command **TRAM'** when you are done with looking at or altering target RAM.

-- Examine and Alter Memory --

3.3 Read from Memory

When you read from memory, you have a choice of disassembling from memory or just dumping the hexadecimal opcodes. You will usually disassemble program memory and dump data memory.

If you don't have a disassembler package for your processor, you will not, of course, be able to disassemble.

Disassemble from memory

Two commands allow you to disassemble from memory:

DM and **DN**

Both commands take as a parameter the starting address to disassemble from. **DM** has a second parameter-- the number of lines of disassembled code to display.

DN outputs until it fills up the right hand window of the screen, and so does not need a second parameter.

Watch out

On some processors-- those that do not have a signal to mark the first fetch of a multi-byte opcode-- if you specify an address that starts the disassembly from anywhere but the first byte of an instruction, you will see at least a few incorrectly decoded instructions. Your Orion disassembler simply starts disassembling from whatever address you give it.

No matter what address you tell the disassembler to start from, it will try to interpret the hexadecimal code it sees as the first byte of an opcode. Once the disassembler gets back "in sync," it will decode properly.

Good disassembly:

```
0 7 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
```

"Out of Sync" disassembly:

```
1 7 DM
0001 00 NOP
0002 19 ADD HL,DE
Back in sync ----> 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
```

-- Examine and Alter Memory --

Peeking at, dumping or comparing memory

You can either "peek" at a byte or two, or dump a range of memory. You can also compare two ranges of memory.

Peeking

You peek into memory with:

<addr> **M?**
<addr> **MM?**

The first command looks at a byte, the second at a word (two bytes).

Peeking example

13 **M?** 3C ok

C **MM?** BCDE ok

Dumping

One command allows you to see the hexadecimal contents of a range of memory:

<start> <end addr> **MDUMP**

MDUMP will start dumping from whatever address you specify, showing 10 (hex) bytes of memory on each line. It always displays a full line, so that the second address will get rounded up, if necessary. The right-hand side of the display shows what ASCII characters, if any, the hexadecimal codes correspond to.

```
0 14 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  <<<<<<<<<<<<<<<<<<<<<<
```

-- Examine and Alter Memory --

Comparing

This command compares two ranges of memory, and reports any discrepancies it finds:

```
<start> <end addr> <comparison addr> MCOMP
```

MCOMP will start comparing from whatever address you specify. It compares the range that you specify.

You will find this command especially useful for comparing the contents of a PROM to the expected contents:

- 1) Put the expected contents in one range of memory,
- 2) move the actual contents to another range (with the PROM reader menu), and
- 3) use **MCOMP** to compare the two.

Example

Notice how, in this example, **MCOMP** starts finding bytes that don't match after comparing five of them. It would continue to compare bytes until it had compared the data at 120 (hex) to that at 820 (hex)-- or you can terminate the display by hitting any key.

```
105 120 805 MCOMP  
Data is 16 at addr 0110 ..but is 5 at addr 0810  
Data is 90 at addr 0112 ..but is 80 at addr 0812  
Data is 27 at addr 0116 ..but is 23 at addr 0816
```

3.4 Alter Memory

You can alter memory in a heavy-handed manner, filling or moving blocks of memory. You can also alter it byte by byte, or word by word.

And with the optional on-line assembler, described in subsection 3.5, you can alter memory by entering assembly language commands rather than poking bytes into memory.

Filling blocks

You generally fill blocks only for test purposes-- putting into a range of memory a long series of identical instructions. You move blocks only for patching purpose-- pushing a block of memory up or down to make room for an extra instruction or block of code.

Altering bytes or words

You will more often patch code on the fly by altering code a byte or a word at a time. For example, you can change the value a program sets the register to, or you can replace one instruction with another, by pushing in the hexadecimal opcode.

-- Examine and Alter Memory --

Fill memory

You fill a range of memory with the **MFILL** command. It sets every byte in the range to the same value:

```
<from address> <to address> <byte value> MFILL
```

Testing

This is a handy way to test the data and address lines of your processor board:

- 1) Fill a range of emulation memory with the opcode for NOOP, or other simple instruction, starting at the reset address.
- 2) Start up the processor and capture a trace, using the command **STARTUP**.
- 3) Examine the trace and verify that the address lines and data lines work properly.

Example

To fill 80 bytes of emulation ROM with FA, starting at address 00:

```
0 100 FA MFILL
```

Move memory

You copy information from one range of memory to another with:

<start address> <end address> <copy starting at address> **MMOVE**

You will want to do this to make room for extra instructions when patching code, or to move large chunks of code or data for other reasons.

Limitations

You can copy from any address to any other. However, neither the source range nor the destination range is allowed to cross over a 32K boundaries.

That is, you can copy from the range 5000-7000 to the range 8000 to 10000.

But you cannot copy from the range 7FFE-8001 to anywhere, since that range, small as it is, crosses a 32K boundary.

Overlapping ranges

This command is smart enough to decide whether to start moving from the front or the back when moving into an overlapping range of addresses.

Example

To copy the code at 100 through 152 into 105 through 157:

100 152 105 MMOVE

To copy from 230 through 370, starting at 220:

230 370 220 MMOVE

-- Examine and Alter Memory --

Change memory byte-by-byte and word-by-word

There are three ways to alter memory on a small scale:

poking bytes into specific addresses, **M!**
poking words into specific addresses, **MM!**
setting up an origin address, and then storing bytes
and words at sequential addresses. **ORG M MM**

Poking bytes

You poke bytes into specific addresses with:

<byte> <address> **M!**

Poking words

You poke words into specific addresses with:

<word> <address> **MM!**

If you have a disassembler, then the UniLab program knows which order to store bytes into memory.

Setting up an origin and storing bytes and words

You set up an origin address with:

<address> **ORG**

and then can store either bytes or words with:

<byte> **M**
and
<word> **MM**

These commands both store the value and increment the address.

You will want to use this method whenever you need to store several opcodes at sequential addresses.

3.5 On-Line Assembler

The processor specific on-line assemblers, **ASM** and **ASM-FILE**, allow you to write assembly language patches to your target program, instead of having to poke hexadecimal codes into memory.

Type <addr> **ASM** to invoke the assembler on the code that you type in from the keyboard.

Type <addr> <from screen #> <to screen #> **ASM-FILE** to invoke the assembler on code that has already been written into a FORTH file.

Overwrite memory locations

Both commands process assembly code instructions and write machine language codes into memory. You overwrite-- and therefore lose-- the data already in memory.

Choose the starting address

If you do not include the address, the assembler will use the last address stored by the **ORG** command.

Conventions

The on-line assembler will only accept assembly language instructions, not **ORIGIN** statements or **EQU** statements. (You can use the UniLab command **IS** to define symbols.)

Only one instruction per line.

The normal conventions of assembly language apply. For example, at least one space between the instruction and the operands.

You can include comments on a screen by putting a semicolon (;) on a line. The assembler will ignore everything after the semicolon on that line. The semicolon must either be the first character on the line, or be preceded by at least one space.

-- Examine and Alter Memory --

Entering instructions from the keyboard

When you use **ASM** you can include an assembly language instruction on the command line, and assemble only that one instruction:

```
1200 ASM INC A
```

You can enter multiple lines if you do not include an assembly language instruction on the command line:

```
1100 ASM
```

ASM will give you as a prompt the address to which it is assembling, and wait for you to give it an instruction followed by a carriage return.

The assembler will continue to prompt you with an address and patch assembled code into memory, until you feed a blank line (hit return on an empty line).

Entering instructions from a FORTH file

If you only have a few lines of code, you can use the screen that **MEMO** puts you into, and the two following (screens 1D through 1F). See the command reference entry for **MEMO** to get a few pointers on using the FORTH screen editor. You might also want to look at Appendix F.

You will want to put the code into a file of its own if you have many lines of code, or if you want a more convenient way to archive the code.

-- Examine and Alter Memory --

Putting code in its own FORTH file

First close the current file (**UniLab.SCR**) with the command:

CLOSE

Next create a new file with:

OPEN-NEW <file name>

and give it a size with:

<# of screens> **SCREENS**

1K is allocated per screen. Use the command:

<screen #> **EDIT**

to get into the file. NEVER use screen zero.

Assembling code from FORTH screens

You will then be able to use **ASM-FILE** to assemble the code stored in your new file. For example, to assemble screens one through four into emulation ROM, starting at address 1200:

1200 1 4 ASM-FILE

When you are done with assembling, use **OPEN UNILAB.SCR** to close your file and re-open the UniLab.SCR file. If you don't do this, then some of the on-line help facilities and error messages will not work.

4. Setting a Trigger (Generating a Trace)

Introduction

This section shows how you describe to the UniLab the bus activity that you want it to search for. The power of the UniLab comes from its ability to capture and display to you only the program activity that you want to see.

Usually you will be looking at the bus activity when you are trying to find a bug. But sometimes you will want to look at your code as it executes just to see what is going on.

Contents

4.1	Feature Summary	6-65
4.2	Overview	6-67
4.3	A Simple Example	6-68
4.4	The NORMx Words	6-72
4.5	RESET ting	6-74
4.6	General Purpose Triggers	6-75
4.7	Real-life Examples	6-78
4.8	The Limits of Triggers	6-80
4.9	Filtered Traces	6-82
4.10	Qualifying Events	6-85
4.11	Stepwise Refinement	6-89

-- Set Trigger --

4.1 Feature Summary

Feature	Menu	Command
Start the target program and show first cycles	Yes	STARTUP
Show what the program is doing right now	Yes	NOW?
Sample address lines, twice/second	Yes	ADR?
Sample all lines, once each second	Yes	SAMP
Set a trigger on an address	Yes	<16 bit addr> ADR
Set trigger on CONTROL inputs	NO	<byte> CONT
Set trigger on a data value	Yes	<byte> DATA
Set trigger on high byte of data	NO	<byte> HDATA
Set trigger on high byte of address	NO	<byte> HADR
Set trigger on low byte of address	NO	<byte> LADR
Set trigger on MISCellaneous inputs	NO	<byte> MISC
Set trigger on Range of values	Yes	TO
Invert following trigger	Yes	NOT
Add following trigger to current	NO	ALSO
Startup Analyzer	Yes	S
Startup Analyzer, capture new trace that starts where current trace ends	NO	S+
Don't restart target program when Analyzer starts	Yes	RESET
Do restart target program when Analyzer starts	Yes	RESET'
Clear out previous trigger spec	NO	NORMT NORMM NORMB
Change Delay Count	NO	<count> DCYCLES
Produce a filtered trace, showing only the cycles that match trigger	Yes	ONLY
Produce a filtered trace, showing one, two or three following cycles	NO	1AFTER 2AFTER 3AFTER
Set up a "qualifier" for trigger	NO	AFTER

-- Set Trigger --

Command:

Menu:

ANALYZER MENU

STARTUP	F1	RESET AND TRACE FIRST CYCLES
NOW?	F2	TRACE IMMEDIATELY
NORMT <addr> ADR S	F3	TRACE FROM A SPECIFIC ADDRESS
<from> <to> CYCLES?	F4	COUNT CYCLES BETWEEN TWO ADDRESSES
SAMP	F5	SAMPLE THE BUS CONTINUOUSLY
ADR?	F6	SAMPLE ADDRESS ACTIVITY
RESET RESET'	F7	TURN RESET OFF
	F10	RETURN TO MAIN MENU

ANALYZER TRIGGER MENU

NORMT <addr> ADR S	F1	TRIGGER ON AN ADDRESS
NORMT <from> TO <to> ADR S	F2	TRIGGER ON A RANGE OF ADDRESSES
NORMT <f> TO <t> ADR <byte> DATA S	F3	TRIGGER ON A RANGE OF ADDRESSES AND A DATA VALUE
NORMT NOT <f> TO <t> ADR S	F4	TRIGGER OUTSIDE A RANGE OF ADDRESSES
ONLY NOT <f> TO <t> ADR AFTER <addr> ADR S	F5	FILTER, EXCLUDING A RANGE OF ADDRESSES AFTER A ADDRESS
RESET RESET'	F6	TURN RESET OFF OR ON (reset is now xxx)
	F10	RETURN TO MAIN MENU

-- In Detail --

-- Set Trigger --

4.2 Overview

All the examples show traces of a Z80 program, with the CONT column and the MISC column turned off.

The first part introduces triggers with a simple example. The simplest trigger, and the most commonly used, is a trigger on a program address.

The next part covers general purpose triggers. You can trigger on several different values and on ranges of values. You can tell the analyzer to look at the control lines, the address lines, the data lines, the miscellaneous inputs, or any combination of them.

The real-life examples show how you can put trigger specs commands together to solve specific problems.

Filtered traces, introduced in the following part, allow you to look at only the cycles that interest you. You use qualifiers to set up preconditions-- the trigger will not occur until after the preconditions are met.

-- Set Trigger --

4.3 A Simple Example

When you use the UniLab, you will most often want to look at a trace of the bus activity that follows a certain instruction.

For example, if you have a conditional jump instruction at address 510 of your program and want to see where it jumps to, type in the command:

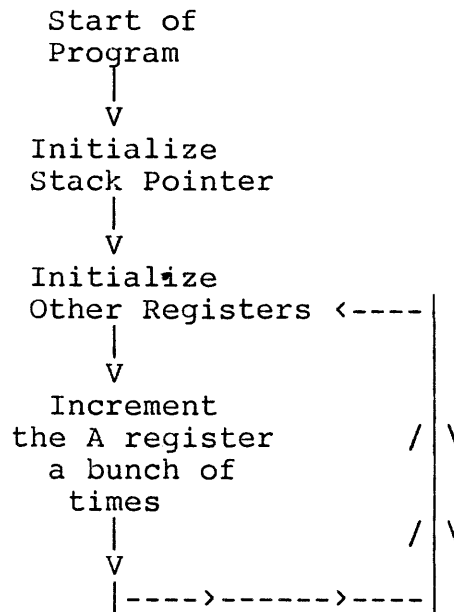
510 AS

After you hit a carriage return, the UniLab will start searching for address 510 on the target system's bus. The first time it sees that address, it will "trigger," and then freeze the trace buffer 165 bus cycles later.

While your target program continues, the UniLab sends that trace buffer to the host computer. The top of the trace fills your screen, showing the five bus cycles that preceded address 510 (labeled -5 to -1), the trigger cycle (labeled 0), and some of the cycles that follow.

Simple Z80 example

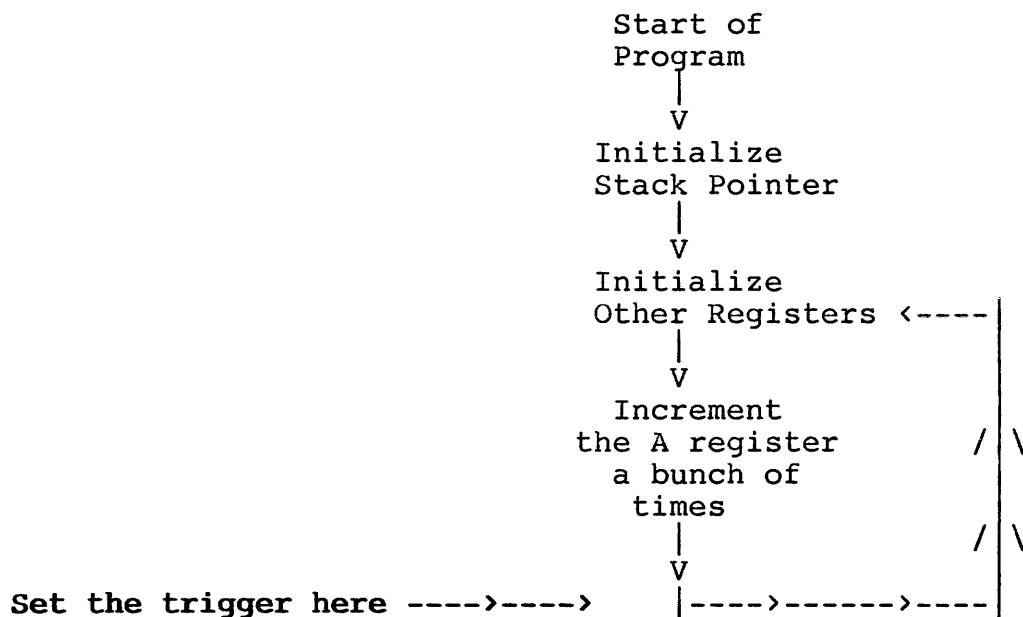
It's easy to understand the test program loaded into the UniLab's memory by the **LTARG** command. It initializes some registers, and then goes into an infinite loop.



-- Set Trigger --

The program that you work on will, of course, be more complicated. But no matter how complicated or simple your program, you can always tell the UniLab to trigger on the address of an instruction.

There is only one mildly interesting point in the test program for the Z80. That is the unconditional jump at address 29, that jumps back to address 3.



-- Set Trigger --

To get a trace starting at that address, type in:

NORMT 29 ADR S

NORMT clears out all previous trigger specifications, and tells the UniLab that you want the trigger event at the **Top** of the trace.

29 ADR is the trigger specification

S starts the analyzer

Which results in the following display (with MISC, HDATA and CONT columns removed for the sake of simplicity):

resetting

cy#	ADR	DATA	
-5	0024	3C	INC A
-4	0025	3C	INC A
-3	0026	3C	INC A
-2	0027	3C	INC A
-1	0028	3C	INC A
0	0029	C30300	JP 3 <-----<----- Here is the trigger
3	0003	3E12	LD A,12
5	0005	015634	LD BC,3456
8	0008	119A78	LD DE,789A
B	000B	21DEBC	LD HL,BCDE
E	000E	C5	PUSH BC
F	18FF	34	write
10	18FE	56	write
11	000F	C1	POP BC
12	18FE	56	read
13	18FF	34	read
14	0010	3C	INC A
15	0011	3C	INC A
16	0012	3C	INC A
17	0013	3C	INC A
18	0014	3C	INC A

-- In Detail --

6-70

-- Set Trigger --

Cycle numbers

The analyzer found the trigger event, and then sent to the host computer a record of bus activity starting five cycles before the trigger. The trigger event is labeled as cycle 0, the cycles before it have negative numbers.

Explanation

The rest of the trace is fairly simple-- and very similar to the display that results from **STARTUP** with the Z80 test program.

There are only two mysteries to clear up, before continuing the discussion of trigger specifications:

- 1) what **NORMT** does
- 2) the meaning of the "resetting" message that appears just before the trigger display

-- Set Trigger --

4.4 The NORMx Words

The three NORMx commands, **NORMT**, **NORMM**, and **NORMB**, first clear out all previous trigger settings. They wipe the slate clean.

And then each one sets up the "display window" to show a different time portion of the program's execution. The diagram below shows the effect of each instruction on a program that, rather boringly, executes instructions starting at address 0 without any jumps or calls or branches:

	<u>NORMB</u>	<u>NORMT</u>	<u>NORMM</u>	<u>NORMT</u>
	<u>128</u> <u>ADR</u> <u>S</u>	<u>128</u> <u>ADR</u> <u>S</u>	<u>128</u> <u>ADR</u> <u>S</u>	<u>128</u> <u>ADR</u> <u>S</u>
	<u>200</u> <u>DCYCLES</u>			<u>200</u> <u>DCYCLES</u>
	0	0	0	0
	:	:	:	:
	cy# adr	:	:	:
	:	:	:	:
	-A5 83	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	cy# adr	:
	:	:	:	:
	:	cy# adr	-54 D3	:
	:	:	:	:
	:	-9 11A	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
'RIGGER-->	0 128	0 128	0 128	128 <--TRIGGER
	:	:	:	:
	4 12C	:	:	:
	:	:	:	:
	:	:	55 17D	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	cy# adr
	:	:	:	:
	:	:	:	157 27F
	:	:	:	:
	:	:	:	:
	:	:	:	:
	:	:	:	:

-- Set Trigger --

All four displays have the same trigger, and all of them number the trigger cycle as cycle zero. They vary only in the value of **DCYCLES**. The **Delay CYCLES** is the number of cycles that will pass between when the trigger is seen, and when the buffer is frozen. If this value is small (as happens when you use **NORMB**), then most of the trace buffer will show what happened before the trigger.

The fourth example shows how you can manually set the delay count. Here the delay is so large that the trigger is not even in the window. This example uses the **NORMT** command to clear out the previous trigger spec, but then uses **DCYCLES** to change the delay count.

Notice how the **NORMx** commands change the value of **DCYCLES** in the following:

```
NORMT  
TSTAT  
Analyzer Trigger Status:  
RESET  
A0 DCYCLES  0 QUALIFIERS
```

```
NORMM  
TSTAT  
Analyzer Trigger Status:  
RESET  
55 DCYCLES  0 QUALIFIERS
```

```
NORMB  
TSTAT  
Analyzer Trigger Status:  
RESET  
4 DCYCLES  0 QUALIFIERS
```

Summary

The first address you see on the trace display after you start the analyzer with the S command depends on two things:

- 1) The trigger address you selected with **ADR**
- 2) The delay you selected with **NORMT**, **NORMM**, **NORMB**, or **DCYCLES**.

-- Set Trigger --

4.5 RESETTING-- Restarting the target program

The UniLab software usually responds with "resetting" when you start up the analyzer. This message lets you know that the UniLab is sending a reset signal to your processor, causing it to start executing your program from the beginning.

Whenever you start up the analyzer with **S**, you can either

start analyzing the program running on the target board, starting from whatever point the program has reached (**RESET'**)

OR

restart the program at the same time as you restart the analyzer (**RESET**).

Turning RESET on and off

The **RESET** feature gets turned on by **STARTUP**. You can turn it on and off yourself with the commands **RESET** and **RESET'**, or with the mode panel (function key 8).

Mode Panel:

1. ANALYZER modes
DISASSEMBLER on
SYMBOLS off
RESET enabled

-- Set Trigger --

4.6 General Purpose Trigger Definitions

While the previous examples were limited to address triggers for simplicity, the UniLab allows much more complex triggers to be defined, using all 48 analyzer inputs.

Each of the groupings of inputs can be 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.

Just as we used

<16 bit value> **ADR**

to define a single address trigger, we can define triggers for the other input bytes:

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

-- Set Trigger --

Modifying the input group words

All of the input group words can be altered in several different ways, by preceding them with keywords. You can also combine several input group words, as detailed on the next page.

Alone

enter a single number to trigger on a single value, **12 DATA**

Using NOT and TO

enter a range separated by **TO**, to trigger on a range of values, **12 TO 34 DATA**

enter the command **NOT** to trigger on anything but the value that follows, **NOT 10 DATA**

use both **NOT** and **TO** to trigger outside of a range of values **NOT 10 TO 13 DATA**

Using MASK

or, most complexly, use the **MASK** command to ignore certain input lines while triggering on other lines. The following is identical to **10 TO 13 DATA: FC MASK 10 DATA**

Or, in binary: **B# 11111100 MASK B# 00010000 DATA**

Which tells the UniLab that we are only interested in the values of the first six wires

1111 1100

and that on those wires we want to see the signals

0001 00

Since we don't care what the lowest two bits are, they can be any value-- 00 or 01 or 10 or 11.

A Note on scope

These three words, **NOT**, **TO**, and **MASK**, only affect the first input group word which follows.

For example:

NORMT NOT 12 DATA 400 ADR S

will trigger when the data is not 12 and the address is 400.

-- In Detail --

-- Set Trigger --

Triggering on combinations

You can set a trigger on several different input bytes, and the UniLab will search for a bus cycle that satisfies all the conditions you describe. If you want to search for 12 on the data lines AND for 100 on the address lines, all you have to do is type in the command:

12 DATA 100 ADR

Using ALSO

However, declaring a trigger for any group of inputs will clear the previous settings. If you want to search for 12 on the data lines OR 15 on the data lines, you have to use **ALSO**:

12 DATA ALSO 15 DATA

which tells the UniLab to trigger on either 12 or 15. If you had left out the ALSO and just entered 12 DATA 15 DATA, then the UniLab would watch the data lines for only one value, 15, the last number specified.

When you make a new description for an input group without **ALSO**, you clear out the previous trigger for that group, without affecting the other groups. For example, if you enter

NORM 123 ADR 45 DATA S

trigger will occur only when the data is 45 during a bus cycle with 123 address. If you then enter

60 TO 71 DATA S

the analyzer will restart and trigger will occur when data is between 60 and 71 during a bus cycle with 123 address. You have altered the **DATA** specification, but not the **ADR** spec.

-- Set Trigger --

4.7 Real-life Examples:

Catching the program when it goes outside of program memory

One of the nastiest problems you encounter while checking out hardware or software is when your program "blows up" and begins executing data or garbage.

These errors not only are troublesome to recover from, but the mistake that caused the blow up is almost impossible to find - until now. Trapping these problems is a pleasure with the UniLab.

If, for example, your program is supposed to be limited to addresses 0 to 1234, you can enter

RESET NORMB NOT 0 TO 1234 ADR S

The UniLab will reset the target system and wait for the target program to access an address outside of the specified range. You can then look back through the trace memory for the abnormal operation which caused the program to "blow up."

Whether it is a hardware malfunction or a software bug you will have trapped it effortlessly.

You can add **FETCH** to the above example, so that the UniLab doesn't trigger on reads and writes outside of memory. Some processors lack the fetch indicator.

Catching garbage values being written to a single memory location

Another common bug you encounter is when some location in RAM gets accidentally overwritten.

For example, a variable called **STRING_LEN** gets written with the length of a string. But when your program reads the value, it isn't the same as the value written into it.

One way to catch this bug is to produce a filtered trace that shows every access to this variable, and the cycles that immediately follow the access. You can then examine the trace, and find the region of the program that causes the overwrite:

2AFTER STRING_LEN ADR S

You can then trigger on the address of the bad instruction:

NORMM <address> ADR S

-- In Detail --

-- Set Trigger --

Catching a stack overflow

You can set the UniLab to trigger when your stack grows too large.

For example, a target board has ROM at locations 0 to 1FFF, and RAM at 2000 to 3FFF. The program sets the stack pointer to address 20FF in RAM. This means the stack can grow to FF bytes before running into ROM.

You can tell the UniLab to trigger when the program makes reference to some address that the stack will write to when it grows "too large"-- whatever too large means to you.

Some might want to wait until the stack is about to run into hardware limitations:

NORMB 2001 ADR S

Others will want to trigger when the stack holds more than 2F bytes:

NORMB 20D0 ADR S

Either way, you get to see what the program was doing just before the stack grew too large.

Catching bad data going into a string

You can use a combination of a range of data and a range of addresses to catch the trace of a bug that causes an inappropriate character to be written into a string.

Of course, you don't want to look at every access to the memory locations-- you just want to see when the bad data comes in.

Suppose the string sits at a location with the symbolic name STRING1 and has a length of 50 (hex) characters. The string should only contain characters between A (41 hex) and z (7A hex).

The instruction:

NORMM NOT 41 TO 7A DATA STRING1 TO STRING1 4F + ADR S

will cause the UniLab to trigger when any data outside the range 41 to 7A gets written to any of the 50 data locations starting at STRING1.

-- Set Trigger --

4.8 The Limits of Trigger Complexity

Since the UniLab trigger logic uses high speed truth tables instead of comparators, there is no limit to the complexity of triggers within bytes. For example:

12 DATA ALSO 34 DATA ALSO C0 TO C5 DATA ALSO FF DATA

is perfectly acceptable.

Another way to state the same thing is by entering:

12 34 C0 C1 C2 C3 C4 C5 FF 9 NDATA

Note that the 9 is the number of terms listed.

ALSO with ADR

You can run into problems with **ADR**, since that word actually describes two bytes. If the high byte of several addresses that you are using **ALSO** on don't match, you can produce unanticipated cross products. For example:

1200 ADR ALSO 1535 ADR

would cause the UniLab to trigger on either **1200** or 1535-- and also on either 1235 or 1500

These cross products usually are not a problem, but you should be aware of them.

-- Set Trigger --

MASKing

You can also specify triggers with a MASK format. For example,

80 MASK 0 DATA

requires the MSB of the data bus to be 0, but doesn't care about the other 7 bits. It is identical to entering **0 TO 7F DATA** or **NOT 80 TO FF DATA**. All three commands give the same result so you should simply use the format that seems most natural to you.

Triggering on 20-bit addresses

If your system uses more than 16 bits for addressing, you can set triggers on 5-digit hex addresses by ending the address with a period. For example, **12345.ADR** will actually set a trigger on 1 in the right digit of the CONT column (which is connected to address bits A16-A19) and 2345 on the ADR inputs.

-- Set Trigger --

4.9 Filtered Traces

The UniLab's trace buffer stores 170 48-bit samples of bus activity. Other analyzers need gigantic trace buffers because they lack the sophisticated triggering and filtering logic of the UniLab.

Often the majority of bus cycles are not of interest-- for example when most of the time is spent in a status loop or a delay loop.

The sledgehammer solution: have a huge trace buffer. Then you get to look through that buffer, hunting for the relevant information.

The UniLab approach: have the computer throw away the boring parts of the program.

With the UniLab you never have to look through thousands of uninteresting cycles. The UniLab will filter the trace, and record only the cycles that interest you.

An introduction to ONLY

If you enter:

ONLY 1234 ADR S

the UniLab will record only cycles that address location 1234. If the instruction at 1234 is the one that reads input samples, you will end up with a trace recording of nothing but input samples.

Filter, excluding addresses

More practically, suppose that a boring status loop occupies program memory from 1020 to 1060. You want to get a trace that does not include the trace of the opcodes in those addresses. The command is:

ONLY NOT 1020 TO 1060 ADR S

-- Set Trigger --

Filter the trace, but don't start until AFTER . . .

You can make a filtered trace even more useful by setting up a separate trigger that tells the UniLab when to start checking cycles against the filter specification. For example, the program might not get interesting until after 30 gets written to address 3000 of RAM:

ONLY NOT 1020 TO 1060 ADR **AFTER 30 DATA 3000 ADR** S

Further discussion of **AFTER** is deferred to the following subsection 4.10 on **Qualifying Events**.

The rest of the filter commands

ONLY is most useful when you want to exclude some type of operation or some section of the program.

But when you filter to include cycles, you usually want to see at least one cycle after the trigger.

For example, if you are looking at all the writes to RAM, you can find out which section of the program performed the write with

3AFTER WRITE S

which will show you every write along with the three cycles that follow it.

2AFTER captures the two cycles that follow each trigger, and **1AFTER** captures only one cycle after each trigger event.

Filtering and disassembly

Since filtering will produce a trace with partial opcodes, the disassembler will not be able to interpret the sequence of cycles properly. You will probably want to turn off the disassembler when producing a filtered trace. Use **DASM'** or the mode panel (**F8**).

Mode Panel:

1. ANALYZER modes
DISASSEMBLER on
SYMBOLS off
RESET enabled

-- Set Trigger --

Filtering and the MISC inputs

The filtering logic of the UniLab does not look at the MISC inputs. This lets you tell the UniLab to filter a trace while waiting for a trigger condition to appear on the MISC inputs.

This is not the same as using **AFTER** with the filter commands-- with **AFTER** you get a filtered trace starting at some bus event. With the use of the MISC lines, you can get a trace that shows the bus activity before some event.

For example, if you want a trace with the delay subroutine at A0-B0 removed, but you want to trigger on an active high error signal, you connect the error signal to one of the MISC inputs and enter:

ONLY NOT A0 TO B0 ADR FF MISC

The filtered trace will exclude cycles accessing addresses A0 to B0, but trigger will not occur until the error input goes true, thus causing FF on the MISC inputs.

-- In Detail --

-- Set Trigger --

4.10 Qualifying Events

The UniLab can trigger on sequences of events, instead of just when it sees a single trigger event. For example,

NORMT 78 DATA AFTER 56 DATA S

will not trigger until first 56 appears on the data bus and then, anytime later, 78 appears.

The 56 is the qualifier, and 78 the trigger.

Up to three qualifiers

You can specify up to three sequential qualifying events. Use **AFTER** when you want to start the description of the next qualifier. For example:

NORMT 10 DATA AFTER 250 ADR AFTER 300 ADR S

will trigger on 10 data, anytime after 300 is immediately followed by 250 on the address bus.

The UniLab will not start to search for the trigger itself until after it sees the qualifiers.

The qualifiers must appear on the bus without any intervening bus cycles. If the sequence does not appear, then the UniLab starts searching for the first qualifier. However, once all the qualifiers have shown up, the trigger does not have to occur immediately.

You can specify a minimum number of bus cycles after the time the last qualifier is seen, before the UniLab starts looking for the trigger. The default is **0 PCYCLES**. You can also specify a number of complete repetitions of the sequence of qualifiers. The default is **1 PEVENTS**.

See the flowchart on the next page.

-- Set Trigger --

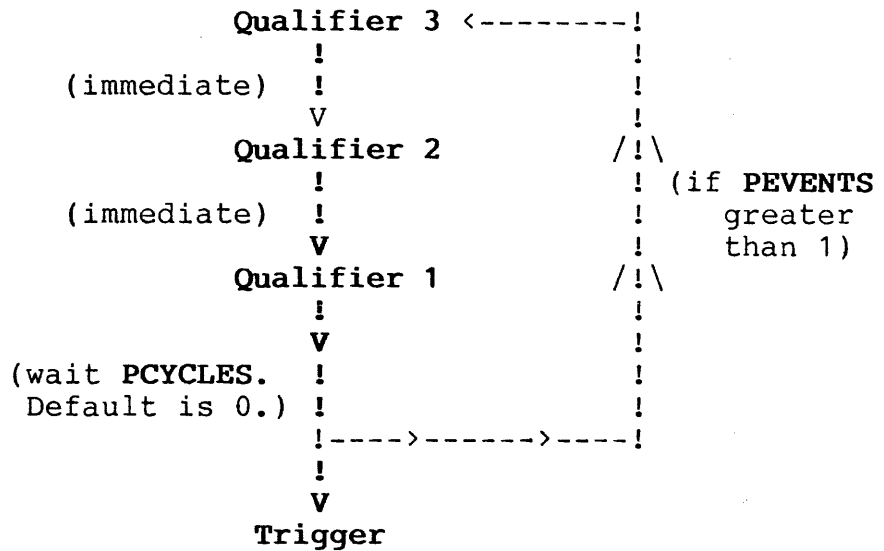
The big picture

When you start up the analyzer, the UniLab will first search for qualifier #3, then qualifier #2, and then qualifier #1.

After that, the UniLab waits until **PCYCLES** pass. Usually this value will be zero.

Then the UniLab will check whether it has gone through the qualifier sequence enough times. You specify this with <value> **PEVENTS**.

If it requires more qualifier sequences, the UniLab will start searching for qualifier #3 again. Otherwise, the UniLab will start searching for the trigger itself.



-- Set Trigger --

Triggering for a filtered trace

Qualifiers also allow you to set up a trigger that is different from the filter specification. That way you can produce a filtered trace that starts after the qualifiers:

ONLY NOT 200 TO 250 ADR AFTER 368 ADR S

This trigger specification will make a filtered trace that excludes addresses 200 through 250. The UniLab will not start making the filtered trace until it sees address 368 on the bus.

Triggering on sequential events

You can use qualifiers to trigger on a consecutive sequence. Suppose there is a conditional branch at address 1010, which will jump to address 250. Other instructions also cause a jump to 250, but you are not interested in those. You want to see what happened before the branch at address 1010 is taken-- so you want to trigger when 250 follows immediately after address 1010.

Using address 1010 as the qualifier and then 250 as the trigger will not work, because the UniLab would trigger on address 250 even if it occurred hours after address 1010. Instead, you want to have both addresses as qualifiers, and no trigger event:

NORMB AFTER 250 ADR AFTER 1010 ADR S

Watch out

Note that the qualifiers must always appear one immediately after another on the target system bus:

NORMT 10 DATA AFTER 250 ADR AFTER 300 ADR S

In this example, repeated from the previous page, as soon as the UniLab sees 300 on its address inputs, it will look for address 250. If that value does not appear on the address bus immediately after 300, then the UniLab will go back to searching for 300.

That particular trigger only makes sense if there is a jump, call or conditional jump at 300, that could cause the next address to be 250.

A RETURN from 300 to 250 would not qualify, because the address would have to be pulled off the stack, and so several bus cycles would appear between address 300 and address 250.

-- Set Trigger --

Delay between qualifiers and trigger

Though the qualifiers must follow one after another, the trigger can come anytime after the qualifiers.

In fact, you can specify a minimum length to the delay between the qualifiers and the trigger. This is useful for avoiding trigger immediately after the qualifiers are seen.

If you want to keep the trigger disabled for 200 cycles after the qualifying sequence you can simply enter

200 PCYCLES

This is the pass count.

For example:

NORMB 10 DATA AFTER 250 ADR 200 PCYCLES S

tells the UniLab to trigger when the data is ten. The UniLab will not start to search for data 10 until 200 bus cycles after the address appears on the bus.

Repetition of the qualifying events

You can also specify to the UniLab that the sequence of qualifying events be repeated. This is useful for looking at the nth pass through a loop, or the nth call to some routine.

If you want the UniLab to wait for 150 complete repetitions of the qualifiers before starting to search for the trigger enter

150 PEVENTS

For example:

NORMT AFTER 1100 ADR 150 PEVENTS S

causes the UniLab to trigger after address 1100 has appeared on the bus 150 times.

-- Set Trigger --

4.11 Stepwise refinement

The UniLab allows you to build on existing trigger definitions.

Trigger definitions can be gradually expanded in complexity as you find limitations in your original idea. If you are trying to see a subroutine at address 1200, that gets called from a certain section of code, you might first enter

NORMB 1200 ADR S

only to find that the trace shows a call to the subroutine from a section of the program that you are not interested in.

You can add a qualifier and restart the analyzer by entering

AFTER 5670 ADR S

This time the UniLab will search for 1200 only after address 5670 (an address in the desired calling routine) has been detected.

If you had thought of the need for a qualifier in the first place, you could have entered

NORM 1200 ADR AFTER 5600 ADR S

This ability to polish trigger definitions makes your interaction with the UniLab conversational. You ask questions about what the system is doing and receive immediate answers -- all from the same keyboard you use to write and change the programs.

5. Saving Information

Introduction

The UniLab software lets you save transcripts of your sessions, and also lets you save specific information as encoded DOS files.

Contents

5.1	Feature Summary	6-91
5.2	Overview	6-92
5.3	Screen History	6-93
5.4	Save Record of Session to Text File	6-94
5.5	Save Record of Session to Printer	6-95
5.6	Save Only Memory Changes to Printer	6-95
5.7	Save Trace	6-96
5.8	Save Symbol Table	6-98
5.9	Save a Range of Memory	6-98
5.10	Save the State of UniLab Software	6-99

5.1 Feature Summary

<u>Features</u>	<u>Mode Panel</u>	<u>Commands</u>
Print out commands that alter memory	Yes	LOG LOG'
Send all screen display to DOS file	Yes	TOFILE TOFILE'
Print out everything	Yes	PRINT PRINT'
Save a trace to a file	NO	TSAVE <file>
Compare current trace to one saved as a file	NO	<n> TSAVE <file>
Save symbol table to a file	NO	SYMSAVE <file>
Save current state of UniLab program	NO	SAVE-SYS <file>
Save memory to a file	NO	<from> <to> BINSAVE <file>
Look at one line of "screen history"	NO	Up Arrow key
Look at one page of history	NO	PgUp key

Mode Panel:

3. LOG modes
LOG TO PRINT
LOG TO FILE
PRINTER

Commands:

LOG LOG'
TOFILE TOFILE'
PRINT PRINT'

-- Saving Information --

5.2 Overview

While using the UniLab software, you can preserve any information you want about your session.

The software always preserves a history of your screen. You can save up to 60K in this history, which starts up every time you begin a session with the UniLab. After the history buffer fills, you start losing the oldest information.

You can also turn on features that will save all screen displays

to a text file **TOFILE**
or to your printer. **PRINT**

You can also log only memory changes to the printer. **LOG**

Other commands save, as DOS files:

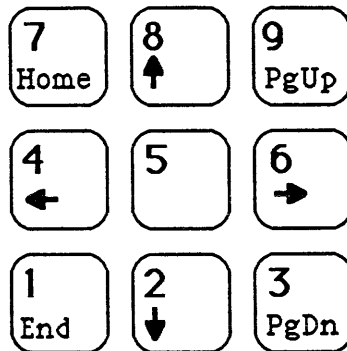
the current trace display, **TSAVE**
the current symbol table, **SYMSAVE**
any range of memory, **BINSAVE**
or the current state of the system. **SAVE**

5.3 Screen History

The screen history always preserves the last 20 to 60K of screen display.

The information that scrolls off the top of either the full screen or the lower window gets saved.

You look at the history with the Up Arrow and Pg Up keys, numbers 8 and 9 on the numeric key pad.



Setting the size of screen history

The size of the history is set by giving the command:

`<hex # of Kbytes> =HISTORY`

then saving the newly altered UniLab software with **SAVE-SYS**. You must exit the program with **BYE** and start it again.

The size of the history buffer is allocated when the program starts up, and cannot be changed on the fly.

The maximum number of kilobytes that you can allocate to history is 3C (decimal 60K).

Use the command **?FREE** to find out how many bytes are allocated to history and to symbols. That display appears in decimal base, not hexadecimal.

-- Saving Information --

5.4 Save Record of Session to a Text File

You can save the record of a session with a text log file. You can only save to one text file per session, but once you have created a log file you can turn the logging on and off at will, with the mode panel or with **TOFILE** and **TOFILE'**:

Mode Panel:

```
3. LOG modes
LOG TO PRINT inactive
LOG TO FILE off
PRINTER off
NMI VECTOR active
SWI VECTOR active
```

Creating the log file

You cannot create the file from the mode panel-- you must use the command

```
TOFILE <file name>
```

to create the file in the first place.

This command can be used as a "command tail" when you call up the UniLab software from DOS:

```
A> ULZ80 TOFILE JUNE3
```

will call up the UniLab program with "june3" as the log file.

You can also name the log file from within the UniLab program, with **TOFILE** <file name>.

You will not be able to turn on logging to a file until you have named a file.

5.5 Save Record of Session to a Printer

You can save all screen output to a printer with the Mode Panel or with the **PRINT** and **PRINT'** commands.

Mode Panel:

```
3. LOG modes
LOG TO PRINT inactive
LOG TO FILE  off
PRINTER    off
NMI VECTOR  active
SWI VECTOR  active
```

5.6 Save Only Memory Changes to Printer

This feature helps make certain that you don't forget any patches that you make to your program. It keeps a record on your printer of all commands that alter memory.

You turn it on and off with the Mode Panel, or with the commands **LOG** and **LOG'**.

Mode Panels:

```
3. LOG modes
LOG TO PRINT inactive
LOG TO FILE  off
PRINTER     off
NMI VECTOR  active
SWI VECTOR  active
```

-- Saving Information --

5.7 Save and Compare Trace

You can save the trace as an encoded file, which can later be retrieved with **TSHOW** or compared to the current trace with **TCOMP**.

You save the current trace with the command:

TSAVE <file name>

This is very useful for production checkout of systems. You can save the trace of a program on a known good system, and then use **TCOMP** to compare the known good trace to the trace of the hardware you want to check.

Comparing traces

Enter **AA TCOMP** <file name> to compare the last AA cycles of the trace (which is the whole trace) currently in the trace buffer with the trace previously saved as a file.

If the two traces are identical, the UniLab will respond with an "OK" message. Otherwise it will display 14 lines of the trace on disk including the first non-matching bus cycle, followed by the non-matching cycle in the current trace.

You can completely test a system with a few such trace comparisons, using programs that exercise the hardware of the system. The UniLab's macro capability allows you to write a macro which completely tests a system, automatically. (UniLabs are given their final test at the factory with just such a macro.)

-- In Detail --

Example: Trace compare

This example shows the result of performing **TCOMP** on a faulty trace produced by a Z80 board running the simple target program (**LTARG**). One of the address lines of the board was grounded, which pulled it low.

AA TCOMP TESTZ80.TRC

cy#	CONT	ADR	DATA		HDATA	MISC
3	B7	0003	3E12	LD A,12	11111111	11111111
5	B7	0005	015634	LD BC,3456	11111111	11111111
8	B7	0008	119A78	LD DE,789A	11111111	11111111
B	B7	000B	21DEBC	LD HL,BCDE	11111111	11111111
E	B7	000E	C5	PUSH BC	11111111	11111111
F	D7	18FF	34	write	11111111	11111111
10	D7	18FE	56	write	11111111	11111111
11	B7	000F	C1	POP BC	11111111	11111111
12	F7	18FE	56	read	11111111	11111111
13	F7	18FF	34	read	11111111	11111111
14	B7	0010	3C	INC A	11111111	11111111
15	B7	0011	3C	INC A	11111111	11111111
16	B7	0012	3C	INC A	11111111	11111111

No Good! (Above is correct.) Was:

cy#	CONT	ADR	DATA		HDATA	MISC
F	D7	18DF	34	write	11111111	11111111

TCOMP reports a discrepancy to you by showing the relevant section of the trace on disk, and then showing the non-matching line from the current trace.

You then have to perform a visual comparison of the two cycles that don't match up.

In this case, you can see that in cycle F of the trace on disk, the Z80 wrote to address 18FF. In that same cycle of the new trace, the Z80 wrote to 18DF. Since the program is the same in both cases, the difference is in the hardware.

ADDRESS LINE: 7654 3210
 FF hexadecimal is 1111 1111 binary,
 while DF hexadecimal is 1101 1111 binary.

So, obviously, address line A5 has been accidentally grounded.

-- Saving Information --

5.8 Save Symbol Table as DOS File

You can save the symbol table as an encoded file, which can later be retrieved with **SYMLOAD**.

You save the current symbol table with the command:

SYMSAVE <file name>

5.9 Save a Range of Memory as DOS file

You can save any range of emulation ROM as an encoded file, which can later be retrieved with **BINLOAD**. You can use this command to save the program you are working on.

You can also save from and load to RAM if you have first established debug control. See section 6 on Breakpoints and The Debugger.

You save a range of memory with the command:

<from address> <to address> **BINSAVE** <file name>

5.10 Save the State of the UniLab Software

You can save the current state of the UniLab program to a command file. This allows you to save the software with a certain range of memory enabled, and with other variables set up to your preference. Saving the system will also preserve the current trace.

You can save to a file with the same name as the current **.COM** file, or to a different one.

To save the current state of the system, use the command:

SAVE-SYS <file name>

Unless you specify a different path, the file will get saved to the Orion directory.

6. Breakpoints and the Debugger

Introduction

The UniLab emulator includes special hardware that makes possible virtually all of the traditional processor-pod development system features. The basic UniLab software includes all of the processor-independent debug features.

Processor-specific software packages add more features, such as the ability to change specific registers, or take advantage of special functions of the processor.

Contents

6.1	Feature Summary	6-101
6.2	Overview	6-103
6.3	Establish Debug Control	6-104
6.4	Interpret the Breakpoint Display	6-109
6.5	Within the Debugger	6-111
6.6	"Trigger," style breakpoints	6-121
6.7	Exit from the Debugger	6-122
6.8	Disable Debugger-- How and Why	6-124

6.1 Feature Summary

<u>Feature</u>	<u>Menu</u>	<u>Command</u>
To enter debugger:		
Establish debug control	Yes	RESET <addr> RB
Gain debug control without setting a breakpoint. Not supported on all processors.	NO	NMI
Within debugger:		
All commands for reading and altering memory work on RAM.		
Resume execution to a breakpoint	Yes	<addr> RB
Set breakpoint at next code address	Yes	N
Show the "breakpoint display" again	NO	R
Execute the next instruction-- use when single stepping for jumps and branches. Not supported on all processors.	NO	SSTEP
Alter Program Counter, then resume to breakpoint	Yes	<New PC> <addr> GB
Set Multiple Breakpoints	NO	<addr> <bp #> SMBP
Clear one multiple breakpoint	NO	<bp #> RMBP
Clear all multiple breakpoints	NO	CLRMBP
Trigger style breakpoints (Not supported on all processors):		
Set up a trigger for debugger	NO	RI <trigger spec>
Start program and gain debug control when trigger seen on bus	NO	SI
To exit debugger:		
Exit immediately-- set program running again	NO	RZ
Alter Program Counter, then exit from debug control	NO	<New PC> G
Alter Program Counter, then wait for the analyzer to start	NO	<New PC> GW
Additional (target specific) debugger commands alter register contents, output values to ports, etc.		

-- The Debugger --

Command:

Menu:

		DEBUG MENU	
RESET <addr> RB	F1	SET A BREAKPOINT TO ESTABLISH DEBUG CONTROL	
<addr> RB	F2	RESUME EXECUTION TO A BREAKPOINT	
N	F3	EXECUTE THE NEXT STEP (WON'T FOLLOW JUMPS)	
<New PC> <addr> GB	F4	GO TO AN ADDRESS WITH A BREAKPOINT SET	
<New PC> G	F5	GO TO AN ADDRESS AND EXIT THE DEBUGGER	
	F10	RETURN TO MAIN MENU	

6.2 Overview

The debugger commands of the UniLab software provide you with the tools traditionally associated with development systems.

With the debugger commands, you can

- set single or multiple breakpoints,
- single step through code,
- read and alter internal registers.

Also, once you have established debug control, you can read and alter RAM using any of the commands that access memory (see section three of this chapter).

The UniLab's powerful bus state analyzer replaces most of the functions of the traditional debugging tools. But when you have tracked a bug down to a small segment of code, it is handy to be able to set a breakpoint and single step through the program.

-- The Debugger --

6.3 Establish Debug Control

Most of the debugger commands will not work until after the UniLab's special debugger hardware has taken control of your processor.

You can establish debug control with either **RESET <addr> RB**, or, if it is supported by your processor, with **NMI**.

You cannot invoke the debugger until after your program initializes the stack pointer. The debugger actually runs code on your processor, and then uses the **RETurn** instruction to resume execution of your program.

If the stack pointer is not initialized, you will not be able to establish debug control at all.

Run to a breakpoint

You can invoke the debugger by setting a breakpoint at a particular code address.

RESET <address> RB

The address you give must be the first address of an opcode. In the fragment of Z80 code below, you could set a breakpoint at any of the addresses that appear in the **adr** column. But you would not be able to set a breakpoint on 131 or 132, for example.

Adr	Opcode	Instruction
012B	012C00	LD BC,2C
012E	7C	LD A,H
012F	BA	CP D
0130	C23801	JP NZ,138
0133	7D	LD A,L
0134	BB	CP E
0135	CA4201	JP Z,142
0138	7E	LD A,(HL)

Use a trace display, or disassemble from memory (with **DN** or **DM**) to determine what addresses you can use for breakpoints.

Establish control with NMI

Using the Non-Maskable Interrupt (**NMI**) command is the other way to gain debug control. This command will only work if

1) your processor was designed with a pin that lets you give a non-maskable interrupt signal or some equivalent feature,

and

2) your hardware does not make use of that feature.

The **NMI** command of the UniLab software sends a signal to the NMI pin of your processor, which interrupts your program-- no matter what it happens to be doing.

Of course, your program has to be running before **NMI** can interrupt it, and your program must initialize the stack pointer or the Orion dbugger will not work.

If you don't know whether your processor supports NMI, look at Appendix H: Processor Features.

-- The Debugger --

Commands dependent on NMI

NMI is used by the **SSTEP** and **SI** features. If your processor does not have a non-maskable interrupt feature, then the UniLab software does not support the commands **NMI**, **SSTEP**, nor the **RI** & **SI** combination.

Of course, if you disable **NMI**, the features that make use of it will not work.

Disabling NMI

If your hardware does make use of the NMI feature of your processor, you will need to disable the UniLab software's use of that feature. Disable the debugger's use of the NMI feature of your processor with either the Mode Panel option "NMI VECTOR" or the command **NMIVEC'**.

Whichever you use, the result is the same-- **NMI** and the commands dependent upon it no longer work, but the rest of the debugger commands, such as **RB** and **GW**, work fine.

However, you can turn off all debugger commands, including **NMI**, with either the mode panel option "SWI VECTOR" or the command **RSP'**.

Mode Panel:

```
3. LOG modes
LOG TO PRINT inactive
LOG TO FILE off
PRINTER off
NMI VECTOR active
SWI VECTOR active
```

Common pitfalls

1. Watchdog Timer:

Your microprocessor stops executing your program when you are at a breakpoint.

If you have a watchdog timer, it will then try to restart your target board. The watchdog thinks that something has gone wrong with your program.

You must disable the watchdog timer to use the debugger.

2. Stack Pointer:

The Orion overlay routines make use of your processor's stack. You cannot set a breakpoint until after your program initializes the stack pointer. Most programs initialize the stack pointer as one of the first few steps.

3. Opcode Address:

You can only set a breakpoint on the first address of an instruction, as explained on the previous page.

4. Reserved bytes and the Overlay Area:

Your program cannot make use of the reserved bytes, and you cannot set a breakpoint in the overlay area.

The addresses of the reserved bytes and the overlay area appear in Appendix H, or hit **CTRL-F3**. The reserved area is between one and six bytes of ROM, and the overlay area is the area of 30 to 70 bytes above the reserved bytes.

Debug commands referencing addresses in the overlay area may produce strange results. The safest practice is to not have any code in there at all.

-- The Debugger --

Example: Establish debug control with RB

When your program reaches a breakpoint, the UniLab takes control of your processor. Your program actually stops executing.

Your processor executes an Orion "overlay routine" that results in a display of your processor's internal registers. The display you get will depend on your processor. We can set a breakpoint in the following Z80 code:

Adr	Opcode	Instruction
012B	012C00	LD BC,2C
012E	7C	LD A,H
012F	BA	CP D
0130	C23801	JP NZ,138
0133	7D	LD A,L
0134	BB	CP E
0135	CA4201	JP Z,142
0138	7E	LD A,(HL)

In this example, we set a breakpoint on address 12F. The UniLab replies with a "resetting" message, to let us know that it is restarting the target board's program. When the processor reaches the breakpoint, the register display appears.

RESET 12F RB resetting

AF=02A0 (Sz-a-pnc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFF SP=1C00 PC=012F
012F BA CP D (next step) ok

-- In Detail --

6.4 Interpreting the Breakpoint Display

The breakpoint display varies from processor to processor, but always contains the same two basic parts:

The register display

AF=02A0 (Sz-a-pnc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=012F

and the display of the next step.

012F BA CP D (next step) ok

Register display

You can show the breakpoint display again with the command **R**.

The register display varies from processor to processor, but always includes the stack pointer (**SP**), the program counter (**PC**), and the flags register (**F**).

AF=02A0 (Sz-a-pnc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=012F

All registers are displayed in hexadecimal, and a single letter abbreviation for each flag is also displayed.

AF=02A0 (**Sz-a-pnc**) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=012F

Notice how the flags display changes as the value stored in the F register changes-- a capital letter indicates that the value is high, a lowercase letter indicates that it is low:

AF=02A0 (Sz-a-pnc)

AF=0242 (sZ-a-pNc)

-- The Debugger --

Next step

The display of the next step shows you the **address** of the opcode which will execute next,

AF=02A0 (Sz-a-pnc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFF SP=1C00 PC=012F
012F BA CP D (next step) ok

and the **opcode** stored at that address,

012F BA CP D (next step) ok

and the **disassembled instruction** that the processor is about to execute:

012F BA CP D (next step) ok

The address of the next instruction is, of course, the same as the address contained in the Program Counter or Instruction Pointer register.

6.5 Within the Debugger

Once you have established debug control, you can:

- run to another breakpoint,
- single step,
- follow jumps while single stepping,
- change the program counter and then run to a bp,
- set multiple breakpoints,
- examine and alter internal registers,
- and examine and alter RAM or ROM.

You can also exit from the debugger, and start using analyzer commands again.

Since your processor is idling, you can safely look at or change emulated ROM without crashing the program.

If your processor supports **NMI**, then it also supports analyzer trigger-style breakpoints, as described in the next subsection, 6.6.

On some processors, you can also send data to a port, and examine the contents of a port.

Missed breakpoints

You will lose debug control if you use **RB** or **GB** to set a breakpoint which your program never reaches.

When this occurs you can press any key and **NMI** will be executed, regaining debug control-- but only if your processor supports **NMI** (see Appendix H). Otherwise, the only way to again establish debug control is with **RESET <addr> RB**.

If you accidentally set a breakpoint in the middle of an instruction, it will probably crash your program, in which case **RESET <addr> RB** will be the only way to regain debug control.

-- The Debugger --

Run to another breakpoint

After establishing debug control, you can let the program run to another breakpoint:

<address> RB

In the transcript below, the UniLab was used to first establish debug control, and then to run to a second breakpoint. From the second breakpoint, we let the program run to a third.

RESET 14C RB resetting

```
AF=786A (sZ-a-pNc) BC=040E DE=0200 HL=1801 IX=FFFF IY=FDFD SP=1C00 PC=0140
014C 0B          DEC BC                      (next step) ok
```

15E RB

```
AF=0044 (sZ-a-Pnc) BC=0086 DE=FFFE HL=0000 IX=FFFF IY=FDFD SP=1BFE PC=0150
015E 39          ADD HL,SP                    (next step) ok
```

177 RB

```
AF=0044 (sZ-a-Pnc) BC=0086 DE=1BEE HL=0000 IX=FFFF IY=FDFD SP=1BF0 PC=0170
0177 C9          RET                          (next step) ok
```

Notice that you only enable **RESET** to establish debug control, not when running to subsequent breakpoints. If you enabled **RESET** each time, your program would start running again from the beginning, rather than continuing from the breakpoint where it is stopped.

Single-Stepping

After you have established debug control, you can step through your program, executing one opcode at a time.

To step through a series of instructions that do not have jumps, calls, or branches, use **N**. This command actually sets a breakpoint following the next opcode.

You use this command when:

the next instruction is not a jump,

or when it is a jump, call, or branch, but you don't want to see the program until it reaches the instruction that comes immediately after the current instruction.

Example

In the transcript below, we first establish debug control with **RB**, and single-step through a series of stack and register manipulations.

Notice that the Program Counter is always the same as the address of the "(next step)." You can also see the effects of the register manipulations in this code. The registers that are about to change are in **bold** text, and the ones that have just changed are underlined.

```
RESET 170 RB resetting
AF=0040 (sZ-a-pnc) BC=00DE DE=0002 HL=0F83 IX=FFFF IY=FDFF SP=1BEA PC=0170
0170 EB      EX DE,HL                      (next step) ok
N
AF=0040 (sZ-a-pnc) BC=00DE DE=0F83 HL=0002 IX=FFFF IY=FDFF SP=1BEA PC=0171
0171 E1      POP HL                        (next step) ok
N
AF=0040 (sZ-a-pnc) BC=00DE DE=0F83 HL=1BEE IX=FFFF IY=FDFF SP=1BEC PC=0172
0172 F9      LD SP,HL                      (next step) ok
N
AF=0040 (sZ-a-pnc) BC=00DE DE=0F83 HL=1BEE IX=FFFF IY=FDFF SP=1BEE PC=0173
0173 C1      POP BC                        (next step) ok
N
AF=0040 (sZ-a-pnc) BC=0086 DE=0F83 HL=1BEE IX=FFFF IY=FDFF SP=1BF0 PC=0174
0174 EB      EX DE,HL                      (next step) ok
N
AF=0040 (sZ-a-pnc) BC=0086 DE=1BEE HL=0F83 IX=FFFF IY=FDFF SP=1BF0 PC=0175
0175 7C      LD A,H                        (next step) ok
```

-- The Debugger --

Single stepping, following jumps

When the next instruction is a jump, branch, or call, and you do want to see the program execute it, you should use the command **SSTEP**-- if it is included in the debugger package for your processor.

Otherwise, you will have to manually set a breakpoint on the address that the code jumps to.

Do not use **N**, since that command does not follow jumps.

Example: Watch a Jump

Notice how **N** was used to step up to the jump, and then **SSTEP** was used to execute the jump itself.

RESET 134 RB resetting

```
AF=7842 (sZ-a-pNc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=0134
0134 BB CP E (next step) ok
```

```
N
AF=786A (sZ-a-pNc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=0135
0135 CA4201 JP Z,142 (next step) ok
```

The program executes the jump to 142.

SSTEP NMI

```
AF=786A (sZ-a-pNc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=0142
0142 210018 LD HL,1800 (next step) ok
```

Change the program counter and the run to breakpoint

To restart the program at a different address than the one you are stopped at, use

`<New PC> <address> GB`

This command takes two arguments. It puts the first value into the program counter, and sets a breakpoint at the second value. Then the UniLab releases the processor, so that it runs the program starting at the new code address pointed to by the program counter.

Note that this can have some unexpected results-- you are interfering with the program flow.

See the example on the next page.

-- The Debugger --

Example: Change PC, then run to a breakpoint

First, while stopped at a breakpoint
reset the PC and set a breakpoint
on the very next opcode address

RESET 160 RB resetting

```
AF=004C (sZ-a-Pnc) BC=0086 DE=1BFE HL=FFFE IX=FFFF IY=FDFD SP=1BFE PC=0160
0160 39          ADD HL,SP                      (next step)  ok
```

170 171 GB

```
AF=004C (sZ-a-Pnc) BC=0086 DE=FFFE HL=1BFE IX=FFFF IY=FDFD SP=1BFE PC=0171
0171 E1          POP HL                        (next step)  ok
```

Of course, you can set the breakpoint
and the new PC to the same address

RESET 160 RB resetting

```
AF=004C (sZ-a-Pnc) BC=0086 DE=1BFE HL=FFFE IX=FFFF IY=FDFD SP=1BFE PC=0160
0160 39          ADD HL,SP                      (next step)  ok
```

171 171 GB

```
AF=004C (sZ-a-Pnc) BC=0086 DE=1BFE HL=FFFE IX=FFFF IY=FDFD SP=1BFE PC=0171
0171 E1          POP HL                        (next step)  ok
```

Notice the difference in the HL register when
you actually run the program to the next
breakpoint, instead of changing the PC
This is an example of the unexpected results that
come from interfering with program flow

RESET 160 RB resetting

```
AF=004C (sZ-a-Pnc) BC=0086 DE=1BFE HL=FFFE IX=FFFF IY=FDFD SP=1BFE PC=0160
0160 39          ADD HL,SP                      (next step)  ok
```

171 RB

```
AF=0040 (sZ-a-pnc) BC=00DE DE=0000 HL=0002 IX=FFFF IY=FDFD SP=1BEA PC=0171
0171 E1          POP HL                        (next step)  ok
```

-- In Detail --

Set multiple breakpoints

Traditionally, multiple breakpoints were used when you did not know where the program was going to go next. You would try to block all exits by setting a breakpoint at every place the program could go.

The UniLab's ability to show you program flow makes multiple breakpoints obsolete. But, if you want to use them, here's how:

After establishing debug control, use

`<address> <breakpoint #> SMBP`

to set one of the eight numbered breakpoints.

You should set all but one of your breakpoints with **SMBP**, and then use

`<address> RB`

OR

`<New PC> <address> GB`

to set the last breakpoint and set the processor running again.

Establish debug control

You can also use **SMBP** before a

`RESET <addr> RB`

to establish debug control in the first place.

Clear breakpoints

If you want to clear out all multiple breakpoints, use **CLRMBP**. The command `<breakpoint #> RMBP` will clear one of the breakpoints.

-- The Debugger --

Example: Set multiple breakpoints

The transcript below shows an example of the use of the **SMBP** command while checking out the following code:

```
014D 79      LD A,C
014E B0      OR B
014F C24A01  JP NZ,14A
0152 C38000  JP 80
```

The problem to be solved: where does the program go after executing the code at address 014E. It might jump back to 014A, it might continue beyond that instruction and jump to 80. So you have to set two breakpoints:

RESET 14E RB resetting

```
AF=0D6A (sZ-a-pNc) BC=040D DE=0200 HL=1801 IX=FFFF IY=FDFD SP=1C00 PC=014E
014E B0      OR B                      (next step) ok
```

14A 1 SMBP

```
1 $014A 2 $---- 3 $---- 4 $---- 5 $---- 6 $---- 7 $---- 8 $----
```

80 RB

```
AF=0D08 (sz-a-pnc) BC=040D DE=0200 HL=1801 IX=FFFF IY=FDFD SP=1C00 PC=014E
014A 73      LD (HL),E                      (next step) ok
```

We find, not surprisingly, that the program jumps back to address 014A. But to find out about how the program flows, you will probably prefer to use the analyzer command `<addr> AS` as illustrated below. The analyzer trace shows you what happens each time the program reaches the code at 014E.

14E AS resetting

cy#	CONT	ADR	DATA		HDATA	MISC
-1	B7	014D	79	LD A,C	11111111	11111111
0	B7	014E	B0	OR B	11111111	11111111
1	B7	014F	C24A01	JP NZ,14A	11111111	11111111
4	B7	014A	73	LD (HL),E	11111111	11111111
5	D7	1801	00	write	11111111	11111111
6	B7	014B	23	INC HL	11111111	11111111
7	B7	014C	0B	DEC BC	11111111	11111111
8	B7	014D	79	LD A,C	11111111	11111111
9	B7	014E	B0	OR B	11111111	11111111
A	B7	014F	C24A01	JP NZ,14A	11111111	11111111
D	B7	014A	73	LD (HL),E	11111111	11111111
E	D7	1802	00	write	11111111	11111111

-- In Detail --

Examine and alter internal registers

The breakpoint display shows your internal registers. Of course, this display varies from processor to processor.

You can display all registers again with **R**.

And you can alter them with commands that follow this pattern:

`<value> =Name_of_Register`

The commands for altering registers are processor specific. For example, the Z80 package includes:

=AF **=BC** **=DE** **=HL** **=IX** **=IY**

Check the glossary section in the Disassembler/Debugger writeup for your processor, or press **CTRL-F3**.

Example: Alter the flags register

Notice how, in the example below, we change the flow of the program by altering the "Zero" Flag.

R shows you the register display again-- very handy for verification after you've changed a register.

After we alter the flag, we single step, and see that the program takes the jump, because of the change to the flag.

RESET 12F RB resetting

AF=02A8 (Sz-a-pnc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=012F
012F BA CP D (next step) ok

N

AF=0242 (sZ-a-pNc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=0130
0130 C23801 JP NZ,138 (next step) ok

0202 =AF ok

R

AF=0202 (sz-a-pNc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=0130
0130 C23801 JP NZ,138 (next step) ok

SSTEP NMI

AF=0202 (sz-a-pNc) BC=002C DE=0278 HL=0278 IX=FFFF IY=FDFE SP=1C00 PC=0138
0138 7E LD A,(HL) (next step) ok

-- The Debugger --

Examine and alter RAM

While stopped at a breakpoint, you can use all the memory access commands to examine and alter either RAM or ROM.

If you have not established debug control, most of these commands will only work on emulation ROM-- and access to emulation ROM will cause your program to crash.

For details on the memory commands, see section 3: Examining and Altering Memory.

6.6 Trigger-Style Breakpoints

On some processors you can establish debug control using trigger-style commands-- and thus establish a breakpoint when certain conditions appear on the bus. This feature is only supported on microprocessors that support **NMI** (see Appendix H).

You use **RI** and **SI** to invoke the debugger on your program a cycle or two after the bus conditions appear.

You can use **RI** and **SI** either to establish debug control in the first place, or to run to the next breakpoint after you have already established control.

RI declares that the trigger spec which follows will be used to establish debug control. **SI** then sets the analyzer going. Like this:

RI <trigger spec> **SI**

Qualifiers should not be used in this trigger spec. If you do use them, the result will be that the qualifier and trigger must occur one immediately after another.

Example: Trigger style breakpoints

Here, as usual in these examples, we are again looking at the Z80 test program. The example shows the setting of two breakpoints using analyzer style commands. Note that the breakpoint occurs one cycle after the trigger event occurs. The disassembly of the code in which we are setting breakpoints appears below, for convenience.

```
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
(... jump back to address 3 ... )
```

RI 10 ADR 3C DATA SI resetting

```
AF=1300 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=E5C1 IY=E5C1 SP=1900 PC=0011
0011 3C INC A (next step) ok
```

RI 18FF ADR AFTER 0E ADR SI

```
AF=1228 (sz-a-pnc) BC=3456 DE=789A HL=BCDE IX=E5C1 IY=E5C1 SP=18FE PC=000F
000F C1 POP BC (next step) ok
```

-- The Debugger --

6.7 Exit from Debugger

There are four ways to exit from the debugger:

- 1) **RZ** immediately releases the program from debug control, so that it starts running again,
- 2) **<addr> G** releases the processor from debug control after changing the Program Counter,
- 3) **<addr> GW** changes the Program Counter, and then waits to release the processor until you restart the analyzer,
- 4) or you can define a trigger spec, and start up the analyzer.

If you exit from the debugger by starting up the analyzer be sure to remember that the debugger has disabled reset. If you do want the program to start over from the beginning, you have to enable automatic resetting with **RESET** or the mode panel (function key 8).

You will definitely want to use **NORMx** to clear out the special trigger specifications that the debugger commands use.

A simple alternative is to just use **STARTUP**, which clears out the previous trigger and starts the program over from the beginning.

Exit after a target system crash

If you crash the target system while you are in the debugger, you will need to start the target program over from the beginning.

Examples: Exiting from the debugger

If you want the processor to start executing the program again, without restarting the program, use:

<New PC> **GW** <analyzer trigger spec>

to change the program counter, and then wait for the analyzer to start up. The analyzer trigger specification can appear on the same line or on a separate line.

For example:

```
8 GW 03 AS
cy#  ADR DATA
-5  0027 3C      INC A
-4  0028 3C      INC A
-3  0029 C30300  JP 3
0   0003 3E12    LD A,12
2   0005 015634  LD BC,3456
5   0008 119A78  LD DE,789A
8   000B 21DEBC  LD HL,BCDE
B   000E C5      PUSH BC
```

If you want to release the processor and set it running, without setting any analyzer trigger spec, use:

<New PC> **G**

-- The Debugger --

6.8 Disabling the Debugger: How and Why

Why

The Orion debuggers reserve between one and six bytes of your ROM, and overlay code into an additional 30 to 70 bytes.

You can put your own code in the overlay area, but never in the reserved area (**CTRL-F3** tells you where the reserved area is on your processor). If you disable the debugger, then you can make use of these small areas of memory.

The **NMI** command makes use of the non-maskable interrupt feature of your processor. If your target board hardware makes use of that feature, you will want to disable the UniLab NMI Vector.

If you want to run a program from a ROM chip on your target board, you must first clear out emulation memory enables with **EMCLR** and then disable the debugger as well-- you will, of course, still be able to use the analyzer and disassembler.

How: Disable the debugger

If you need to use that reserved area of one to four bytes, you can turn off all debugger commands with **RSP'** or with the mode panel (**F8**).

Mode Panel:

```
3. LOG modes
LOG TO PRINT inactive
LOG TO FILE  off
PRINTER     off
NMI VECTOR  active
SWI VECTOR  active
```

How: Disable use of the NMI feature

If you need to disable the Orion software's use of the hardware interrupt feature of your processor, use **NMIVector'** or the mode panel (**F8**). This will disable **NMI**, **SSTEP**, and **RI** and **SI**.

```
NMI VECTOR  active
```

-- In Detail --

6-124

7. Burning PROMs

Introduction

You can do all your EPROM programming from the menu system. The menus allow you to program any EPROM with just a few key strokes. The menus include reminders about which "personality module" is required.

When your program is working perfectly under emulation, you can copy it into virtually any single-supply EPROM or EEPROM directly from the emulator memory. To program a 2716, for example, from target locations 800 to FFF, you just put an erased 2716 in the socket and choose the appropriate menu option. You will be prompted for the starting and ending addresses, then the ROM will be programmed.

Erase check, programming, and verification will immediately begin, and the LED to the right of the socket will light. The light goes out when the PROM has been programmed (usually just a few seconds).

Contents

7.1	Feature Summary	6-126
7.2	Personality Modules	6-127
7.3	Plugging in PROMs	6-129
7.4	Calculate Checksums	6-131
7.5	Verify Your PROM	6-132
7.6	PROMs for 16-bit Processors	6-132
7.7	Programming in Standalone Mode	6-133
7.8	Sample Macro for EPROM Production	6-134

-- Burning PROMs --

7.1 Feature Summary

Though there are commands for burning programs into each type of EPROM we support, we recommend that you use the EPROM burning Menu whenever possible. These menus are always just a few keystrokes away, and include reminders about which PM (personality module) you need for each EPROM.

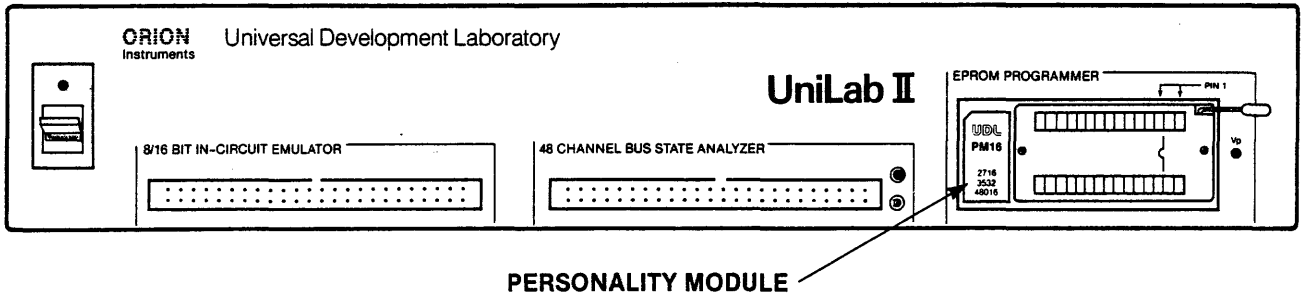
The only time you would really need the command rather than the menu item, is when you want to burn a EPROM from within a macro. See also Appendix G for information on EPROMs.

All the EPROM programming commands are covered by the menus:

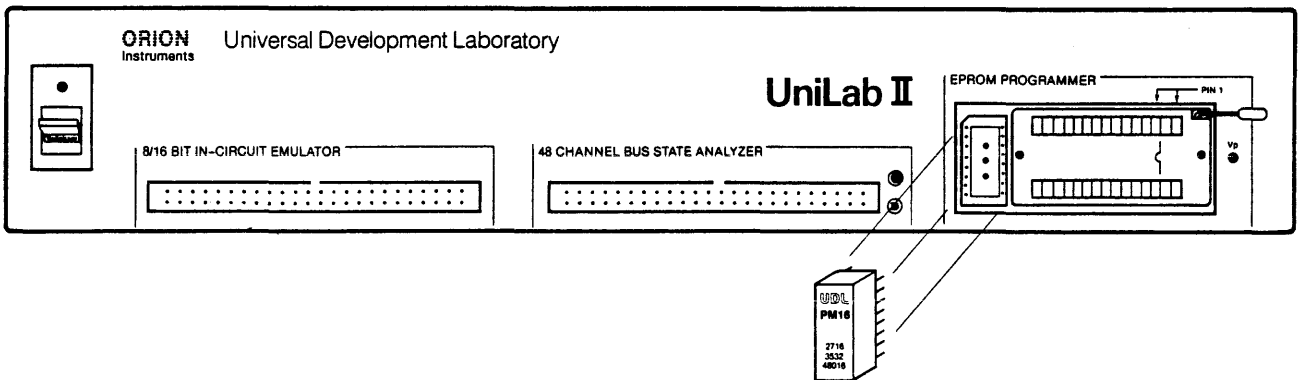
Menu:	Command:
PROM PROGRAMMING MENU #1	
F1 PROGRAM A 2716 (use PM16 personality module)	P2716
F2 PROGRAM A 2532 (use PM16 personality module)	P2532
F3 PROGRAM A 2732A (use PM32 personality module)	P2732A
F4 PROGRAM A 2764A (use PM64 personality module)	P2764
F5 PROGRAM A 27128A (use PM56 for A version)	P2764
F6 PROGRAM A 27256A (use PM56 personality module)	P27256
F7 PROGRAM A 27512 (use PM512 personality module)	P27512
F9 Next page of Prom Programming Menu	
F10 RETURN TO MAIN MENU	
PROM PROGRAMMING MENU #2	
F1 PROGRAM A 27C16 (use PM16 personality module)	PD2716
F2 PROGRAM A 48016 (use PM16 personality module)	P48016
F3 PROGRAM A 27C32 (use PM16 personality module)	P27C32
F4 PROGRAM A 2764 (use PM64 personality module)	PD2764
F5 PROGRAM A 27128 (use PM64 personality module)	PD2764
F6 PROGRAM A 27256 (use PM56 personality module)	PD27256
F9 RETURN TO PROM READER MENU	
F10 RETURN TO MAIN MENU	

7.2 Personality Modules

Whether reading or burning an EPROM or EEPROM, you have to have the correct Personality Module in the 16 pin socket just to the left of the EPROM PROGRAMMER socket.



The personality module is necessary because control signals vary from EPROM to EPROM. The Orion UniLab uses the personality module to alter the voltage and pin location of control signals. This makes it possible for one unit to program all the most popular EPROMs, with a simple change in PM.



-- Burning PROMs --

See Appendix G for full information on EPROMs and Personality Modules.

The UniLab is shipped with:

For 21 volt EPROMs:

PM16 for programming 2716, 27C16 and 2532 EPROMs.

PM32 for programming 2732 and 2732A EPROMs.

PM64 for programming 2764, 27C64 and 27128 EPROMs.

For 12.5 volt EPROMs:

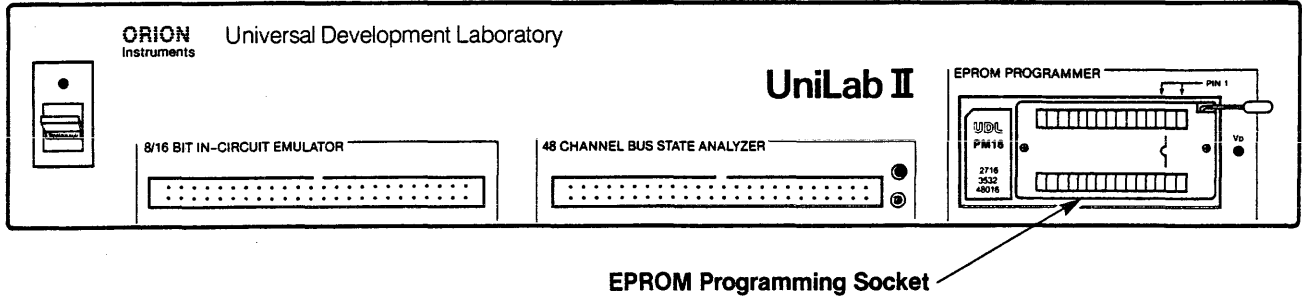
PM56 for programming 12.5 volt programmed EPROMs
such as 2764A, 27C64, 27128A, 27C128, and
27256.

You can also purchase:

PM512 for programming 12.5 volt 27512 EPROMs.

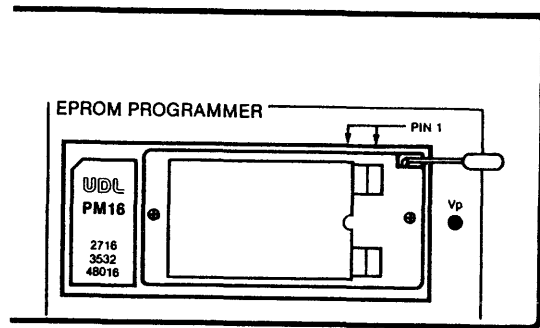
PM56-21 for programming 21 volt 27256 EPROMs.

7.3 Plugging in PROMs

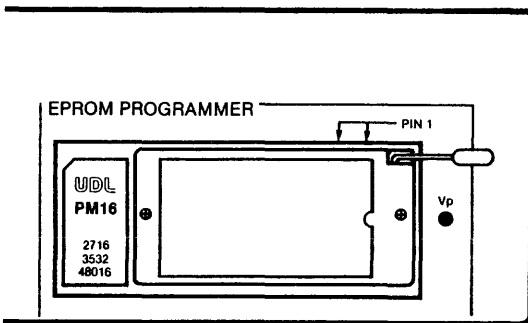


Be sure to plug PROMs into the socket with the notch to the right.

24-pin EPROMs should be inserted into the socket shifted as far to the left as possible:



24 Pin EPROM in Programming Socket



28 Pin EPROM in Programming Socket

28-pin EPROMs will fill up the whole socket.

-- Burning PROMs --

Never turn power on and off with a PROM in the socket-- this could erase location 0. The same warning applies to changing the personality module with the PROM in the socket.

In general, don't leave the PROM in the socket any longer than necessary to read or program it.

The UniLab's smart programming algorithm guarantees a 4:1 margin on stored charge while taking a minimum amount of programming time. An erase check is done before programming starts and all locations are verified during programming.

-- In Detail --

6-130

7.4 Calculate Checksums

If you want to put checksums in your EPROMs, a **CKSUM** command is provided to compute them for you. Enter:

```
adr toadr CKSUM
```

to calculate the sum.

Then use the **MM!** command to put the checksum in the desired location before burning the PROM.

Be sure to have a known value (such as 0 or 1) in the location you plan to put the checksum in (usually the top or bottom of memory) before executing **CKSUM**.

-- Burning PROMs --

7.5 Verify Your PROM

PROMs are verified during the programming process.

However, if you want to separately verify a PROM, you can read it into another area of memory and use the **MCOMP** command to compare to the original data.

7.6 Program PROMs for 16-bit Processors

When the 16-bit mode has been selected (by entering **16BIT**) the prom programmer will automatically select odd or even bytes to go with the first address you have selected.

1 TO FFF RPROM or 1 TO FFF P2716

will thus read or write odd bytes only, while

0 TO FFF P2716

will write even bytes only.

The same rule applies when programming or reading using the PROM menus (**F9** under the menu system).

7.7 Standalone PROM Programming

Programming a large EPROM can take a significant amount of time. If you want to use your host computer for some other task while the UniLab burns the EPROM, you can precede an EPROM programming command with the word **STANDALONE**.

You can also type in the command **STANDALONE**, and then use one of the EPROM programming menus to start the programming operation going.

This will cause the UniLab to do its work without needing to be in contact with the host computer. When the UniLab is finished programming, the red light next to the EPROM PROGRAMMING socket will go out. You can then use the command **PROMMSG** to get the message that tells you the completion status of the programming operation.

-- Burning PROMS --

7.8 Sample Macro for Production of EPROMs

It can be tiring, when programming many identical EPROMs, to keep typing in the same series of instructions to the menu.

Fortunately, you can make a simple macro that will take care of the EPROM programming for you. For example, if you are burning a 2764 with the code that starts at 0 and goes to 1300:

```
      : BURN 0 1300 P2764 ;
```

After that, all you have to do is type in **BURN** to program the ROM.

Appendix F tells you more about macros.

If you choose to not use the menus, check Appendix G to find out more about what commands and **PMs** you'll need for each EPROM.

8. Generating Stimuli

Introduction

Often in system checkout it is useful to build a switch panel to allow system inputs to be changed easily. The UniLab stimulus outputs make this unnecessary by providing eight latched output bits that are controlled from your keyboard.

Contents

8.1	Feature Summary	6-136
8.2	How to Do It	6-137

-- Generate Inputs --

8.1 Feature Summary

<u>Feature</u>	<u>Menu</u>	<u>Command</u>
Generate a high signal on one wire	Yes	<wire #> SET
Generate a low signal on one wire	Yes	<wire #> RESET
Define bit pattern of all 8 wires	Yes	<hex byte> STIMULUS

Command:

Menu:

STIMULUS MENU

<wire #> SET	F1	SET A STIMULUS BIT
<wire #> RESET	F2	RESET A STIMULUS BIT
<hex byte> STIMULUS	F3	DEFINE ALL 8 STIMULUS BITS
	F10	RETURN TO MAIN MENU

-- In Detail --

8.2 How to Do It

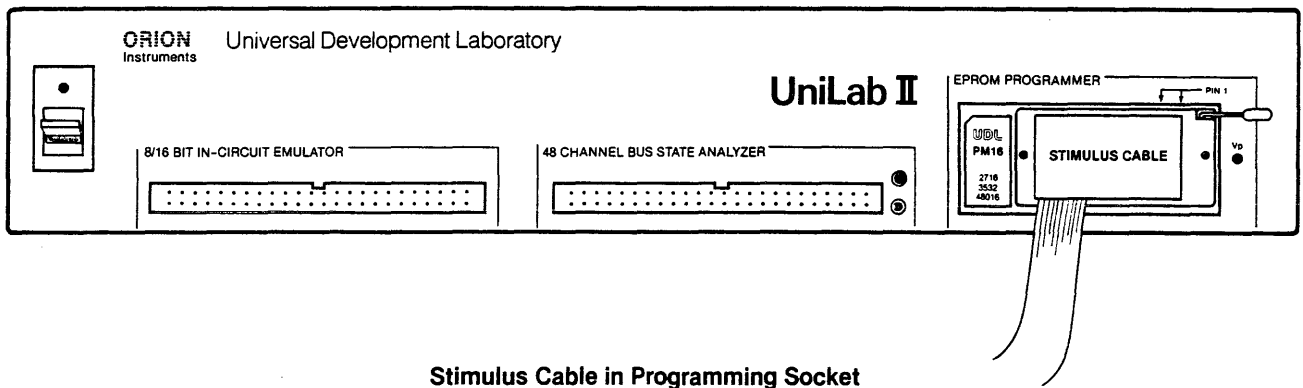
The eight stimulus generator signals, asserted at the EPROM programmer socket, can be individually set or reset from the keyboard, set and reset as a group, or programmed to produce a repeating pattern.

A ninth output (ST-) gives a 4-microsecond low pulse whenever any of the wires are changed.

Connecting Stimulus Cable

The stimulus cable actually plugs into the EPROM programming socket and brings the signals out to .025" receptacles, like the ones used on the analyzer cable. These output signals of the UniLab can be plugged into wire wrap pins or DIP-CLIPS.

You then connect the signals to the inputs of your target system, and use the stimulus generator as a "control panel" during system checkout.



-- Generate Inputs --

Specify all 8 bits

You can specify the 8 bits of the stimulus signal with <value> **STIMULUS**. For example, to make bits 7 and 2 high, while all other bits are low, type in:

84 STIMULUS

The number 84 hex is, of course, 1000 0100 binary.
bit # 7654 3210

Change one bit at a time

You can also set or reset the bits individually with <bit #> **SET** and <bit #> **RES**. For example,

1 SET

will set bit # 1 high.

You can also use these two commands to "pulse" target system inputs. For example, to pulse an "active high" signal with stimulus wire 3:

3 SET 3 RES

Stimulus generator and macros

You can assign a convenient name to any stimulus configuration by simply preceding the name with a colon, and ending the definition with a semicolon. For example,

```
: START1 0 SET 1 RES 3 RES ;
```

will define a word **START1**, which causes the UniLab to perform three operations: set stimulus #0 high, then set stimulus #1 low, last set stimulus #3 low.

You could also define **START** this way:

```
: START2 01 STIMULUS ;
```

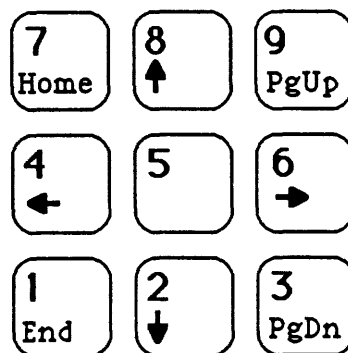
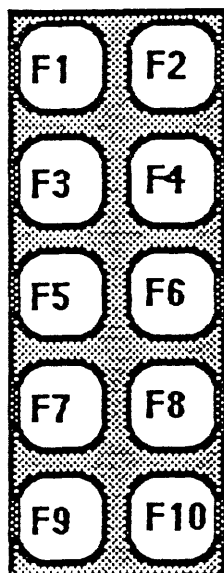
though this will have a slightly different effect than the first definition-- it sets bit 0 high, bits 1 through 7 low, and it does this all at the same time.

The stimulus generator commands are very useful in test programs. You can use the generator to change the inputs to the target system in sequence, and then compare the resulting traces. See Appendix F for more info on test macros.

9. Special Keys

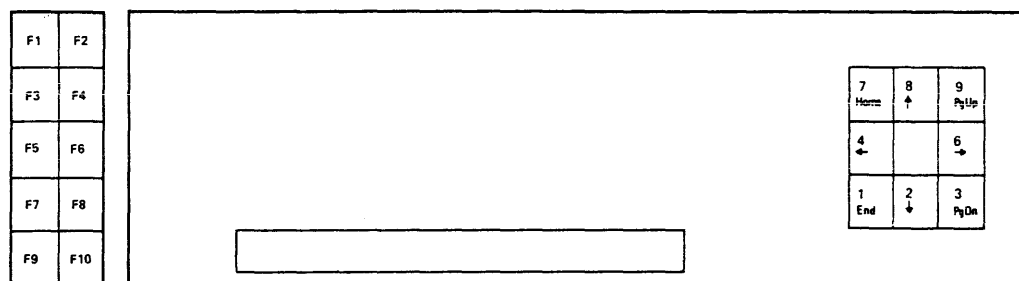
Contents

9.1	Feature Summary	6-141
9.2	Function Keys	6-142
9.3	Cursor Keys	6-144



9.1 Feature Summary

When you are in command mode, some UniLab features can be accessed through the function keys and cursor keys.

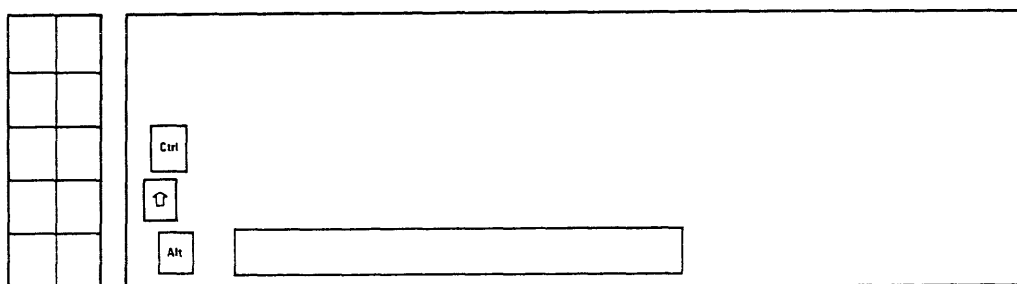


Function Keys and Cursor Keys

In the UniLab software, the cursor keys are always used by themselves. The function keys can be used by themselves, or while you are holding down any one of:

the ALT key,
the SHIFT key,
the CTRL key.

This means that you really have access to forty function keys.



ALT, SHIFT and Control Keys

Often used commands and help screens have been pre-assigned to many of the 40 function keys. You can change the function associated with any of the function keys.

You use the cursor keys to move through textfiles, trace displays, and the line history.

-- Special Keys --

9.2 Function Keys

All the commands that you call up with the function keys can also be executed by typing in the command-- but it is usually more convenient to use the function key. The only time you would need to use the command is within a macro definition.

See the commands assigned to the function keys

Function key one tells you the current assignments of the function keys.

Hit function key one (**F1**) to find out what commands have been assigned to the "bare" function keys.

Hit **F1** while holding down **ALT** to see the current assignments of the ALTEred function keys.

Hit **F1** while holding down **SHIFT** to see the current assignments of the SHIFTEd function keys.

Hit **F1** while holding down **CTRL** to get a display of the help screens assigned to the CONTROLLEd function keys.

Change the commands assigned to the function keys

You can easily change the command that gets executed by any function key. You use one of four commands to do this. All of them take the same parameters:

```
<# of key> FKEY <command>  
<# of key> ALT-FKEY <command>  
<# of key> SHIFT-FKEY <command>  
<# of key> CTRL-FKEY <command>
```

Any command that does not take parameters can be reassigned to a function key-- including any macros that you write.

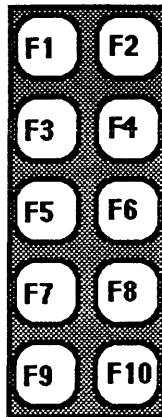
Help for using on-line displays

Help for Debuggers

Help for Emulation memory functions

Help for loading/saving programs

Help for displaying/altering memory




Help for using windows

Help for simple analyzer triggers

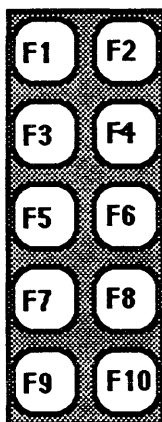
More help for analyzer triggers

Help for mode panel switches

Help for trace display

Function Key assignments when  key held down


List Function Key assignments for Shift



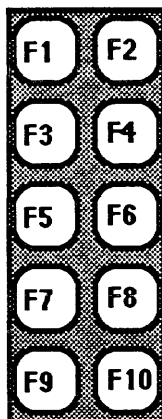
Memo - Bring up system editor for use as custom memo pad


Ascii display - Shows ascii values for keys.

Set new window split size

Function Key assignments when  key held down

List Function Key assignments for Alt



Function Key assignments when  key held down

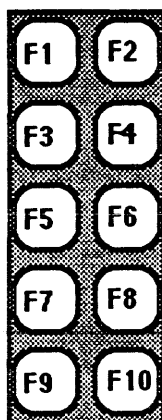
HELP with general instructions for using glossary. Also Function Key assignments.

Next Step - Execute next instruction. Will not follow jumps or branches.

Restore window split to Default sizes.

TSTAT - Display current trigger spec.

STARTUP - Issue reset pulse to target and trace first cycles of target operation.



SPLIT mode - Enter/Exit split screen mode.

NMI - Issue NMI pulse to target to get breakpoint.

Single Step - Execute next instruction. Will follow jumps and branches. May be same as NMI.

MODE - Bring up pop-up mode panels for changing display or system modes.

MENU - Enter/Exit menu mode.

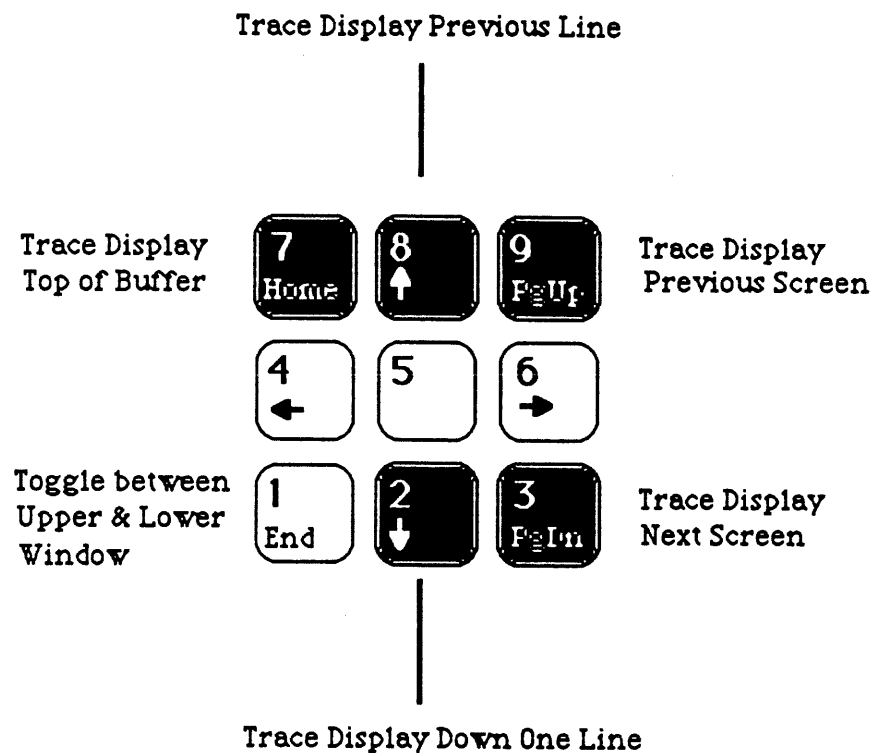
Function Key assignments when no other key held down

-- Special Keys --

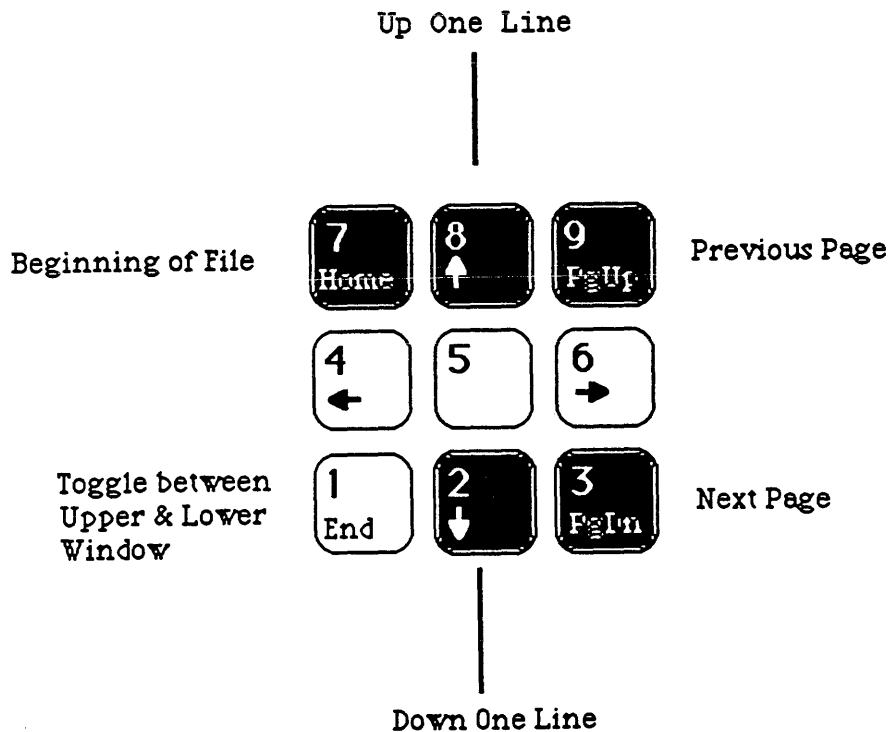
9.3 Cursor keys

You use the cursor keys on the numeric key pad to move through various displays. The functions of the keys changes as you change the task you are working on.

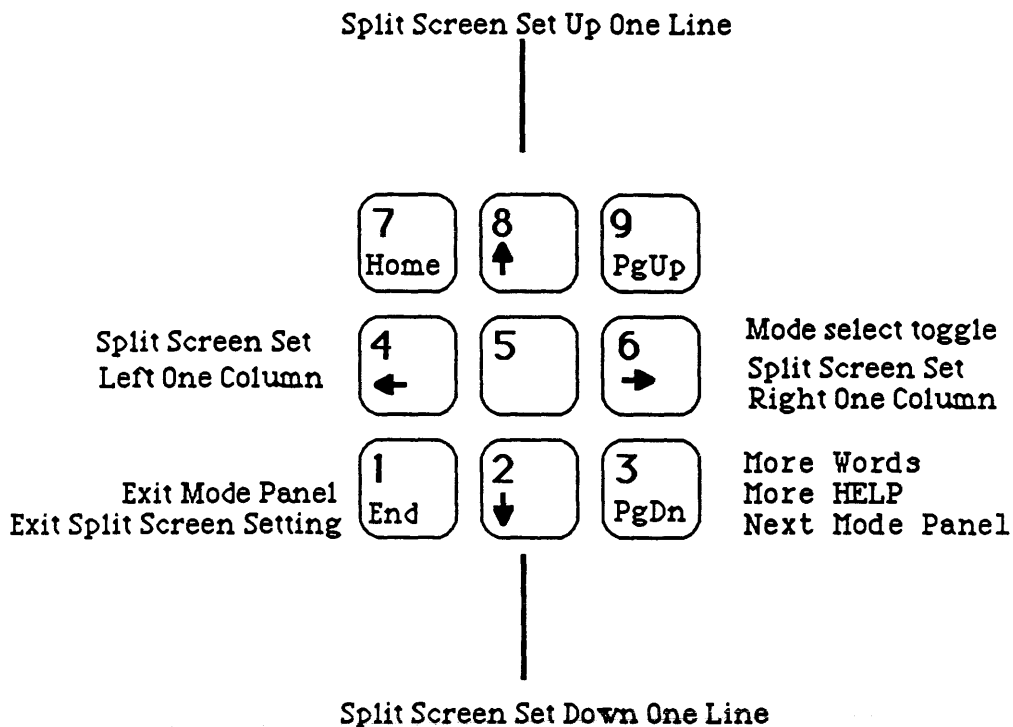
Cursor Key Assignments for Viewing Trace Buffer Display



Cursor Key Assignments for Viewing Text Files



Other Cursor Key Uses



10. Mode Panels

Introduction

The mode panels, available at the press of a button (F8), let you toggle features on and off without any need to remember commands.

The Mode Panels are very simple to use, including on-line help. Each mode panel and its help screens are reproduced here as a courtesy, as well as in Chapter 5.

Contents

10.1	Feature Summary	6-147
10.2	Analyzer Modes	6-148
10.3	Display Modes	6-149
10.4	Log Modes	6-150

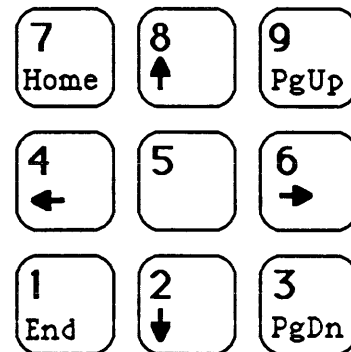
10.1 Feature Summary

Press **F2** to get into the Mode Panels. Once you have a mode panel on the screen, you can run through all of them, by hitting **F2** repeatedly.

Use the **UpArrow** and **DownArrow** keys to move around within each mode panel, from option to option.

The **RightArrow** toggles the current option. Press **F1** to get help screen for the current option.

Press **END** to exit from the Mode Panels.



Equivalent commands:

DASM DASM'
SYMB SYMB'
RESET RESET'

SHOWM SHOWM'
SHOWC SHOWC'
<value> =MBASE
PAGINATE PAGINATE'
HDG HDG'

LOG LOG'
TOFILE
PRINT PRINT'
NMIVEC NMIVEC'
RSP RSP'

Mode Panel:

1. ANALYZER modes
DISASSEMBLER
SYMBOLS
RESET

2. DISPLAY modes
MISC COLUMN
CONT COLUMN
MISC # BASE
PAGINATE
FIXED HEADER

3. LOG modes
LOG TO PRINT
LOG TO FILE
PRINTER
NMI VECTOR
SWI VECTOR

-- Mode Panels --

10.2 Analyzer Modes

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 **SYMPFILE** . 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'**

10.3 Display Modes

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'**

-- Mode Panels --

10.4 Log Modes

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'**

11. Windows

Introduction

A complete discussion of the split screen capabilities of the UniLab software appears in the third section of the **Getting Started** chapter. The material here includes only the summary of split screen features.

Feature Summary

<u>Feature</u>	<u>Function key</u>	<u>Command</u>
Toggle the horizontal split on and off	F2	SPLIT
Disassemble into the left-hand window	No	<addr> DN
Display text file in upper window	No	TEXTFILE
Change the size of the split screen	SHIFT-F8	WSIZE
Return split screen size to the latest setting, after a help screen has altered it	F5	DEF
Switch between top and bottom screens	End	TOP/BOT

Screen History display

Look at one page of screen history	PgUp	No
Look at one line of screen history	UpArrow	No

12. Histograms

Graphical Performance Measurement Option (Advance Information)

Introduction

This optional UniLab feature produces a bar graph that helps you to assess the performance of your target software.

The histogram points out to you where your program is spending its time, so that you can decide where to focus your time and energy.

You can produce two types of histograms, as described in the following pages.

Before you can use histograms, you must enable them with the command **DOHIST**. This word saves the system and then causes you to exit to DOS. When you invoke the newly saved system, you will be able to use histograms.

Contents

12.1	Feature Summary	6-153
12.2	When to Use Histograms Address HIST ograms Time HIST ograms	6-154
12.3	Making Histograms	6-157

12.1 Feature Summary

You only need two commands to gain access to the power of the histogram. After you have typed one of the commands, you are interacting with the fully menu-driven histogram generator.

You press **F10** to exit from the histogram.

In normal operation, you can exit from the histogram feature, and later in the same UniLab session re-enter the histogram with all your data preserved. Two additional commands allow you to save the information about a histogram to a file, and later reload it.

<u>Feature</u>	<u>Command</u>
Produce and display histogram of <u>relative</u> time the program spends in each of up to 15 user-specified address ranges. Address HISTogram .	AHIST
Produce and display histogram of how often the elapsed time between two addresses falls into each of up to 15 user-specified time ranges. Time HISTogram .	THIST
Save to a file the data from the most recent histogram	HSAVE <file>
Retrieve from a file the data from a previously saved histogram	HLOAD <file>

-- Histograms --

12.2 When to Use Histograms

The two types of histograms share the same screen layout. Both show bins running down the left-hand side of the screen, column labels at the top of the screen and the function key menu at the bottom.

This similarity masks a fundamental difference: the **Address HISTogram (AHIST)** shows you a graph based on address bins, while the **Time HISTogram (THIST)** shows you a graph based on time bins.

What the address histogram does

The address histogram helps you find bugs, and discover the time hogs in your code.

You invoke this feature with the command **AHIST**, and then specify up to 15 address bins by filling in the starting and ending addresses.

When you start the histogram with **F1**, the UniLab will count the number of times your program executes an address that falls into each bin. This information is automatically sent to your host computer, which displays a histogram of the results.

How to zero in with the address histogram

The very first thing you will want to do is set up address bins that cover the entire range of addressable memory-- that way you will be able to check that your program is executing code only where you expect it to be. Set the histogram going with function key 1 (**F1**). Counts appearing outside of the expected range indicate possible bugs.

After you've verified that the program is well-behaved, you will probably want to divide the active range of memory based on the functional units of your code. For example, the addresses 0 to 200 contain the initialization code, 210 to 400 contains the main loop, 1500 to 1600 is an error handling routine, etc.

You will then be able to start the histogram and see where your program is spending most of its time.

What the time histogram does

The time histogram helps you speed up routines and determine the efficiency of program flow.

You invoke this feature with the command **THIST**. This **Time HISTogram** shows you how frequently the elapsed time between two addresses falls into each of the time bins that you specify.

You will want to first set up the starting and ending addresses of the routine or loop that you are interested in timing (use **F7**). Then you can fill in the limits of the time bins.

When you start the histogram, the UniLab will keep a count of the elapsed time between the single starting address and the ending address. This information is sent up to your host computer, which displays a histogram showing how frequently the elapsed time falls into each bin.

Note that the time histogram functions as a stop watch-- it starts a time count when the starting address appears on the bus, and stops the time count when the ending address shows up. This means that you must choose your address boundaries carefully-- if the ending address is not executed, then the stop watch will never stop.

How to zero in with the time histogram

The very first thing you will want to do is set the time units (**F8**) to the highest value-- 10 milliseconds. Then set up time bins that cover the range from 0 to 60000. Time values are entered in decimal, not in hexadecimal. By starting out with this "big picture" view, you will be certain to catch all the information, and then narrow down with confidence.

Start up the histogram and you will be able to see in what range of time periods the routine falls. By adjusting the time units (in powers of ten, down to 10 microseconds) and the time bins, you will be able to get whatever resolution you desire.

Measure, then adjust

Once you have established how long your time-critical routine takes, you can make modifications to your code or outside conditions, and then measure the routine again. **THIST** will show you how much time has been gained (or lost) by the alterations.

You can also use **THIST** to determine the efficiency of program flow. For example, the time histogram might reveal to

-- Histograms --

you that a routine takes 40 microseconds 80% of the time, but requires 10000 microseconds 20% of the time.

You will probably want to write a special routine to correct this lopsided time performance. **THIST** will then help you evaluate your efforts.

-- In Detail --

6-156

12.3 Making Histograms

When you call up the histogram, the cursor and function keys are reassigned. When you exit from the histogram by hitting **F10** you return to the state you were in when you called the histogram.

Filling in the bin limits

Before you can generate a histogram, you have to fill in the upper and lower bounds of at least one "bin." When you enter the histogram, your cursor will be positioned for entering the lower bound of the first bin. Just type in the number, and then hit return, the right arrow or the TAB key to move to the next entry field. See the shortcut below (**F3**).

Starting display

Once you've filled in as many bins as you want, press **F1** to start the display. You cannot start the display of a time histogram (**THIST**) until after you've filled in the address range (**F7**).

When you do start the display, you will see the data three ways: absolute counts, percentage, and a bar graph of the percentage.

The program will inform you if you've made an error, such as giving overlapping ranges, or leaving off a limit.

Press any key to stop the display.

Name your bins-- Value/Name toggle

You can name your bins by pressing **F2** and then filling in the name fields that replace the limit field.

Press **F2** again to return to the value fields.

-- Histograms --

Subdivide-- Shortcut to entering limits

You can divide a bin into a number of bins of equal size.

Place the cursor on the bin you wish to divide and press **F3**. Then use the **DownArrow** key to move down, and press **F3** again. The range contained in the original bin will be divided among the number of bins that you spanned.

The original bin must already contain a valid range.

Delete entry

Press **F4** to delete the bin that the cursor is on.

Clear accumulated values

Press **F5** to clear any counts accumulated so far, before restarting the histogram with **F1**.

Clear all

Press **F6** to clear out all counts and the bin boundaries.

Address limits (THIST) or title (AHIST)

Within **THIST** you must press **F7** to enter the upper and lower address for the time histogram. You will not be able to start accumulating and displaying data until you enter this information. Hit return after you've entered the values.

Within **AHIST** you can enter a title for your histogram by pressing **F7** and entering the name. Hit return after you've entered the title.

Time unit (THIST) or Trigger Spec (AHIST)

Within **THIST** you can display and alter the unit of time in which your input is interpreted and the output displayed by pressing **F8**. The default is 10 (decimal) microseconds, mainly because it is fairly easy to think in units of 10.

The resolution of the histogram generator is actually 20 microseconds-- don't let the time unit give you a false sense of accuracy. No matter what value you choose as the time unit, the

-- Histograms --

histogram can be off by as much as 10 microseconds.

Within **AHIST** press **F8** to change the trigger spec, or enter any other command.

16/20- bit toggle (AHIST only)

Normally you enter 16 bit addresses as the boundaries of **AHIST** bins. That means the largest address you can enter is **FFFF**. Press **F9** to toggle over to 20-bit addresses-- values up to **FFFFF**.

Exit

Press **F10** to exit.

Chapter Seven: UniLab Command Reference

Introduction

This chapter contains the reference material for the UniLab command language, a rich, flexible language that you use to work on your microprocessor control board.

The first, brief, section divides the words into one of five categories:

1. Beginner words-- the minimal vocabulary you need to talk to the software.
2. Common words-- commands that you will quickly learn and use often.
3. Advanced words-- commands that you will find useful, but can function fine without knowing.
4. Special key and mode panel words-- commands that perform the same function as one of the mode panel switches or one of the function or cursor keys.
5. Rarely used words-- commands that you will probably be able to live without.

The second of the two sections in this chapter contains a page or so of reference material for each important command. The format is explained, and then the entries follow.

You can also get the entry for any command on-line by typing

HELP <command>.

An alphabetical listing of all words appears in Appendix A.

Warning

Each disassembler/debugger includes commands that are specific to that package. These words are not documented in this chapter. To learn more about target-specific words, look at the separate writeup for your software package.

Contents

The Categories	7-2
The Commands	7-9

The Categories

The Orion software provides you with access to commands that let you:

set triggers on any input or combination of
inputs,
alter the display and logging features,
set breakpoints and alter registers.

You will be able to do most of your work with just a few commands: the beginner and common words, with occasional forays into the advanced words.

The panel words help you find out more about the features that you toggle with the Mode Panel.

The rarely used words are provided as a service to those of you who wish to delve deeper into the capabilities of the instrument.

-- The Categories --

1. Beginner words-- the minimal vocabulary you need to talk to the software.

Get On-Line Help

HELP	MENU	MESSAGE
PINOUT	WORDS	

Load the Simple Target Program

LTARG

Trigger on Address

AS

Start Analyzer

S	STARTUP
---	---------

Exit the Program

BYE

Memory Enable

EMENABLE

Memory Reading

DM	DN	MDUMP
----	----	-------

Predefined Triggers

ADR?	CYCLES?	EVENTS?
NOW?	SAMP	

Status Enquiry

ESTAT	TSTAT
-------	-------

Debugger Words

N	RB
---	----

-- The Categories --

2. Common words-- commands you will quickly learn and use often.

Set Trigger

ALSO	ADR	DATA
NORMB	NORMM	NORMT
NOT	ONLY	TO

Set Trigger-- Not supported on all processors

FETCH	READ
-------	------

Start Analyzer

S+

Call DOS

DOS

Trace Reading Commands

TCOMP	TD	TMASK
TN	TSAVE	TSHOW

Debugger Commands

G	N	NMI
RB	SSTEP	

Stimulus Generator Commands

RES	SET	STIMULUS
-----	-----	----------

Memory Reading

M?	MM?	MCOMP
----	-----	-------

Memory Writing

ASM and ASM-FILE (processor specific on-line assembler)

MFILL	M	M!
MFILL	MM	MM!
MMOVE	ORG	

Define Symbols

IS

Save Information

BINSAVE	TOFILE	SAVE-SYS
SYMSAVE	TSAVE	

Load Program From File

BINLOAD	HEXLOAD
---------	---------

Examine Text File

TEXTFILE

Initialize Instrument

INIT

-- The Categories --

3. Advanced words-- commands that you will sometimes find useful, but can live without.

On-Line Displays

CATALOG ASC

Graphical Performance Measurement (optional feature)

AHIST THIST

Trigger Commands

DCYCLES HADR HDATA
LADR MASK MISC

Start the Analyzer Repeatedly

SR

Filter the Trace

1AFTER 2AFTER 3AFTER

Define a Pre-Qualifier

AFTER PCYCLES PEVENTS

Debugger Commands

GB GW RZ

Multiple Breakpoints

CLRMBP DMBP RMBP
SMBP

Symbol Table manipulation

CLRSYM SYMFILE SYMFILE+
SYMLOAD SYMSAVE SYMTYPE

Disable Emulation Memory

EMCLR

Assign a Function to a Function Key

ALT-FKEY CTRL-FKEY FKEY
SHIFT-FKEY

Calculate a CheckSum

CKSUM

Macro Definition

: ; BPEX

Show Source File on Screen in Trace

SOURCE SOURCE'

Display and Change RAM Allocated to Screen History and to Symbols

?FREE =HISTORY =SYMBOLS

-- The Categories --

4. Special key and mode panel words-- commands that perform the same function as one of the mode panel switches, or the same as the cursor keys or a function key.

Mode Panel Access

MODE

Mode Panel 1

DASM	DASM'
SYMB	SYMB'
RESET	RESET'

Mode Panel 2

SHOWM	SHOWM'
SHOWC	SHOWC'
=MBASE	
PAGINATE	PAGINATE'
HDG	HDG'

Mode Panel 3

LOG	LOG'
TOFILE	TOFILE'
PRINT	PRINT'
NMIVEC	NMIVEC'
RSP	RSP'

Function Keys

ALT-FKEY?	CTRL-FKEY?	DEF
FKEY?	MEMO	SHIFT-FKEY?
WSIZE		

Cursor Keys

TOP/BOT

-- The Categories --

5. Rarely used words-- commands that you will probably not often use.

Trigger Commands

ASEG	CONT	CONTROL
INT	INT'	NDATA
SC	TNT	

Standalone Operation

SST	TS
-----	----

PreQualifier Commands

INFINITE	Q1	Q2
Q3	QUALIFIERS	TRIG

Filter Commands

FILTER	HDAT	MISC'
--------	------	-------

Emulation Memory Enable

=EMSEG

Temporary Number Base Change

B#	B.	D#
H>D		

Special Display Characteristic Commands

CLEAR	CLEAR'	COLOR
SET-COLOR		

Serial Port Setting

AUX1	AUX2
------	------

Receive HEX format file from another system

HEXRCV

Symbol File Format

SYMPFIX

PROM Burning Mode

8BIT	16BIT
------	-------

Loading from Host RAM

MLOADN

Pause for a Few Milliseconds

MS

128K UniLab Only

PAGE0	PAGE1
-------	-------

Burn PROMs in Standalone Mode

STANDALONE	PROMMSG
------------	---------

(this page intentionally left blank)

THE UniLab COMMANDS

-- The Commands --

Entry format

The First Line

The first item on the first line of each entry is the command itself, always printed in bold capital letters. The rest of the line always contains either the phrase "no parameters" or the command repeated along with its parameters.

The parameters always appear inside <pointy brackets>.

Last, if the word can be called up with a function key, the name of that function key appears on the far right hand side of the first line. If the word is rarely used, than the phrase "RARELY USED" appears on the far right.

The Definition

The first block of text tells you what the command does.

Usage

The next block of text tells you how and when you use the command-- sometimes warning you that you only want to use the word in extraordinary circumstances.

Example

Almost every command includes a section showing examples of how to use the word.

Comments

This optional section includes warnings, historical notes, and various other bits and pieces of information.

1AFTER

1AFTER <trigger spec>

Clears out previous trigger spec and enables trace filtering. Only the bus cycle that satisfies the trigger spec and one cycle immediately after will be kept.

USAGE

The UniLab stores the trigger cycle and the one immediately after, every time it sees conditions that match the trigger specification. The "trigger status display line" shows how many cycles have been stored away.

Note that you have to use **S** to start the analyzer after setting this trigger spec.

The UniLab automatically displays the trace after the entire trace buffer has been filled.

The disassembler will not work properly on fragments of code. The disassembler should be disabled with **DASM'** while you are looking at the results of any of the **xAFTER** commands.

CHECKING THE TRACE

If you want to see the trace before the buffer has been completely filled, then press any key to stop the cycle recording. Then type in **TD** to dump the trace, and display part of it on the screen.

The trace buffer fills from the bottom, and each new cycle pushes up the already recorded data. If you end up with a partially filled buffer, then the cycles you want to see are in the last part of the buffer.

EXAMPLES

1AFTER 1200 ADR S

shows only those cycles with adr =1200 and one cycle following.

1AFTER 235 TO 560 ADR S

shows 2 consecutive cycles each time a cycle has an address between 235 and 560.

(continued on next page)

-- The Commands --

(continued from previous page)

COMMENTS

Do not put a space between the number and **AFTER**.
1AFTER is a single word, not a word preceded by a parameter. This command can be used when seeking the cause of a memory cycle error. It will show the program address of the cycle after the one that caused the memory access. **xAFTER** initializes all trigger features, so **NORMx** is unnecessary with these commands.

16BIT

no parameters

Selects 16-bit mode for memory emulation and for trace display and for PROM burning and reading.

USAGE

You will probably not use this command. It sets up the UniLab to work with processors that have a 16 bit data bus. If you have purchased a disassembler, then either this command or **8BIT** has been "built-in" to your software.

COMMENTS

Note that **16BIT** is one word with no space after the 16. The **16BIT** command changes both the signals put onto the target system's bus by the UniLab and the way the UniLab displays the trace display. That means you need a 28 pin ROM emulation cable, or the 16 bit emulation will not work.

The **HL** and **LH** commands determine the order in which the trace displays the bytes. If you have a disassembler these modes have already been set for you.

2AFTER **2AFTER** <trigger spec>

Same as **1AFTER** except that two cycles are kept immediately following each trigger cycle.

USAGE

Enables a filtered trace that gives you a little more information than **1AFTER** does.

COMMENTS

See **1AFTER**.

3AFTER **3AFTER** <trigger spec>

Same as **1AFTER**, except that the three cycles after the trigger cycle get stored.

USAGE

Enables a filtered trace that gives you a little more information than **2AFTER** does.

COMMENTS

See **1AFTER**. And notice that this filtered trace will contain enough information to make a disassembled trace sensible-- sometimes.

8BIT no parameters
Selects 8-bit mode for trace display and memory emulation and for PROM burning and reading.

USAGE

You will probably not use this command. It sets up the UniLab to work with processors that make use of 8 bit data. If you have purchased a disassembler, then either this command or **16BIT** has been "built-in" to your software.

COMMENTS

Use the 24 pin ROM cable with this command. Note that **8BIT** is one word, with no space between the number 8 and the rest of the command.

: no parameters

The colon character starts a macro definition. The word that follows the colon is the name of the macro.

USAGE

Once a macro has been defined, you can execute any lengthy series of commands with a single word. See Appendix F for further information. See also BPEX.

WHAT A MACRO IS

A macro is a command that you create out of previously defined commands.

For example,

```
: LOADUP      0 TO 3FFF BINLOAD A:MYPROG ;
```

creates a macro called **LOADUP**, which uses the previously defined UniLab command **BINLOAD**.

LOADUP will always load from a file on drive A: called myprog. You can see how this would be easier than using **BINLOAD** every time you wanted to load this file.

HOW TO WRITE MACROS

A macro definition begins with a colon and ends with a semicolon (;). The first word after the : is the name of the macro, and all the other words are the definition of it.

There must be at least one space between the colon and the name of the macro, and at least one space between the last word and the semicolon. Like this:

```
: NAME FIRSTWORD SECONDWORD VALUE THIRDWORD ;
```

FORTH

When you define a macro, you are actually making use of the programming language FORTH. With this powerful language you can define new words that make use of conditional statements, looping, and more. The best introduction to the language is Leo Brodie's Starting FORTH.

(continued on next page)

(continued from previous page)

WHY MACROS

The example below defines a macro called READRAM. After the new word has been defined, you would just type in READRAM every time you want to set up the trigger specification that shows only the cycles that read from the address range 1000 to 1FFF. This will save you a lot of keystrokes.

EXAMPLE

```
: READRAM ONLY READ 1000 TO 1FFF ADR S ;  
      defines a macro called READRAM.
```

COMMENTS

Whenever the word immediately following **:** is entered the result is the same as if the rest of the words up to **;** were entered. After typing in the example above, the word READRAM will have the same effect as entering " ONLY READ 1000 TO 1FFF ADR S ." Note that to preserve the macro definition, you must **SAVE-SYS** before leaving the UniLab program.

See also appendix F.

```
;                                no parameters
```

Ends a macro definition started by **:** .

-- The Commands --

=BC <word> =BC

Changes the contents of the BC register to n.

USAGE

An example of the type of register control command available with debuggers. This command addresses the Z-80 internal register BC. Consult the target notes for your debugger.

EXAMPLE

1234 =BC
puts 1234 in the BC register.

COMMENTS

You can use the register commands only after the debugger has gained control of your microprocessor. See **NMI** or **RB** for more information on debug control.

This is a typical register changing instruction format. A similar command is provided for each of the processors internal registers (except SP). No space appears between the = and the register name.

=EMSEG <hex digit> =EMSEG

Sets A16-A19 context for subsequent **EMENABLE** statement(s).
Determines which 64K "bank" of memory the emulated ROM will be
in.

YOU PROBABLY DON'T NEED TO BOTHER

This value must be set properly, or the UniLab will not
put the program opcodes onto the target system bus.
However, if you have a disassembler/debugger, then this
variable is already set properly.

WHY IT MIGHT MATTER

Though the upper 4 bits of our 20-bit address bus are
meaningful only with processors that can address more
than 64K of memory, =EMSEG must always be set.

On some microprocessors, those four lines are floating
high, on other mp's several of the lines are pulled
low.

HOW IT WORKS

This command only sets a variable. **EMENABLE** is the
command that actually enables memory.

WHEN IT MATTERS

The UniLab looks at the upper 4 bits of address (A16
through A19) during fetch and read cycles, to determine
whether your microprocessor wants to fetch an
instruction from emulation ROM. If the upper 4 bits
that the UniLab sees don't match the =EMSEG
specification, then the UniLab will not respond to the
mp's request.

Use **ESTAT** to see how this command effects the settings
of emulated memory.

EXAMPLES

7 =EMSEG
sets A19 to 0 and A16, A17, and A18 to one.

(continued on next page)

-- The Commands --

(continued from previous page)

F =EMSEG 0 TO 1FFF EMENABLE
enables addresses F0000 to F1FFF.

E =EMSEG 0 EMENABLE ALSO F =EMSEG 0 EMENABLE
enables emulation of addresses
E0000 - E07FF and F0000 - F07FF.

COMMENTS

The 4 most significant bits of the 20 bit UniLab enable addressing are selected with =EMSEG so that subsequent statements only refer to 16-bit addresses. EMENABLE commands enable emulation memory in blocks of 2K.

A read or fetch command from the target microprocessor will reference emulation memory only when the A16-A19 inputs agree with an =EMSEG statement and A11-A15 indicate an enabled 2K block of emulation ROM.

Note that latched inputs A16-A19, displayed on the trace display as the right-hand digit of the CONT column, are the values seen by the emulation enable logic. If the inputs are not connected then they will "float," and appear as all 1s (hex F).

=EMSEG itself has no effect on the UniLab until an **EMENABLE** or **INIT** sends the data to the UniLab.

=HISTORY <hex# of Kbytes> =HISTORY

Selects the size of the screen history saved during each session with the UniLab.

USAGE

Allows you to change the amount of host RAM dedicated to saving information that scrolls off the top of the screen. The maximum is hexadecimal 3C Kbytes (decimal 60).

The new setting will not take effect until you **SAVE-SYS**, exit from the UniLab software, and start it up again.

Use **?FREE** to find out how much is allocated right now.

WHY CHANGE

You might want to have a longer history, or you might want to free up some of the host RAM for other purposes.

EXAMPLE

3C =HISTORY
allocates the maximum space to the line history.

=MBASE <n> =MBASE F8

Selects number base for the trace display of the MISC inputs to the UniLab, M0 through M7.

USAGE

The miscellaneous inputs (MISC) to the UniLab usually get displayed in binary format. This format allows you to easily tell which MISC inputs are receiving a high signal, and which are receiving a low.

This command also changes the number base for the HDATA column for 8 bit processors.

However, you might have an application for these inputs, such as reading the data from onboard RAM, where a hex or decimal display would be more useful.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

The panel only toggles between binary and hex.

EXAMPLES

- 10 =MBASE**
hexadecimal display, the most space efficient
- 8 =MBASE**
selects octal display mode.
- A =MBASE**
selects decimal display mode.
- 2 =MBASE**
returns to binary display mode.

COMMENTS

The MISC inputs can be connected to any signals you like.

Note that **A**, not **10**, must be used to specify decimal ten.

=SYMBOLS <hex # of Kbytes> =SYMBOLS

Selects the amount of space allowed for symbol tables within the UniLab software.

USAGE

Allows you to change the amount of host RAM dedicated to storing the symbol table. The maximum is hexadecimal 80 Kbytes (decimal 128).

The new setting will not take effect until you **SAVE-SYS**, exit from the UniLab software and start it up again.

Use **?FREE** to find out how much is allocated right now.

WHY CHANGE

You might want to have a larger symbol table, or you might want to free up some of the host RAM for other purposes.

EXAMPLE

80 =SYMBOLS
allocate the maximum space you can to the symbol table.

?FREE no parameters

Displays the amount of host RAM allocated to the screen history and to the symbol table. Also shows how much host RAM is currently free.

USAGE

Find out how much you can increase the amount of space dedicated to history or symbol table, or whether you need to reduce it. See **=HISTORY** and **=SYMBOLS**.

ADR

<word> ADR
<word> TO <word> ADR

Sets up the trigger specification for analyzer inputs A0 through A15. (Sets trigger for A0 to A19 if five-digit address ends in a period.)

USAGE

Determines which 16 bit addresses the analyzer will trigger on. Can also trigger on 20-bit addresses.

With **TO** the trigger will occur on the address range from ADR1 to ADR2.

If **NOT** precedes the value(s) of the address, the UniLab will trigger outside of the specified address or range of addresses.

All previous entries to the address trigger spec are erased unless you precede this spec with the word **ALSO**.

Note that you can inadvertently produce "cross products" when making use of **ALSO** with **ADR**. See the fourth example below.

EXAMPLES

NORMT 1023 ADR S
trigger on address 1023. **NORMT** causes the trigger to appear at the Top of the trace.

NOT 120 TO 455 ADR S
trigger if address outside 120-455 range.

12345. ADR S
trigger on 20-bit address 12345. The 1 will appear in right digit of the CONT column.

1200 ADR ALSO 8 ADR
sets the analyzer to trigger when the address is 1200 or 0008. Because of cross products, will also trigger on address 0000 and 1208.

(continued on next page)

(continued from previous page)

COMMENTS

ALSO must be used with caution with **ADR**. Generally you can use **ALSO** once, if the high-order byte of the previous spec and the new one match. To do more than that you should work with the two bytes of the address separately using **HADR** and **LADR**.

AS is a convenient abbreviation for **NORMT ADR S**.

You can define a 20-bit address trigger by ending the number in a period. See **ASEG** for another approach to 20-bit addresses.

ADR?

no parameters

Displays random examples of the addresses seen on the bus-- approximately two every second.

USAGE

This command displays two of the addresses that appear on the bus each second. A useful command for getting a rough-grained idea of how the program behaves.

Terminate the display by pressing any key.

EXAMPLE

ADR?

This command is never used in combination with anything else.

COMMENTS

Useful for monitoring program flow in a rough manner. For example, it will be obvious to you in the target program gets stuck in a loop. **ADR?** turns **RESET** mode off and sets up a trigger spec of its own. Be sure to use **NORMx** at the start of the first trigger spec after using this word.

AFTER AFTER <qualifier specification>

Sets the stage for the description of a qualifying event. Qualifying events are bus states that must be seen before the analyzer starts to search for the trigger.

USAGE

When you have specified qualifying events, the UniLab will not recognize the trigger until after the "qualifiers" have been seen.

You can set up to three qualifying events. Each qualifier spec must start with **AFTER**.

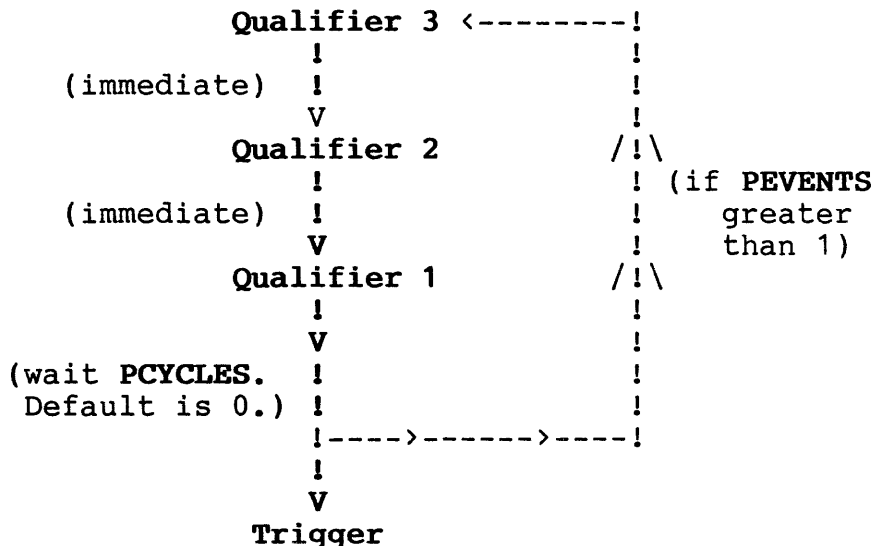
All the qualifiers must appear on the bus one immediately after another, without intervening bus cycles. However, the trigger itself can appear anytime after all the qualifiers have been satisfied.

You cannot use MISC inputs as qualifiers.

DELAYS AND REPETITIONS

You can specify a minimum number of bus cycles after the time the last qualifier is seen, before the UniLab starts looking for the trigger. See **PCYCLES**. The default is **0 PCYCLES**.

You can also specify a number of complete repetitions of the sequence of qualifiers. See **PEVENTS**. The default is **1 PEVENTS**.



(continued from previous page)

EXAMPLES

NORMT 100 ADR AFTER 535 ADR S

will trigger on address 100 only after address 535 gets seen on the bus.

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.

NORMT 100 ADR AFTER 535 ADR AFTER 3F DATA S

a single statement with the same result as the two above.

NORMT AFTER NOT 345 ADR AFTER 344 ADR S

triggers if any address other than 345 follows immediately after 344. By starting with **AFTER** we are able to describe two events which must follow one another without intervening bus cycles.

COMMENTS

Equivalent results can be obtained by using <n> **QUALIFIERS** to set the number of qualifiers. The four related commands **TRIG**, **Q1**, **Q2**, and **Q3** can then be used to set the various triggers. But **AFTER** is the more natural way to do it.

You will find **Q1**, etc., handy when you want to "change context" to alter the description of an event that you though you had completed.

AHIST

no parameters

Address **HIST**ogram invokes the optional histogram generator that allows you to display the relative time your target program falls into each of up to 15 user-specified address ranges. See also **THIST**.

USAGE

Allows you to examine the performance of your software. You can find out where your program is spending most of its time.

Press **F10** to exit from this menu-driven feature.

You must (only once) issue the command **DOHIST** to enable this optional feature. **DOHIST** performs a **SAVE-SYS**, and then causes an exit to DOS. The next time you call up the software, both **AHIST** and **THIST** will be enabled.

MENU DRIVEN

You produce a histogram by first specifying the upper and lower limits of each address "bin" that you want displayed, then starting the display.

When you give the command **AHIST** you get the histogram screen with the cursor positioned at the first bin. You can then start typing in the lower and upper limits of each bin. Use return, tab or an arrow key after you enter each number, to move to the next entry field.

Press function key 1 (**F1**) to start displaying the histogram.

SAVE TO A FILE

You can save the setup of a histogram as a file with the **HSAVE** <file>. Issue this command after you exit from the histogram.

You load the histogram back in with **HLOAD** <file>. Issue this command before invoking the histogram.

EXAMPLE

AHIST

This command is never used in combination with anything else.

ALSO no parameters

Used with both **EMENABLE** and with trigger specification commands. Prevents clearing of previous settings.

USAGE

The trigger spec commands, **CONT**, **ADR**, **DATA**, **HDATA**, **HADR**, **LADR** and **MISC**, normally cause the UniLab to trigger on the new conditions instead of the old conditions. By using **ALSO**, you can instruct the UniLab to trigger on the old conditions OR the new conditions.

The memory enable command, **EMENABLE**, normally enables only the new settings of memory. By using **ALSO**, you can enable both the old range of memory and the new.

Note that you have to use **ALSO** for each new setting that you declare. See the second example below.

ALSO is not necessary when you want to trigger on several different categories. The UniLab will automatically AND together the specifications in different categories.

Note that you can inadvertently produce "cross products" when making use of **ALSO** with **ADR**. See **ADR**.

EXAMPLES

12 DATA ALSO 34 DATA

sets the analyzer to trigger on either 12 or 34 data (without the ALSO only 34 data would remain set).

10 DATA ALSO 5 DATA ALSO 3 DATA 1200 ADR

sets the analyzer to trigger when the data is 10 or 5 or 3 and the address is 1200.

0 TO 7FF EMENABLE ALSO 2000 TO 2FFF EMENABLE

enables two ranges of emulation ROM.

COMMENTS

Applies only to the first **EMENABLE** or trigger spec command that follows.

-- The Commands --

ALT-FKEY <# of key> ALT-FKEY <command>

Assigns a command to an **ALTERed** function key.

USAGE

Reassign the function keys on PCs and PC look-alikes. Use **ALT-FKEY?** (or hit F1 while holding down ALT) to find the current assignments.

The function keys allow you to execute any command or string of commands with a single keystroke. The initial assignments represent our best guess at what you will need. But you might want to change them.

To make your reassignments permanent, use **SAVE-SYS**.

EXAMPLE

2 ALT-FKEY WSIZE
assigns WSIZE to ALT-F2.

COMMENTS

To execute a string of commands, define a macro first (using :) and then assign the macro to the function key.

See also **FKEY**, **CTRL-FKEY**, and **SHIFT-FKEY**.

ALT-FKEY? no parameters ALT-F1

Displays the current assignments of the **ALTERed** function keys.

USAGE

Whenever you want to be reminded what command will be executed when you press a function key while holding down the ALT key.

See **ALT-FKEY** to reassign the keys.

AS <addr> AS

An abbreviation for **NORMT ADR S**.

USAGE

Defines an analyzer trigger spec, and starts the analyzer working. The trigger event appears near the top of the trace as cycle zero. A useful abbreviation-- saves you key strokes. When entering the most common trigger spec-- triggering on a code address.

Will not work on ranges of addresses (with **TO**) or with **NOT**.

EXAMPLE

1234 AS
triggers when address is 1234

COMMENTS

The macro definition of this command:
: AS NORMT ADR S ;

-- The Commands --

ASC no parameters SHIFT-F4

Displays the handy reference ASCII table.

USAGE

Shows each character, along with its decimal and hex value.

EXAMPLE

ASC

This command is never used in combination with anything else.

COMMENTS

This is a bonus feature provided to save you the trouble of hunting for a printed ASCII table.

ASEG <hex digit> ASEG RARELY USED

Sets a trigger spec on address bits A16-A19. Note that ASEG cannot be used with NOT, ALSO, or TO.

USAGE

Normally, you set a trigger address with **ADR**, either a 16 bit or 20-bit address. This command allows you to set a trigger on the upper 4 bits of the 20 bit address. See **=EMSEG** for a longer discussion of the addressing scheme of the UniLab.

EXAMPLES

5 ASEG
 requires a hex value of 5 on A16-A19 for trigger.

COMMENTS

Normally useful only if you have over 64K of memory in your target system. Even then, a better way to define a trigger on a 5-digit address is just to enter the 5-digit address ending in a period followed by ADR.

The command "n ASEG" has the same effect as "F MASK n CONT."

ASM <address> ASM <instruction>

Invokes the processor-specific assembler.

USAGE

Patch assembly language code to the given address in emulation ROM. Allows you to overwrite locations in the copy of your target program residing in the UniLab's emulation ROM, so that you can quickly fix bugs when you find them. Note that the assembler writes over memory-- it does not insert instructions.

If you do not include the address, **ASM** will refer to the current value stored by the **ORG** command.

ASSEMBLING MULTIPLE INSTRUCTIONS

If you do not include an assembly language instruction, then **ASM** will give you as a prompt the address to which it is assembling, and wait for you to give it an instruction followed by a carriage return.

The assembler will continue to prompt you with an address and patch assembled code into memory, until you feed a blank line (hit return on an empty line).

CONVENTIONS

The on-line assembler will only accept assembly language instructions, not ORIGIN statements or EQU statements. (You should use the UniLab command **IS** to define symbols.)

Only one instruction per line.

The normal conventions of assembly language apply. For example, at least one space between the instruction and the operands.

BASE

The default number base is hexadecimal, as it is throughout the UniLab software. You can change the base by storing a new value in the variable **BASE**. For example, to change to decimal base type in:

A BASE !

(continued on next page)

(continued from previous page)

EXAMPLES

0 ASM LD SP,3000

alters the first instruction of the LTARG program of the Z80 package.

100 ASM

invokes the assembler, starting at address 100. The assembler will prompt you with that same address, and wait for you to enter an assembly language instruction.

ASM-FILE <addr> <start screen> <end screen> ASM-FILE

Invokes a version of the on-line assembler that assembles code contained on the screens of a FORTH file.

USAGE

A way to make large patches to your program, or to write prototype code without leaving the UniLab environment-- or just to write a few lines that you will want to be able to edit and re-enter.

ASM-FILE follows the same conventions as **ASM**.

You can include comments on a screen by putting a semicolon (;) on a line. The assembler will ignore everything after the semicolon on that line. The semicolon must be the first character on the line, or be preceded by at least one space.

FORTH FILES AND THE EDITOR

If you only have a few lines of code, you can use the screen that **MEMO** puts you into, and the two following (screens 1D through 1F). See the entry for **MEMO** to get a few pointers on using the FORTH screen editor.

OPENING A NEW FILE

You will want to put the code into a file of its own if you have many lines of code, or if you want a more convenient way to archive the code.

First close the current file (**UniLab.SCR**) with the command **CLOSE**.

Next create a new file with **OPEN-NEW** <file name>, and determine its size with <# of screens> **SCREENS** (1K allocated per screen). Use the command <screen #> **EDIT** to get into the file. Don't make use of screen zero.

You will then be able to use **ASM-FILE** to assemble the code stored in your new file.

When you are done with assembling, use **OPEN UNILAB.SCR** to close your file and re-open the UniLab.SCR file. If you don't do this, then some of the on-line help facilities and error messages will not work.

(continued on next page)

(continued from previous page)

EXAMPLES

1200 1D 1F ASM-FILE

loads assembly code, starting at address 1200,
from screens 1D through 1F of the currently opened
FORTH file.

1 4 ASM-FILE

loads code from screens 1 through 4, starting at
the current value of **ORG**.

AUX1 no parameters

Tells the host computer to look for the UniLab on serial port 1.
This is the normal default condition.

AUX2 no parameters

Tells the host computer to look for the UniLab on serial port 2.
Use this command if you can't get the UniLab to initialize.

B# **B#** <binary number>

Interprets the number following as a binary number.

USAGE

Useful when you want to input a number as a binary-- saves time with pencil and paper. Quick, what is the hex value of a number with 1 at locations 0, 3, 7, 9 and 10? Let the computer do that work for you.

EXAMPLES

B# 0101010001001
has the same effect as entering 0A89H

NORMT B# 1111110 MISC S
will trigger when the MISC inputs are 11111110

COMMENTS

Changes the base to binary, just for the next number. Allows entering numbers in binary format, just as D# allows decimal format.

B. **<hexadecimal number> B.**

Displays the hex number as a binary number.

USAGE

When you want to find out the binary equivalent of a hex number, saves you time with pencil and paper.

EXAMPLE

A89 B.
displays the binary equivalent of A89, which is 0101010001001.

BINLOAD <from addr> <to addr> BINLOAD <filename>

Loads a binary file from disk into emulation memory. Prompts you for the name of the file if you don't include it on the command line.

USAGE

Starts loading a binary file into the **from addr**. Stops loading at the **to addr**, or when end of file is reached. The binary file should contain a program. Can be used to load the product of a cross compiler into emulation memory.

This command fully supports DOS pathnames.

You can save a program to a file with **BINSAVE**.

EXAMPLE

0 400 BINLOAD \ASM\MAIN.BIN
loads a binary DOS file, starting at location 0
and ending at location 400.

COMMENTS

Loads exact binary contents of file until DOS indicates end of file, or the "to address" is reached. If you don't know the ending address, you can just enter FFFF as toadr and loading will stop on end-of-file.

As with all memory writing commands, don't write into your stack area when loading into RAM.

Use with .COM,.BIN, or .TSK files. See **HEXLOAD** for Intel Hex files.

You can use the DOS command **EXE2BIN** to convert .EXE files into .BIN files.

The system can load to target RAM-- if debug control has been established (see **RB**).

-- The Commands --

BINSAVE <1st addr> <2nd addr> BINSAVE <file name>

Saves the specified section of memory as a file. Prompts you for the file name if you do not include it.

USAGE

This command saves the program memory to disk. Saves everything in memory between the first address and the second address.

This command fully supports DOS pathnames.

EXAMPLE

100 4FF BINSAVE
saves target locations 100 - 4FF.

COMMENTS

Saves exact binary contents of a range of target memory as a named file. This file can later be re-loaded with the **BINLOAD** command.

Can save from target RAM, but only if debug control has been established. See **RB**.

BPEX

BPEX <macro name>

Executes the specified macro at each breakpoint, after the register display.

USAGE

Allows you to automatically execute any command or group of commands, at every breakpoint. You must first define a macro, or use one of the pre-defined Orion command words.

BPEX will not accept a string of commands, only the first word that follows. This means that only certain commands are suitable-- those that require no parameters. In the example below, we first write a macro that requires no parameters, called **SEE-RAM**. Notice that **SEE-RAM** makes a call to **MDUMP**, which does require parameters.

See : for more info on macros.

TURN IT OFF

To turn off the automatic execution use **BPEX NOOP**.

EXAMPLES

: SEE-RAM 8000 8080 MDUMP ;
defines a macro called **SEE-RAM** which dumps out 80 memory locations.

BPEX SEE-RAM
executes your macro at every subsequent breakpoint.

COMMENTS

Available only with debugger packages. Useful if, for example, you want to watch a memory window as you single step through the program.

Note that you must define a macro first because **BPEX** patches in only the single word following it.

-- The Commands --

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.

EXAMPLE

BYE

This command never used in combination with anything else.

CATALOG no parameters

Displays a directory of all the available pinouts-- the proper cable hook-ups for each microprocessor.

USAGE

Once this word is entered, any of the listed pinouts can be displayed on the screen.

This word "opens" the pinout library. It closes again as soon as you enter another command.

Until you use this command, the only pinout diagram available is that of the mp you are using. You get that with the command **PINOUT**.

CKSUM <from addr> <to addr> CKSUM

Calculates the checksum for a given range of memory. Useful for error-checking.

USAGE

A good way to make a PROM easy to check for burn-in errors, or corrupted locations. Allows you to record the checksum of your program-- or better yet, make the checksum equal to zero.

EXAMPLE

800 FFF CKSUM
prints a 16-bit checksum for the data in addresses
800-FFF

COMMENTS

You may want to patch the complement of this value into your PROM. You can produce a PROM with a checksum of zero, using the following method, which sacrifices only two bytes.

First store zero where the checksum will be (0 FFE MM! in the above example). Second, find the checksum, using **CKSUM**. Lastly, patch in the complement of the sum.

For example, if the sum is 1234, then use the command **-1234 FFE MM!**. The resulting PROM will have a checksum of 0.

-- The Commands --

CLEAR

no parameters

Clears the screen before performing a **PgUp**. Use with some of the older color monitor cards, that will otherwise flicker when you use **PgUp**.

CLEAR'

no parameters

The normal default condition-- the screen is not cleared before a **PgUp** is executed. Use only to restore the default condition after executing a **CLEAR**.

CLRMBP no parameters

Clears all multiple breakpoints.

USAGE

Use to wipe the slate clean, and start out setting multiple breakpoints again. **SMBP** sets the breakpoints.

EXAMPLE

CLRMBP

This command never used in combination with anything else.

COMMENTS

Use to clear all the numbered breakpoints which you set with **SMBP** and can clear one at a time with **RMBP**.

CLRSYM no parameters

Clears out the current symbol table.

USAGE

When you want to get rid of the symbols that you have defined for your program. It's a good idea to first save the symbols, just in case you decide you want those symbols after all. See **SYMSAVE**.

The symbol table also gets cleared by **SYMFILE** and **SYMLOAD** before they load in the new symbols. **SYMFILE+** adds to the existing symbol table.

Unless you save the symbols, you cannot recover them later. You could instead use **SYMB'**, which turns off the symbol table without erasing it.

EXAMPLE

CLRSYM

This command never used in combination with anything else.

COMMENTS

You might want to clear out the table before loading in a new one from a file. See **SYMFILE** and **SYMLOAD**.

COLOR no parameters

Displays in color. Only has an effect with a color monitor.

USAGE

Turns on color display.

You have to save the system afterward, if you want the UniLab program to start up with color display.

CHANGING COLORS

Use the UniLab command **SET-COLOR**, which shows you what the new settings are as you change them.

You will have to save the system with **SAVE-SYS** if you want to preserve the new colors.

EXAMPLES

COLOR

This command never used in combination with anything else.

COM1 no parameters

Enables dumb terminal emulation mode, using serial communications port 1 of your personal computer. This is the port normally used by the UniLab.

USAGE

Allows you to use your PC as a dumb terminal while within the UniLab software. Press the ESCape key to exit.

COMMUNICATION SETTINGS

The default settings are:

300 baud

8 bits, 2 stop bits, no parity.

CHANGING SETTINGS

You can change these settings by changing the values stored in two constants, **BR2** (Baud Rate) and **LCR2** (Line Control Register: bits per character, etc.).

Put the value 60 into **BR2** to change to 1200 baud:

60 ' BR2 !

You may miss characters at 1200 baud, due to the screen scroll times. Put a 180 into **BR2** to change back to 300 baud.

You can change to 7 bits, 2 stop bits with:

6 ' LCR2 !

TABLE OF SETTINGS

<u># bits</u>	<u>parity</u>	<u>#stop bits</u>	<u>value to store at LCR2</u>
7	None	1	2
7	None	2	6
7	Odd	1	A
7	Odd	2	E
7	Even	1	1A
7	Even	2	1E
8	None	1	3
8	None	2	7
8	Odd	1	B
8	Odd	2	F
8	Even	1	1B
8	Even	2	1F

(continued on next page)

(continued from previous page)

To change to 5 or 6 bits per character look at the information on the Line Control Register of the INS8250 in a reference manual on that chip, or in the Hardware Technical Reference Manual for your computer.

COM2

no parameters

Enables dumb terminal emulation mode, using serial communications port 2 of your personal computer. See the entry for **COM1** for details.

USAGE

Allows you to use your PC as a dumb terminal while within the UniLab software. Press the ESCape key to exit.

Change the communications settings the exact same way that you do for **COM1**.

-- The Commands --

CONT

 <byte> CONT
 <byte> TO <byte> CONT
 <byte> MASK <byte> CONT

Sets up the analyzer trigger spec for the CONT inputs (control lines C4 - C7, and A16 - A19).

USAGE

The CONT input lines actually represent two different types of information. The upper four bits represent the processor cycle type. The lower four bits come from the four highest address lines, A16 through A19.

When you precede it with one number, CONT causes the UniLab to trigger when the inputs equal that number. When you use **TO** the UniLab triggers on any value from **m** to **n**. **NOT** causes the UniLab to trigger when the value falls outside of the specified range or value.

You can use **k MASK l** to examine any subset of the 8 input lines. See Comments below for more details.

Unless you use **ALSO** the previous trigger spec gets cleared out.

EXAMPLES

B# 00011111 CONT

requires C7-C5 = 0, C4 & A19-A16 = 1.

70 TO 7F CONT

requires C7=0 and C6-C4 = 1, A19-A16 any value.

F MASK 3 CONT

requires A19 & A18 = 0, A17 & A16 = 1, C7-C4 any value.

(continued on next page)

(continued from previous page)

COMMENTS

The low four bits of the CONT lines refer to the highest four bits of the address-- the same segment address bits set by =EMSEG.

When you use the command `k MASK l CONT`, the value of `k` determines which bits the UniLab will examine-- the bits with a value of one. The `l` then indicates the value those lines must have before trigger occurs. For example, **FO MASK FF** tells the UniLab to only look at the upper 4 bits of the CONT lines. The **AF** tells the UniLab to trigger when bits 7 and 5 are high while bits 6 and 4 are low. Note that the UniLab will not care about the value of the lower four bits.

CONTROL no parameters RARELY USED

Used before **FILTER** to set up a filter spec based only on the **CONT** inputs.

USAGE -- RARELY USED

You will probably never use this command. Triggers on the full specification, but filters based only on the 8 bits of the **CONT** inputs.

The filter mechanism of the UniLab gets turned on for you by the **xAFTER** macros. Those commands set the filter to **MISC' FILTER**, which allows you to set up a trigger spec based on all inputs except for the MISCellaneous wires.

See also **HDAT** and **MISC**.

THE CONT INPUTS

The upper four bits identify processor cycle type, while the lower four bits identify the address bits A19-A16.

This command makes it possible to filter on cycle type and on memory segments.

EXAMPLE

NORMT CONTROL FILTER WRITE 1200 ADR A7 DEVENTS S
triggers on 1200 address, and then records only writes. Note that you have to use **DEVENTS** to get a trace buffer full of the event you are filtering on.

CTRL-FKEY <# of key> CTRL-FKEY <command>

Assigns a command to a function key hit while the CTRL key is held down.

USAGE

Reassign the function keys on PCs and PC look-alikes. Use **CTRL-FKEY?** (or CTRL-F1) to find the current assignments.

The function keys allow you to execute any command or string of commands with a single keystroke. The initial assignments represent our best guess at what you will need. But you might want to change them.

To make your reassignments permanent, use **SAVE-SYS**.

EXAMPLE

5 CTRL-FKEY DOS
assigns DOS to CTRL-F5.

COMMENTS

To execute a string of commands, define a macro first (using :) and then assign the macro to the function key.

See also **FKEY**, **ALT-FKEY**, and **SHIFT-FKEY**.

CTRL-FKEY? no parameters CTRL-F1

Displays the current assignments of the ControLled function keys.

USAGE

Whenever you want to be reminded what command will be executed when you press a function key while holding down the CTRL key.

See **CTRL-FKEY** to reassign the keys.

CYCLES? <from addr> <to addr> **CYCLES?**

Counts the number of bus cycles between two addresses.

USAGE

Can use to count the number of bus cycles in a loop, as in the first example below, or the "distance" between two addresses.

BUS CYCLE COUNT

Not the number of machine cycles, nor the number of instructions fetched, but instead the number of reads and writes that occur between one command and another. The read could be instruction fetches, or could be data fetches.

EXAMPLES

123 123 CYCLES?

counts cycles between two occurrences of the address 123.

123 456 CYCLES?

counts cycles between address 123 and address 456.

12300. 12450. CYCLES?

counts cycles between address 12300 and address 12450.

COMMENTS

Useful for checking quickly whether a loop works as you intended. **CYCLES?** makes its own trigger spec, so you will have to start fresh on your trigger after using this command. Use one of the **NORMx** commands to clear out the trigger spec set by **CYCLES?**.

When specifying a five-digit address, the . which designates a five-digit address must be used with both addresses.

D# **D#** <decimal number>

Treats the number that follows as a decimal value, rather than as a hexadecimal, which is the default.

USAGE

Saves you the trouble of converting the number by hand or with a calculator.

EXAMPLES

D# 16 ADR
equivalent to entering "10 ADR".

D# 32 .
will display 20 (the hex equivalent of 32 decimal).

D# 135 B.
converts a number from decimal to binary.

D# 1000 MS
will pause 1 second.

COMMENTS

See also **B#** for entering binary numbers.

-- The Commands --

DASM

no parameters

F8

Enables the trace disassembler.

USAGE

Turns on the translation of machine code into assembly language mnemonics. You will usually want to keep this on, only turning it off for special applications such as x**AFTER**. To turn off the disassembler, use **DASM'**.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

EXAMPLE

DASM

selects disassembled mode for trace display.

COMMENTS

Works only if you have an optional disassembler installed.

DASM' no parameters F8

Disables the trace disassembler.

USAGE

Turns off the translation of hexadecimal machine codes into assembly language mnemonics. See **DASM** above for more details.

Typically you will use the **MODE** panel (function key 8) when you want to change this feature.

EXAMPLE

DASM' selects hex mode for trace display.

-- The Commands --

DATA

 <byte> DATA
 <byte> TO <byte> DATA
 <byte> MASK <byte> DATA

Changes the analyzer trigger for the DATA inputs (D0 to D7).

THE DATA INPUTS:

The UniLab gets both the address and the data from the bus during each memory read and write. The "data" that appears on the bus could be either a value or a machine code instruction. See COMMENTS below for information on triggering on a 16-bit data bus.

USAGE

The simplest use sets up a trigger for a single data value. The UniLab will search for the byte value, and trigger when it sees that hex number on the bus as data. See the first example below.

RANGES OF DATA:

TO lets you set up a trigger on any data between two byte values, inclusive. See the second example below.

NOT

NOT causes the UniLab to trigger when the value falls outside the specified range or value.

MASKING

You can use **k MASK l DATA** to examine any subset of the 8 data lines. The high bits of **k** mark which bits will be examined, while the bit configuration of byte **l** indicates the values the lines must have for a trigger to occur.

For example, **80 MASK FF DATA** selects only the highest data bit for examination (with binary value 1000 0000). The UniLab would trigger when this bit has a high value. Note that the instruction **80 MASK 80 DATA** would have the same effect.

(DATA continued on next page)

(continued from previous page)

EXAMPLES

NORMT 12 DATA S

after clearing all previous settings with NORMT, sets up a trigger for data input 12, and then uses S to start the analyzer.

12 TO 34 DATA

requires data value between 12 and 34 hex.

F0 MASK 30 DATA

sets a trigger based only on the four highest bits of data. UniLab will look for a 3 on those lines.

23 DATA ALSO 45 DATA

sets a trigger on cycles where data is either 23 or 45 hex.

COMMENTS

The data inputs (D0-D7) are normally connected via the emulator cable at the ROM socket. On 16-bit processors DATA is only half of the data bus, while HDATA is the other half.

If you need to use a large number of ALSO terms, then see NDATA.

DCYCLES <number of cycles> DCYCLES

Sets number of cycles the UniLab will continue to record after the trigger.

USAGE

When the UniLab sees the trigger event on the target board, it consults the delay cycles variable to determine how many more cycles to record. Each time a new cycle enters the trace buffer you lose the oldest recorded cycle. After the UniLab records the specified number of cycles, it shows the trace buffer on the screen.

WHY YOU DON'T NEED TO BOTHER

This command gets executed by a number of other commands. **NORMT**, for example, sets the delay value to A0 (160 decimal). That delay count puts the trigger event near the top of the trace buffer, after the ten cycles that came just before it.

WHY YOU MIGHT WANT TO

You might want to see the trace starting 260 cycles after a known event-- perhaps you don't know where the program ends up at that time. The **DCYCLES** command will do the job perfectly.

EXAMPLE

104 DCYCLES
selects 104 (hex) delay cycles (260 decimal)

COMMENTS

NORMT, **NORMM**, and **NORMB** select A0, 55, and 4 DCYCLES respectively. **S+** increases the number of delay cycles by A6, so you can see what happens after the end of the current trace.

The maximum possible delay count is FFFF.

DEF no parameters F5

Returns the window to the size last set with **WSIZE**, or to the default if you have not changed the window size.

USAGE

The help screens readjust the window size, to make the lower window as large as possible without overwriting the information in the upper window. After you have used a help screen, you might want to return the Default window size. Just press Function key 5.

SAVING A DEFAULT

You can use **SAVE-SYS** to save all the current settings at any time.

EXAMPLES

DEF

This command never used in combination with anything else.

DM <start address> <count> DM

Disassembles <count> number of lines, starting at the given address.

USAGE

Allows you to check that emulation memory has the proper data stored in it, and that the trace shows the same instructions as the stored program.

See also **DN**.

EXAMPLE

100 10 DM
disassembles 10 lines starting at address 100

COMMENTS

Normally disassembles from ROM. Works only if you have an optional disassembler. Can disassemble from target RAM once debugger has control. See **RB**.

DMBP no parameters

Displays the settings of all eight multiple breakpoints.

USAGE

When you forget the settings of your multiple breakpoints. Automatically executed whenever you set one of the 8 multiple breakpoints with **SMBP**.

EXAMPLES

DMBP

This command never used in combination with anything else.

DN <start address> DN

Disassembles the number of lines necessary to fill the right-hand side of current window.

USAGE

When you want to see your code, and keep it on the screen while looking at a trace. This command is similar to **DM**.

EXAMPLES

20F0 DN
Fills the right side of the current window with assembly language code, starting from address 20F0.

DOS DOS <DOS command>

Execute a DOS command from the UniLab program.
Or, use with no parameters to exit to DOS temporarily. Return to UniLab program by typing EXIT.

USAGE

When you forget the name of the file where you stored that program, or have any other reason to use the DOS utilities. You can either execute a single command, or you can go to DOS and execute a series of commands.

If you go to DOS, you can re-enter the UniLab program. Return to the UniLab program by typing **EXIT** at the DOS prompt (A> or B> or C>).

If you use **BYE** to exit the UniLab program, you have to start it up again by typing **ULxx** at the DOS prompt.

EXAMPLES

DOS DIR /w
Executes the DOS command "DIR /w."

DOS
Allows you to execute any series of DOS commands, then return to the UniLab program.

DOS ASMB SOURCE.ASM OBJECT.BIN
Assembles a new version of the program you are working on.

EMCLR

no parameters

Tells the UniLab not to emulate ROM-- clears out the emulation memory settings.

USAGE

Commands the UniLab to not respond to any microprocessor requests for data or instructions. Use only when you want to run a program from on-board ROM.

Before you can run a program from a ROM chip on your board, you will need to disable the debugger. Use the SWI VECTOR choice on the mode panel(F8) or RSP'. Note that you can use the PROM READER MENU (F9 from the MAIN MENU) to read a program into emulation memory from a ROM chip.

EXAMPLE

EMCLR

This command never used in combination with anything else.

EMENABLE <address> EMENABLE
<from address> TO <to address> EMENABLE

Enables emulation memory, needed before you can load in a program. But first, set **=EMSEG** properly.

USAGE

With a single address, enables the 2K memory region that includes the given address. Note that **=EMSEG** just sets a variable in the host's memory, while **EMENABLE** sends all the information to the UniLab.

You can use **SAVE-SYS** to make the current settings permanent.

ON RANGES OF ADDRESSES

TO enables the emulation memory from the beginning of the 2K segment that includes the <from> address to the end of the 2K segment that the <to> address is in.

CLEARING PREVIOUS SETTINGS

Unless you precede emulation statement with **ALSO**, clears out previous **EMENABLE** statements.

WATCH OUT

When you try to emulate two separate ranges of memory, you can accidentally overlay the two. For example, with a 32K UniLab, 0 and 8000 reference the same memory location in the UniLab.

Of course, you can enable areas that do not overlap. For example, 0 TO 3FFF and also C000 TO FFFF would not conflict.

EXAMPLES

F =EMSEG O EMENABLE
enables target addresses 0-7FF with A16-19 all set high.

O TO 1FFF EMENABLE ALSO FFFF EMENABLE
enables 0-1FFF and F800-FFFF

F =EMSEG O EMENABLE ALSO E =EMSEG O EMENABLE
enables locations F0000 - F07FF and E0000 - E07FF

(continued on next page)

(continued from previous page)

COMMENTS

The UniLab's enable logic first compares the A16-A19 value from the most recent =EMSEG statement with the present bus address. Address inputs A11-A15 then get compared to an enable map, where each entry corresponds to a 2K segment of memory. When both the segment and the 2K block are enabled, the UniLab accepts the address, and puts its data on the bus.

ESTAT no parameters

Tells you the current status of emulation memory.

USAGE

When you want to find out what range of addresses is currently enabled.

EXAMPLES

ESTAT

This command never used in combination with anything else.

EVENTS? no parameters

Starts the analyzer and counts occurrences of the currently defined trigger event.

USAGE

Useful for monitoring occurrences that you don't need a trace of. An excellent way to see whether the program does what it should. If the program messes up spectacularly, or performs flawlessly, then this command will show you that.

Otherwise, you're left in the dark.

EXAMPLES

NORMT 123 ADR EVENTS?
counts occurrences of address 123.

NORMT 123 ADR FF DATA EVENTS?
counts occurrences of data FF when the address is 123.

NORMT WRITE 78 TO FF DATA 1210 ADR EVENTS?
Counts the number of times a data value greater than 78 gets written to location 1210.

COMMENTS

You can also count such things as error conditions or system usage.

You can use this command if you want to sync a scope on the UniLab's test point output.

FETCH

no parameters

Tells the UniLab to look for trigger event only during fetch cycles.

USAGE

To search for a particular opcode. After you give it this command, the UniLab will not look for the trigger event during reads or writes.

This command is not available on all processors.

This command is used as part of a trigger spec, as shown in the examples below.

EXAMPLES

NORMT FETCH 120 ADR S

triggers when the program fetches from address 120.

NORMT FETCH NOT 0 TO 7FF ADR S

triggers if the program tries to fetch an instruction from outside the 0 to 7FF range.

COMMENTS

This command, loaded with the disassembler, specifies a range of CONT values corresponding to fetch cycles.

FILTER no parameters RARELY USED

Selects trace filtering mode, according to previous word: **CONTROL**, **HDAT**, or **MISC**'.

WHY YOU DON'T NEED TO BOTHER

For most filtering of the trace, you will use commands such as **ONLY** or **xAFTER**. These words automatically select the **MISC**' filtering mode for you. The **NORMx** words turn off filtering.

You can use this command to set up a filter spec that is different from your trigger spec. This is sometimes a very useful thing to be able to do.

EXAMPLE

NORMT CONTROL FILTER READ 1200 ADR A7 DEVENTS S
triggers when the processor reads from address 1200-- then produces a filtered trace of the A7 (hex) read cycles that occur after that.

COMMENTS

You would want to bother when inventing your own filtering command.

FKEY <# of key> **FKEY** <command>

Assigns a command to a function key.

USAGE

Reassign the function keys on PCs and PC look-alikes. Use **FKEY?** (or F1) to find the current assignments.

The function keys allow you to execute any command or string of commands with a single keystroke. The initial assignments represent our best guess at what you will need. But you might want to change them.

Note that you have to use "A" (hexadecimal) as the number to assign a command to **F10**.

To make your reassignments permanent, use **SAVE-SYS**.

EXAMPLE

2 FKEY STARTUP
assigns **STARTUP** to the F9 key.

COMMENTS

If you find yourself performing some one action repeatedly, you can save time by making it into a macro and then assigning it to a function key. For example

```
: DUMP100 0 100 MDUMP ;  
6 FKEY DUMP100
```

will allow you to dump locations 0 to 100 by pressing function key 6.

See also **ALT-FKEY**, **CTRL-FKEY**, and **SHIFT-FKEY**.

FKEY? no parameters F1

Displays the current function key assignments.

USAGE

Whenever you want to be reminded what pressing a function key will do for you.

See **FKEY** to reassign the keys.

EXAMPLES

FKEY?

This command never used in combination with anything else.

G <address> G

Goes to the indicated address. Exits debugger, lets the target run.

USAGE

After you have set a breakpoint, and want to release debug control and let the target run. G is one of several ways to do this.

G just gets the target board going. After that, you can enter a trigger spec and restart the analyzer, or you can use one of the "big picture" words: **ADR?**, **SAMP**, or **NOW?**.

You could instead use **STARTUP** to restart the analyzer and the board at the same time. Or use **NORMx** followed by a trigger specification and **S**, to restart the analyzer and give you a trace of the event that you describe.

EXAMPLE

1030 G
exits from debug control, and resumes the target program at location 1030.

COMMENTS

Appropriate only if you have an optional debugger and have established control by entering **RESET adr RB**, or **NMI**. You can return to any point in the program you like, but debug control will be lost.

Use **GB** if you wish to resume the program at an address different from the one you are stopped at but with another breakpoint set.

GB <addr to go to> <bpoint addr> GB

Goes to the first address, and starts executing code, with a breakpoint set at the second address.

USAGE

When you want to move around the program without losing debug control.

EXAMPLES

1200 330 GB
resumes the program at address 1200, with a breakpoint set at 330.

COMMENTS

Available only if you have an optional debugger and have established debug control. See **RB** to establish debugger control.

-- The Commands --

GW

<address> GW

Goes to the indicated address and waits until the analyzer is started. Releases the target board from debugger control.

USAGE

To continue the execution of the program, starting at the given address, after a new trigger spec has been defined.

A rather specialized but very useful command.

EXAMPLE

1100 GW NORMT 1200 ADR S

Goes to address 1100 and waits for the analyzer to be started. The trigger spec command sets the analyzer to capture a trace showing the code at address 1200.

H>D <hex number> H>D

Displays the decimal equivalent of a hex number.

USAGE

Shows you the decimal equivalent-- compare this with **D#**, which allows you to enter a decimal number that will then be used by the next command.

This word is similar to **B**, which shows you the binary equivalent of a hex number.

EXAMPLE

10 H>D
will cause "16" to be displayed.

333 133 - H>D
will display "512," which is the decimal equivalent of 333 minus 133 (hex).

-- The Commands --

HADR RARELY USED < byte > HADR
 < byte > TO < byte > HADR
 < byte > MASK < byte > HADR

Changes the analyzer trigger for the high-order byte of the 16-bit address (A8-A15).

THE ADDRESS INPUTS

You should normally use **ADR** to set 16 or 20 bits at once, but there are limits to the use of **ALSO** in combination with **ADR**.

The UniLab gets both the address and the data from the bus during each bus cycle. The UniLab works with up to 20-bit addresses. You can change the trigger specification of the least significant byte with **LADR**, the second byte with this command, and the high four bytes with **CONT** or **ASEG**.

USAGE

You can use this trigger spec command in the same way you use **DATA**, **CONT**, etc.. However, the most frequent use of this command is to set up a trigger spec on the address lines that makes use of many calls to **ALSO**.

EXAMPLES

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.

COMMENTS

Makes it possible to treat the first two bytes of the address separately. **LADR** is the lower half.

HDAT no parameters RARELY USED

Used before **FILTER** to set up a filter spec based only on the high byte of the DATA inputs (D8 - D15).

USAGE -- RARELY USED

You will probably never use this command. Triggers on the full specification, but filters based only on the 8 bits D8 through D15.

The filter mechanism of the UniLab gets turned on for you by the **xAFTER** macros. Those commands set the filter to **MISC' FILTER**, which allows you to set up a trigger spec based on all inputs except for the MISCellaneous wires.

See also **CONTROL** and **MISC**.

THE HIGH DATA INPUTS

These lines read from the high byte of the 16-bit data path of 16-bit processors. On 8-bit processors, the lines can be left to float, or be used to sense other logic signals on your target board.

USAGE

Used to show only the cycles that meet your description. While deciding whether to include the current cycle in a filtered trace, the UniLab will check only these 8 bits of the 48 inputs.

A good way to look at all the bus cycles that have some specific data value as the upper byte of data.

EXAMPLE

NORMT HDAT FILTER 80 TO FF HDATA 3410 ADR A7 DEVENTS S
will give a trace showing only those cycles with D15 high, starting with the bus cycle that has D15 high and address 3410. Note that you have to use **DEVENTS** to get a trace full of the event you are filtering on.

HDATA < byte > HDATA
< byte > TO < byte > HDATA
< byte > MASK < byte > HDATA

Changes the analyzer trigger for the high byte of 16-bit data path (D8 through D15). Spare inputs on 8-bit processors.

THE DATA INPUTS

The UniLab gets both the address and the data from the bus during each bus cycle. The "data" that appears on the bus could be either a value or a machine code instruction. On 8-bit processors the inputs D8 through D15 can be hooked up to anything you like.

USAGE

The simplest use sets up a trigger for a single value on the high order byte of the data inputs. The UniLab will search for the byte value, and trigger when it sees that hex number on the bus as data.

Note that just looking at the high order byte means that the UniLab doesn't care about the low order byte, and so actually searches for a range of values. See the first example below.

To specify just one full 16 bit wide data value, you must use both **HDATA** and **DATA**.

RANGES OF DATA

TO lets you set up a trigger on any data between two byte values, inclusive. See the third example below.

NOT

NOT causes the UniLab to trigger when the value falls outside the specified range or value.

MASKING

You can use <i> **MASK** <j> **HDATA** to examine any subset of the 8 most significant data lines. The high bits of i mark which bits will be examined, while the bit configuration of byte j indicates the values the lines must have for a trigger to occur.

For example, **01 MASK FF HDATA** selects only data bit D8 for examination (with binary value 0000 0001). The UniLab would trigger when this bit has a high value. Note that the instruction **01 MASK 01 HDATA** would have the same effect.

(**HDATA** continued on next page)

(continued from previous page)

EXAMPLES

NORMT 12 HDATA S

after clearing all previous settings with NORMT, sets up a trigger for data input 12XX -- actually 1200 through 12FF-- then uses S to start the analyzer.

12 HDATA 80 DATA

sets a trigger for only data 1280.

12 TO 34 HDATA

requires data value between 12XX and 34XX hex. That is, 1200 through 34FF.

F0 MASK 00 HDATA

sets a trigger based only on the four highest bits of data. UniLab will look for a 0 on those lines.

12 TO 23 HDATA ALSO 45 HDATA

sets a trigger on cycles where the highest byte of data is either 12 to 23, or 45 hex.

COMMENTS

You must use a special 16-bit cable with processors that use a 16-bit data bus. That cable has two ROM plugs-- one for the even byte, one for the odd byte.

If you need to use a large number of ALSO terms, then see **NDATA**.

The HDATA inputs are named for their use in the 16BIT mode. In the 8BIT mode they are displayed as a separate column and can be used as for anything you like just like the MISC inputs. On eight-bit systems they are typically used to look at system inputs and outputs.

-- The Commands --

HDG no parameters F8

Has a fixed header for trace displays-- one that does not scroll up with the rest of the trace.

USAGE

One of the display attributes. Usually you will toggle this with the mode panel, function key 8.

HDG' no parameters F8

Makes a non-fixed header for trace displays-- one that scrolls with the rest of the trace.

USAGE

One of the display attributes. Usually you will toggle this with the mode panel, function key 8.

HELP

HELP <command>

F1

Finds the reference information for a command or feature. With no word, brings up the help screen, including soft-key prompt line.

USAGE

Look up information on a command, in the abridged on-line command reference. See also **WORDS**.

EXAMPLES

HELP

displays help screen.

HELP BYE

gives information on command "bye."

HEXLOAD

HEXLOAD <file name>

Loads an Intel HEX format object file into the UniLab's emulation memory. Prompts you for the file name if you don't include it.

USAGE

Load into emulation memory a program stored in Intel HEX format. You can then run, debug and alter that program as you would any other.

Binary format files are more compact and load two to three times faster. You might want to direct your assembler to produce binary format files, if it has that capability. Or you can save your program memory with **BINSAVE** to produce a binary format file.

Binary format files are loaded with **BINLOAD**.

Intel HEX format files contain the information about where each opcode should be stored. Be certain to have the proper sections of emulation memory enabled before loading in the file. See **EMENABLE**.

LOADING INTO RAM

The UniLab will not load a file into RAM unless you have first established debug control. To do that you must first have a program already loaded into emulation memory (**LTARG** for example) and then run to a breakpoint with **RESET** <address> **RB**.

If the debugger is not in control, attempts to load memory that is not enabled will generate an error message and will not be loaded. Enabled areas in the same file will be loaded.

EXAMPLE

HEXLOAD MYPROG.HEX

load an Intel HEX format file called MYPROG.HEX.

(continued on next page)

(continued from previous page)

COMMENTS

Only record types 0 to 3 are supported. Bytes 7 and 8 of each line of the file tell what record type that line uses.

16-bit processor note: If the UniLab detects a type 2 (extended address) record then address bits A16-A19 will be compared to the current =EMSEG and data will not be loaded if it is intended for some segment other than the current one. This will be indicated by a "not enb" message for each invalid address. Enabled addresses in the file will be properly loaded.

-- The Commands --

HEXRCV

no parameters

Loads an Intel HEX file from another computer, via a second serial port.

USAGE

The serial transmission must be done on a separate serial channel with the UniLab connected to its normal serial port. XON and XOFF characters are used to start and stop the data transmission. Transmission is normally done on COM2 on IBM PC's while the UniLab is connected to COM1.

EXAMPLE

HEXRCV

loads a hex file serially

INFINITE

INFINITE PEVENTS

RARELY USED

Used only before **PEVENTS**, instead of a count, to indicate that the trigger event must immediately follow the qualifying events.

USAGE

Along with a trigger specification (see **ADR**, **DATA**, **READ**, **WRITE**, etc.) and a qualifying event specification (see **AFTER** or **QUALIFIERS**), when you are only interested in the trigger event if it occurs immediately after the qualifying events.

BACKGROUND

The default is for the UniLab to search for the qualifying sequence only once. After the sequence has been found once, it is discarded and the UniLab looks for the trigger.

With **PEVENTS** and a normal count, the UniLab searches for the qualifying events until it finds them the count number of times. Then it discards the qualifiers, and looks only for the trigger.

WHAT IT REALLY DOES

INFINITE causes the UniLab to search for the qualifying sequence and then immediately look for the trigger event. If the trigger event is not the very next cycle, then the UniLab starts looking for the qualifiers again.

EXAMPLE

123 ADR AFTER 345 ADR INFINITE PEVENTS
triggers if address 123 follows immediately after address 345.

COMMENTS

Pretty obscure. But might be highly useful in certain restricted situations.

Pressing any key stops the search.

INIT no parameters

Sends an initialization message to the UniLab.

USAGE

To reset the UniLab after you are in the UniLab program.

When you start up the program, it tries to initialize the instrument after the screen has been cleared and the UniLab version number displayed. If you tap any key after the screen is cleared, then the automatic init will not occur. You will then have to use **INIT** before you can send any commands to the instrument.

Also, if the UniLab was not properly connected when you called up the program, or if you turned off the UniLab at any time during the program, then the UniLab needs to be initialized.

IF IT FREEZES

If the program stops after printing the "Initializing UniLab. . . ." message, hit the BREAK key while holding down the CONTROL key. This breaks you out of the initializing sequence. Make certain that you have turned on the UniLab and connected it to the host computer.

Try **INIT** again. If it still freezes up, check the Trouble Shooting Chapter.

EXAMPLES

INIT

This command is never used in combination with anything else.

COMMENTS

Initializes all of the mode bits, baud rate and emulation enable map. Sent automatically after PROM programmer operations to re-initialize the analyzer modes.

INT no parameters **RARELY USED**

Enables NMI- interrupt output when trigger state reached.

USAGE-- RARE

Available only on processors that have a non-maskable hardware interrupt feature. If you want the target system to execute an interrupt routine when it goes into trigger search state(i.e., after the "qualifier has been found). Used to prevent damage to equipment by branching control to a "soft shutdown" routine when some error condition occurs.

You must write and install your own shutdown routine.

Note that Orion debuggers use this command internally. If you want to make use of it, you must disable the **NMI** feature of the Orion debugger with the Mode Panel (**F8**) or with **NMIVEC'**.

NORMx disables the **INT** mode.

EXAMPLES

NORMT INT AFTER 123 ADR S

will interrupt the target processor during the bus cycle after address 123 is reached, then trigger immediately.

NORMT INT 12 DATA AFTER 345 ADR S

will interrupt during the bus cycle after address 345 occurs, then the analyzer will trigger when 12 data occurs.

COMMENTS

Note that the interrupt occurs when the qualifying sequence is complete not on analyzer trigger. This makes it possible to trigger on something specific after the interrupt occurs.

-- The Commands --

INT'

no parameters

RARELY USED

Disables the **INT** mode.

USAGE

Rare.

COMMENTS

Not often used since **NORMx** also disables the **INT** mode.

IS <value> **IS** <name>

Assigns a symbol name to an address or data value.

USAGE

To show mnemonic names of memory locations on traces. If you already have an assembler generated symbol table, you will prefer to use the symbol table features of the UniLab. See **SYMFIX** and **SYMFILE**.

You can use the **IS** command to add symbols after you have loaded in a symbol table. **IS** turns on symbol display mode.

EXAMPLE

1234 IS MREGISTER
gives 1234 the symbol name "MREGISTER"

COMMENTS

Used to manually create a symbol table or to add symbols to an existing table.

Use **SYMB** to enable the symbol display on trace. See also **SYMB'**, **SYMSAVE**, **CLRSYM**, **SYMLOAD**, and **SYMFILE**. Symbol translation will work with or without a disassembler.

LADR <byte> LADR RARELY USED
 <byte> TO <byte> LADR
 <byte> MASK <byte> LADR

Sets the truth table for the low order byte of the address (A0-A7) separately.

THE ADDRESS INPUTS

You should normally use **ADR** to set 16 or 20 bits at once, but there are limits to the use of **ALSO** in combination with **ADR**.

The UniLab gets both the address and the data from the bus during each bus cycle. The UniLab works with up to 20-bit addresses. You can change the trigger specification of the least significant byte with this command, the second byte with **HADR** and the high four bytes with **CONT** or **ASEG**.

USAGE

You can use this trigger spec command in the same way you use **DATA**, **CONT**, etc.. However, the most frequent use of this command is to set up a trigger spec on the address lines that makes use of many calls to **ALSO**.

LADR is also useful for setting a trigger on a port address of the Z80. The ports of the Z80 processor have only one byte addresses-- and the Z80 puts the contents of the A register on the upper byte of the address lines when it outputs to a port.

EXAMPLE

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.

COMMENTS

Makes it possible to treat the first two bytes of the address separately. **HADR** is the upper half.

LOG no parameters F8

Enables automatic logging of target program patches on printer.

USAGE

Keeps a record of any program patches you make, but other operations are not logged to the printer.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

LOG' no parameters F8

Disables logging of program patches to printer. See **LOG** above.

USAGE

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

LP no parameters

Goes around a loop once and stops.

USAGE

You must already have established debug control (see **RB**), and be stopped at a breakpoint within a loop. This command allows the program to run once around the loop and stop at the current address, displaying the registers as the UniLab does for any breakpoint.

WATCH OUT

Will not work if the program counter register is pointing above the first instruction or below the last instruction in the loop. Only works when you are within the loop.

EXAMPLES

LP

This command never used in combination with anything else.

COMMENTS

Works by first saving the current breakpoint address, executing **N** (a single step without following branches) and then executing < saved address > **RB**. Note that this will not work if you are at the end of the loop.

LTARG no parameters

Loads a simple target program into the UniLab's emulation memory.

USAGE

A good way to gain familiarity with the UniLab. Comes packaged with the disassembler. This command enables the proper section of emulation memory and loads a simple program. A **STARTUP** command then sets the analyzer and your target system going.

WATCH OUT: PROCESSORS WITH EXTERNAL STACKS

The **LTARG** program uses the memory map of the Orion demo board. If your target system was designed with the RAM and ROM resources at different addresses, then the **LTARG** program might not run on your board without some patching.

See **LTARG** in the reference section of your debugger notes if you have a problem.

EXAMPLE

LTARG

This command never used in combination with anything else.

COMMENTS

Available only if you have an optional disassembler. The program loaded by this command is used in the debugger writeup as a demonstration. This makes it possible for you to duplicate exactly the demo in the target-specific notes.

If you are having trouble using the debugger with your program, try using it with the **LTARG** program. Note that the ROM that occupies the addresses used by the **LTARG** program must be unplugged to prevent bus conflicts.

M <byte> M

Stores one byte in ROM or RAM and increments reference address.

USAGE

Used after an **ORG** statement (which sets up address), to patch program memory. Can only be used to change RAM after debug control has been established. See **RB**.

REMEMBERING CHANGES

LOG sends all memory access commands such as **M** to the printer, saving a record of any changes you make.

EXAMPLES

3000 ORG 12 M
stores a 12 at 3000

150 ORG 5 M 10 M
stores 5 at location 150, 10 at 151

COMMENTS

Used for entering data tables, program patches, etc. See also **MM**, **MM!**, and **M!**.

Will store to emulated memory if the address is enabled, otherwise will attempt to store to target RAM if the optional debugger is in control (see **RB**).

As with all memory writing commands, don't write into your stack area when loading into RAM.

M! <byte> <address> **M!**

Stores a byte of data at the specified address.

USAGE

Used to patch program memory. Does not require a previous **ORG** command-- instead requires an address as the second parameter. See **M**. Can also be used to change RAM, but only after debug control has been established. See **RB**.

REMEMBERING CHANGES

LOG sends all memory access commands, such as **M!**, to the printer, saving a record of any changes you make.

EXAMPLES

12 3000 M!
stores a 12 at 3000.

5 150 M! 10 150 M!
stores 5 at location 150, 10 at 151.

COMMENTS

Used for entering small patches-- anything larger than one byte can be done by one of the other memory patch commands with fewer keystrokes. See also **MM**, **MM!**, and **M**.

Will store to emulated memory if the address is enabled, otherwise will attempt to store to target RAM if the optional debugger is in control (see **RB**).

If the debugger is not in control, you will be told:
"Debug Control not established!"

-- The Commands --

M? <address> M?

Displays the byte that is stored at the specified address.

USAGE

To find out what is stored at a single memory location, either ROM or RAM. Use **MM?** for looking at words, and **MDUMP** or **DM** for larger areas of memory.

EXAMPLES

1210 M?
displays the byte stored at 1210.

COMMENTS

If the address is **EMENABLEd** then emulation memory will be displayed, otherwise the UniLab will attempt to use the debugger to display target RAM contents.

MASK <byte> MASK <byte>

Specifies a mask for the trigger spec that immediately follows.

USAGE

A modifier to **ADR**, **CONT**, **DATA**, **HADR**, **HDATA**, **LADR**, or **MISC**.

The first byte describes which of eight wires to pay attention to-- a one means pay attention, a zero means don't care.

The second byte tells the UniLab what inputs to look for on the wires that you care about. The UniLab ignores the bits for the inputs that the first byte told it to ignore. Thus **01 MASK 01** has the same affect as **01 MASK FF**.

EXAMPLES

NORMT 2 MASK 2 MISC S
will trigger if input M1 goes high.

NORMT B# 0010 MASK B# 0010 MISC S
has the same effect as the first example-- will trigger if input M1 goes high.

NORMT 3 MASK 2 MISC S
requires inputs M1=1, M0=0 for trigger.

COMMENTS

MASK cannot be used with TO, NOT, ALSO

MCOMP <start addr> <end addr> <comp addr> MCOMP

Compares two areas of memory and indicates discrepancies.

USAGE

Compares the two areas of memory, and gives you a message about each discrepancy. Hit any key to abort. For example:

```
110 117 810 MCOMP
      Data is 16 at addr 0110 ..but is 5 at addr 0810
      Data is 90 at addr 0112 ..but is 80 at addr 0812
      Data is 27 at addr 0116 ..but is 23 at addr 0816
```

You only need to enter three addresses-- the starting and ending address of the first block of memory, and the starting address of the second.

VERIFYING ROMS

If you want to compare a ROM to a program on disk, first load the program using **BINLOAD** or **HEXLOAD**. After that use the PROM READER MENU to read from the PROM into a different memory area.

You can then use **MCOMP** to compare the two target areas.

EXAMPLE

```
100 300 800 MCOMP
      compares data at target addresses 100-300 to the
      data at 800-A00.
```

COMMENTS

Works on any combination of emulated ROM and, if the debugger is in control, target RAM.

Both areas must be in the same 32K block of memory-- that is, A15-A19 must be the same for both sections.

MDUMP <from addr> <to addr> MDUMP

Display the contents of an area of memory.

USAGE

Allows you to look at any block of memory. For example:

121 131 MDUMP

```
121  F3 31 00 1C 21 78 02 11  78 02 01 2C 00 7C BA C2  .1..!x..x... ..
131  38 01 7D BB CA 42 01 7E  12 23 13 0B 79 B0 C2 38  8...B. .#..y..8
```

Press any key to freeze scrolling of display. Press any key again to continue scrolling. While scrolling is stopped, press any key twice quickly to stop.

EXAMPLE

1234 1334 MDUMP

displays the contents of locations 1234 to 1334 in hex and ASCII.

COMMENTS

As with all M commands, display will be from emulation memory if the address has been EMENABLED. If the debugger is in control, you can also display target RAM memory.

-- The Commands --

MEMO

no parameters

SHIFT-F2

Displays and allows editing of the on-line memo pad.

USAGE

A handy way to write notes to yourself. Hitting CONTROL and Z at the same time toggles the on-line editor help screen on and off. This screen shows you the ESCape key sequences and ConTRL key combinations that you use with the editor. See COMMENTS below.

You exit the full screen editor with ESCAPE followed by F if you want to save the changed memo pad. ESCAPE followed by ESCAPE allows you to leave the memo pad without saving your changes.

EXAMPLE

MEMO

This command never used in combination with anything else.

COMMENTS

This command works only when the EDITxx.VIR file is present in the same directory as the UniLab program.

The powerful editor allows you to write complicated macros and enable them at will. If you want to use this feature to the fullest, order the PADS manual from
Mountain View Press
PO Box 4656
Mountain View, CA 94040

(continued on next page)

(continued from previous page)

EDITOR HELP (repeated on-line):

Hit **SHIFT-F2** to get the editor.
Once in the editor, hit **CTRL-Z** to get the on-line help.

HIT WHILE HOLDING DOWN THE CONTROL KEY:

CURSOR CONTROL:

S=Left	D=Right	
E=Up	X=Down	Q=Home
F=Rtab	I=Ltab	
F=Forwd	A=Bkwrđ	

CHARACTER CONTROL: LINE CONTROL:

Del>Delete char	K=Kill line
J=Jerk-->buffer	G=Gobble-->buffer
C=Chars<--buffer	Y=Copy-->buffer
V=Insert chars	L=Line<--buffer
P=Pullup words	N=New lines

HIT THE ESCAPE KEY AND FOLLOW WITH:

ESC=Esc no update	F=Updat & Fin edit
W=Word for search	B=Updat & Back scr
S=Search screens	N=Update & Nxt scr
U=Update now	L=Update & Load
R=Restore screen	

MENU no parameters F10

Selects the menu-driven mode.

USAGE

The menu- driven mode helps first time users by allowing you to use the UniLab simply by choosing from list of options. This command, whether typed in or picked with function key 10, reassigns the function keys and shows the menu on the screen. The command line that you would use gets displayed as it is executed, so you can learn how to enter the command directly.

While using the menu, you can also type commands directly.

Menu mode also comes in handy when you have forgotten a command.

All PROM programming commands are available under the PROM menu.

Hitting F10 again from the main menu gets you out of menu mode.

EXAMPLE

MENU

This command never used in combination with anything else.

MESSAGE no parameters

Gives a screenful of information on the most recent updates and additions to the UniLab software.

USAGE

Make certain that you know all the capabilities of the UniLab software.

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. You can fill an area of memory, and then examine it with **MDUMP**.

One way to find out what is happening on your board when **LTARG** test program will not run: fill a block of memory with NOOP instructions, starting at the reset address, and then use **STARTUP**. You should see a trace of consecutive addresses.

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.

EXAMPLE

1200 1300 20 MFILL
fills locations 1200-1300 with the value 20 hex.

COMMENTS

To fill target RAM a debugger must be in control.

As with all memory writing commands, don't write into your stack area when loading into RAM.

MISC

 <byte> MISC
 <byte> TO <byte> MISC
 <byte> MASK <byte> MISC

Changes the analyzer trigger for the miscellaneous inputs.

THE MISCELLANEOUS INPUTS

The UniLab's 48-bit-wide trace buffer has room for 8 more bits than are used for data, address, and control lines. These eight input lines are available to you, for sensing anything on the target board that you want to know about, or that you want the UniLab to trigger on.

For example, you might hook them up to an output port, to trigger when a particular bit configuration gets asserted on that port.

Note: The qualifier and filter specifications always ignore the **MISC** inputs.

USAGE

The simplest use sets up a trigger for a single value on miscellaneous inputs. The UniLab will search for the byte value, and trigger when it sees that hex number on the lines. See the first example below.

RANGES

TO lets you set up a trigger on any input between two byte values, inclusive. See the second example below.

NOT

NOT causes the UniLab to trigger when the value falls outside the specified range or value.

MASKING

You can use **k MASK l MISC** to examine any subset of the 8 miscellaneous lines. This is particularly handy when you only have one or two of the MISC inputs connected to your board. You don't care about the logic level of the other 6 lines, since they don't mean anything.

The high bits of **k** mark which bits will be examined, while the bit configuration of byte **l** indicates the values the lines must have for a trigger to occur.

(continued on next page)

(continued from previous page)

For example, **03 MASK FF MISC** selects only bits M0 and M1 for examination (with binary value 0000 0011). The UniLab would trigger when both these bits have a high value. Note that the instruction **03 MASK 03 MISC** would have the same effect.

WITH TRACING

All trace filtering modes and qualifiers ignore the MISC inputs. Since they still effect triggering, this makes the MISC inputs particularly useful as trigger inputs for filtered traces.

EXAMPLES

NORMT 12 MISC S

after clearing all previous settings with **NORMT**, sets up a trigger for miscellaneous input 12, then uses **S** to start the analyzer.

12 TO 34 MISC

requires miscellaneous input value between 12 and 34 hex.

F0 MASK 00 MISC

sets a trigger based only on the four highest bits. The UniLab will look for a 0 on those lines.

23 MISC ALSO 45 MISC

sets a trigger on cycles where the misc input is either 23 or 45 hex.

ONLY 100 TO 400 ADR FF MISC

traces only cycles where **ADR** is 100-200. Triggers when **MISC** is **FF**. Filtering ignores **MISC**.

COMMENTS

The MISC inputs can be connected to anything you like. They are often used to look at system input and output ports.

MISC'

MISC' FILTER

RARELY USED

Used only before **FILTER** to enable trace filtering on all inputs except the MISCellaneous wires(M0 to M7). **NORMx** turns this mode off.

WHY YOU DON'T NEED TO BOTHER

Because this is taken care of for you by **ONLY** and by **xAFTER**, so it is unlikely that you will need to use this command.

See also **CONTROL** and **HDAT**.

EXAMPLE

MISC' FILTER

enables filtering on all except M0-M7 inputs.

MLOADN <from addr> <to addr> <target addr> MLOADN

Moves a block of memory from the memory of the host to the target memory.

USAGE

Allows you to assemble or load a program into host memory, and then move it to UniLab emulation ROM or target RAM.

Most people will prefer to assemble into a file, and then load from the file into UniLab emulation memory.

FREE MEMORY

The host memory area that is available generally starts right above C000. **PAD 100 + U.** displays the first free address. **SO U.** shows you the upper limit of the unused memory.

EXAMPLE

C000 C800 0 MLOADN
moves data at C000-C800 in the host computer to target locations 0-800.

COMMENTS

You must have emulation memory enabled to load the program into ROM (see **EMENABLE**).

The debugger must be in control if you want to load into target RAM.

MM

<word> MM

Stores one 16-bit word in ROM or RAM and increments reference address.

USAGE

Used after an **ORG** statement (which sets up address), to patch program memory. Can only be used to change RAM after debug control has been established. See **RB**.

REMEMBERING CHANGES

LOG sends all memory access commands such as **MM** to the printer, saving a record of any changes you make.

EXAMPLES

3000 ORG 1210 MM
stores 1210 at 3000.

150 ORG 5000 MM 7001 MM
stores 5000 at location 150, 7001 at 152.

COMMENTS

Used for entering data tables, program patches, etc. See also **M**, **MM!**, and **M!**.

Will store to emulated memory if the address is enabled, otherwise will attempt to store to target RAM if the optional debugger is in control (see **RB**).

As with all memory writing commands, don't write into your stack area when loading into RAM.

If you have a disassembler the byte order is set correctly, otherwise you can set it with **HL** or **LH**.

MM! <word> <address> **MM!**

Stores a 16-bit word of data at the specified address.

USAGE

Used to patch program memory. Does not require a previous **ORG** command-- instead requires an address as the second parameter. See **MM**. Can also be used to change RAM, but only after debug control has been established. See **RB**.

REMEMBERING CHANGES

LOG sends all memory access commands, such as **MM!**, to the printer, saving a record of any changes you make.

EXAMPLES

1200 3000 MM!
stores a 1200 at 3000

5000 150 MM! 1000 152 MM!
stores 5000 at location 151, 1000 at 153.

COMMENTS

Used for entering small patches-- anything larger than one word can be done by one of the other memory patch commands with fewer keystrokes. See **MM** and **M**.

Will store to emulated memory if the address is enabled, otherwise will attempt to store to target RAM if the optional debugger is in control (see **RB**).

As with all memory writing commands, don't write into your stack area when loading into RAM.

If you have a disassembler the byte order is set correctly, otherwise you can set it with **HL** or **LH**. Words are stored to emulation memory if it is enabled, otherwise the debugger is used (if in control) to store to target RAM.

MM? <address> **MM?**

Displays the 16-bit word that is stored at the specified address.

USAGE

To find out what is stored at a single memory location, either ROM or RAM. Use **M?** to look at bytes and **MDUMP** or **DM** for larger areas of memory.

EXAMPLE

1210 MM?
displays the word stored at 1210.

COMMENTS

If the address is **EMENABLED**, then emulation memory will be displayed. Otherwise the UniLab will attempt to use the debugger to display target RAM contents.

If you have a disassembler, the byte order is set correctly, otherwise you can set it with **HL** or **LH**.

MMOVE <start addr> <end addr> <dest> MMOVE

Moves a block of memory from one area to another in the target memory space.

USAGE

Good way to make a little more room when you need to patch some extra code into a program.

You can also use it to relocate a relocatable code module.

SMART MOVER

Automatically chooses the order of moving, to prevent overwriting caused by moving from one area to an area that overlaps. Starts moving from either the beginning or the end of the area to be moved, as necessary. See the two examples below.

The source and destination range can be anywhere in memory, but neither range can cross a 32K boundary. That is, the start and end address of a range must be within a 32K block.

EXAMPLES

1000 2000 1005 MMOVE
moves the data in locations 1000-2000 up 5 places.
Starts moving from the end.

200 300 125 MMOVE
moves the data in 200-300 down 75 spaces. Starts moving from the beginning.

COMMENTS

Make certain that the code you moved is relocatable. If it is not, you might have to patch some of the absolute address references. In general, exercise caution, and use **DM** on the moved memory, to see if the instructions still do what you want them to do.

As with all memory writing commands, don't write into your stack area when loading into RAM.

MODE

no parameters

F8

Gives you the mode panels, which allow you to change mode of display, mode of debugger functioning, etc.

USAGE

Hit function key 8 (F8) once to get the first mode panel, which contains the analyzer mode switches. Press (F8) again to get the second panel that contains the trace display mode switches. The third panel contains the log mode switches and debug disable switches.

MOVING AROUND

To get from one panel to another, hit F8 repeatedly, or use PgDn key. Use the END key to exit from mode setting.

Once you are in a pop-up panel, you can move around, selecting different features, with the up arrow and down arrow keys. The right arrow key toggles the feature on and off.

WHAT THEY ALL DO

See the Special Functions section of the manual for the complete story.

You can also check the listings in the glossary for each feature:

Panel One	DASM	SYMB	RESET		
Panel Two	SHOWM	SHOWC	=MBASE	PAGINATE	HDG
Panel Three	LOG	TOFILE	PRINT	NMIVEC	RSP

EXAMPLE

MODE

This command never used in combination with anything else.

MS <count> MS

Pauses for count number of milliseconds.

USAGE

In test programs where you need a pause.

Note that 400 hex milliseconds is one second.

EXAMPLE

800 MS
pauses for 2 seconds (800 hex ms)

N no parameters

Resumes program, with a breakpoint set to the address after the next instruction.

USAGE

While stopped at a breakpoint, when you want to execute only the next instruction pointed to by the Program Counter. Note, however, that you will "fall through" loops and branches.

This "falling through" is often very useful. For example, if the PC is pointing at a subroutine call, N will show you the state of the processor when it returns from the call.

Use **SSTEP** (see the Disassembler/Debugger writeup on your processor to make certain that your processor supports this feature) if you want to follow loops and branches.

FALL THROUGH LOOPS

When you single-step through a program, you will usually not want to bother going through loops the same number of times that the microprocessor does. This command allows you to go through a loop just once.

HOW IT WORKS

This command uses **RB** to set a breakpoint at the address just after the disassembled instruction that the PC points to. So the program runs until it reaches that address.

WATCH OUT

If the program never reaches the address of the breakpoint, then the program will run without stopping. For example, if the program contains an infinite loop, and you will not want to use **N** on the last command in the loop (the jump back up to the top). The program never reaches the code that follows that last jump.

COMMENTS

Available only when a debugger has control.

NDATA <byte #1> <byte #2> . . . <byte #N> <N> NDATA

Sets N different bytes as trigger events for the analyzer.

USAGE

A quick way to set triggers on many different data codes that do not fall into ranges. Easier than using **ALSO** again and again, as in:

18 DATA ALSO 32 DATA ALSO 36 DATA ALSO 47 DATA.

RANGES OF DATA

If the data does fall into ranges, then you can use **TO** instead. For example, **12 TO 25 DATA** sets the analyzer looking for any data between twelve and 25, inclusive.

EXAMPLE

18 32 36 47 4 NDATA

Does the same thing as the **ALSO** example in the text above.

COMMENTS

Really the same as "ORing" together the terms with **ALSO**. Any number of terms can be listed, but be sure to get the count correct.

You can use **ALSO** in combination with this command to add a range of values.

-- The Commands --

NMI no parameters F4

Establishes debug control immediately.

USAGE

Only supported on microprocessors that provide a hardware Non-Maskable Interrupt feature.

Allows you to establish debug control on a program that is currently running.

See also RB. See Appendix H to find out whether your processor supports NMI.

DISABLE

If your target board makes use of the non-maskable interrupt feature of your processor, or you wish to disable **NMI** for any other reason, use the Mode Panel (**F8**) or **NMIVEC'**.

Disabling the entire debugger (Mode panel choice "SWI VECTOR" or command **RSP'**) also disables **NMI**.

COMMENTS

The hardware interrupt feature is also utilized by **SSTEP** and **SI**. Disabling **NMI** also disables those two commands.

NMIVEC no parameters F8

Enables the Non-Maskable Interrupt vector installation.

USAGE

This command re-enables the UniLab's ability to perform **NMI**. You only want to disable this feature when you want more transparent operation and don't need to use all the debugger features. See **RSP'** for complete transparency.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

NMIVEC' no parameters F8

Disables the Non-Maskable Interrupt vector installation.

USAGE

This command disables the UniLab's ability to perform **NMI**.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

The debugger usually gives you the ability to send a non-maskable interrupt to the microprocessor. This interrupt, a standard feature on microprocessors, tells the processor to jump to whatever code location the NMI vector register contains. Orion debuggers use this feature so that you can assert debugger control over your processor at any time.

WHEN YOU WILL WANT TO DISABLE NMI

When your system makes use of the NMI, and you want to preserve that ability while testing the system.

COMMENTS

Either the panel toggle or **NMIVEC** re-enables the vector installation.

NORMB no parameters

Clears out (NORMALizes) all trigger descriptions and sets the trigger event near Bottom of trace buffer.

USAGE

To start a new trigger definition when you want to see the events that led up to the trigger.

Use **TSTAT** to look at how this command changes the **DCYCLES** setting.

When you want to start from scratch with a new trigger description, always begin with one of the variations of **NORM**. The three commands, **NORMB**, **NORMM**, and **NORMT**, vary only in where within the trace buffer they place the trigger event-- at the bottom, in the middle or at the top.

TO SEE WHAT HAPPENS NEXT

S+ restarts the target board with the same trigger specification, but with 170 (decimal) added to the delay cycle count, so that you can see what happened after the current trace window.

HOW THEY WORK

The commands clear out the truth tables the analyzer used to search for the trigger event, and set the number of delay cycles that the analyzer will wait between seeing the trigger and freezing the buffer. See **DCYCLES** for more information about delay cycles.

EXAMPLES

NORMB
Sets 4 delay cycles

NORMB NOT 0 TO 1000 ADR S
will show what happened before the address went outside of the 0-1000 range.

COMMENTS

NORMB should be used where you want to know what happened before the trigger.

NORMM

no parameters

Clears out (NORMALizes) all trigger descriptions and sets the trigger event at Middle of trace buffer.

USAGE

To start a new trigger definition when you want to see the events that led up to the trigger, and also see what followed.

You will find it very useful when you want to see the complete context within which the trigger occurred.

Use **TSTAT** to look at how this command changes the **DCYCLES** setting.

See **NORMB** for more details.

EXAMPLE

NORMM

sets delay cycles to 85 (decimal).

NORMT no parameters

Clears out (NORMALizes) all trigger descriptions and sets the trigger event near Top of trace buffer.

USAGE

To start a new trigger definition when you want to see the events that followed the trigger.

Use **TSTAT** to look at how this command changes the **DCYCLES** setting.

See **NORMB** for more details.

EXAMPLE

NORMT
sets delay cycles to 165 (decimal).

NOT **NOT** <trigger description>

The trigger description gets interpreted as a description of when not to trigger.

USAGE

To tell the analyzer to trigger when some byte of the 48-channel input bus goes outside of a certain range or value. Most commonly used to trap bad data or a bad address.

EXAMPLES

NORMT NOT 00 TO 4FF ADR S
 triggers if the address goes outside the 00 to 4FF range.

ONLY 127 ADR NOT 12 DATA S
 shows only cycles where the data at 127 address is not 12.

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

COMMENTS

Sets a flag for the next trigger word (**ADR, CONT, DATA, HADR, HDATA, LADR, and MISC**).

Except when used with **ALSO**, the **NOT** command causes the truth table to be cleared to all 1's. Then 0's get written into the specified areas. This is the opposite of what happens without **NOT**.

With **ALSO**, the **NOT** command does not clear out the truth table first.

-- The Commands --

NOW? no parameters

Shows you what is happening on the target board right now.

USAGE

To see the code the microprocessor executes during the next 170 bus cycles.

EXAMPLES

NOW?

This command never used in combination with anything else.

COMMENTS

This command is a simple macro that turns off the RESET, so that it does not restart the target board, then sets its own trigger and captures a trace.

ONLY ONLY < trigger description >

Gives you a trace buffer filled only with cycles that match your description.

USAGE

Clears out the previous trigger spec and enables trace filtering. Only the bus cycles that contain the trigger cycle will be recorded.

Use this command when you want to see on the trace only the cycle described in the trigger specification. For example, only the read cycles, or only the command at address 0100.

ELIMINATE BORING LOOPS

This command is especially useful for filtering out status and timing loops that hog the trace space. See the second example below.

Notice that when filtering you have to use **AFTER** if you want to start the trace at some particular point in the program.

ONLY AND THE DISASSEMBLER

You will sometimes want to turn off the disassembler while using this feature. Disassembling partial instructions will give confusing results. Either the mode panel (**F8**) or **DASM'** turns off the disassembler.

EXAMPLES

ONLY READ

searches for and records only the read cycles.

ONLY NOT 120 TO 135 ADR AFTER 750 ADR S

produces a trace starting at address 750, excludes from the trace the routine at addresses 120 through 135.

ONLY 0100 ADR

records only the cycle executed at address 0100.

(continued on next page)

-- The Commands --

(continued from previous page)

COMMENTS

The analyzer will run until the trace buffer is full while keeping you informed of the number of spaces remaining. You can stop the analyzer at anytime by pressing a key. Then enter **TD** to see what you have captured in the trace buffer.

ORG <address> **ORG**

Sets the origin (address at which you will start to poke new values into memory) for subsequent **M** and **MM** commands.

USAGE

To change the information stored in several sequential bytes of program or data memory.

You can alter emulation ROM at any time. However, before you can alter RAM, the debugger must be in control. See **RB**.

EXAMPLES

101 ORG 12 M 3410 MM
stores 12 to location 101 and 3410 to locations 102 & 103.

COMMENTS

Useful for entering program patches.

See also **M!** and **MM!**.

-- The Commands --

PAGE0 no parameters

Only for UniLabs with 128K of memory. Selects the bottom 64K page of emulation memory.

USAGE

Addresses that are four hex digits long (16 bit binary numbers) cover a 64K memory space, but your UniLab has 128K memory space. You must establish a context for the addresses to follow.

This command sets the offset to 0000, while **PAGE1** sets the offset to 10000. Thus, address 1300 after **PAGE0** refers to location 1300. Address 1300 after **PAGE1** means location 11300.

EXAMPLE

PAGE0

This command never used in combination with anything else.

PAGE1 no parameters

Only for UniLabs with 128K of memory. Selects the top 64K page of emulation memory.

USAGE

See **PAGE0** above.

Address 1300 after **PAGE1** means location 11300.

EXAMPLE

PAGE1

This command never used in combination with anything else.

PAGINATE no parameters F8

Enables pagination of trace display.

USAGE

The default condition. The trace stops after each screenful.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

You can turn this off with the pop-up panel, or with **PAGINATE'**.

COMMENTS

If you press any key while display is scrolling, trace display will stop.

PAGINATE' no parameters F8

Disables pagination of trace display.

USAGE

The trace display will scroll by continuously. Not very useful, unless you want to save an entire trace to a disk file. See **PAGINATE** above.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

PCYCLES

<count> PCYCLES

Sets the number of bus cycles that the analyzer waits between seeing the last qualifier and starting to search for the trigger event.

USAGE

The default is zero. Usually you will want the analyzer to start its search for the trigger event immediately after the qualifiers.

However, you will sometimes want the UniLab to wait some number of cycles after the qualifiers, before it looks for the trigger.

For example, you know that the program jumps to address 1000 from address 235. What you can't understand is why the code at address 1000 is being executed again, later on. So you do not want the UniLab to search for address 1000 until some time has passed since it saw address 235.

EXAMPLES

NORMB 1000 ADR 10 PCYCLES AFTER 235 ADR S
triggers if 1000 occurs 10 or more cycles after address 235.

COMMENTS

A pass cycle count can be used to hold off the search for a trigger, for whatever reason.

If there are several qualifiers the pass count starts after the complete sequential qualifier sequence has occurred.

PEVENTS

<n> PEVENTS

Sets the number of times the UniLab will want to see the qualifying events before starting to search for the trigger event.

USAGE

The default value is one-- the UniLab will start to search for the trigger as soon as it has seen the qualifying event once.

You would use **PEVENTS** when you don't want to search for the trigger until the qualifiers have been seen a number of times. Useful for catching a trace after the nth iteration of a sequence.

This command is different from **PCYCLES**, which delays searching for the trigger an absolute number of bus cycles after the qualifiers have been seen.

EXAMPLES

NORMT 12 DATA 4 PEVENTS AFTER 30 DATA S
searches for 12 data anytime after 30 data has been seen four times

NORMT 100 PEVENTS AFTER 123 ADR S
triggers as soon as address 123 has occurred 100 times.

-- The Commands --

PINOUT no parameters

Displays pinout of target processor.

USAGE

A handy reference showing signal names and analyzer cable connections versus pin numbers.

EXAMPLE

PINOUT

This command never used in combination with anything else.

PRINT no parameters F8

Logs all screen output to the printer.

USAGE

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

PRINT' no parameters F8

Turns off logging all screen output to printer.

USAGE

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

PROMMSG no parameters

Use after a **STANDALONE** EPROM programming command, to display completion message.

USAGE

You use **STANDALONE** when you want to make use of your host computer while the UniLab is programming an EPROM. After the programming light goes out, you can use **PROMMSG** to check the outcome of the programming operation.

Q1 Q1 <trigger spec> RARELY USED

Selects the event description (trigger spec) that follows as
qualifier one.
See **QUALIFIERS**.

USAGE

When you don't want to use **AFTER**, which you will find
to be a more natural way to set qualifiers.

You will rarely use this, since **AFTER** automatically
increments the context from **TRIG** to **Q1** to **Q2** to **Q3** each
time it is used. You will find these words handy when
you want to change your mind about one of the
qualifying steps without entering the entire definition
again.

EXAMPLES

Q1 15 LADR
Changes qualifier number one, so that the UniLab
looks for 15 on the low byte of the address lines.

Q1 ALSO 28 LADR
Alters qualifier one, so that the UniLab accepts
either 15 or 28 on the low byte of the address.

Q2 Q2 <trigger spec> RARELY USED

Selects the event description that follows as qualifier number
two. See **Q1** for details.

Q3 Q3 <trigger spec> RARELY USED

Selects the event description that follows as qualifier number
two. See **Q1** for details.

QUALIFIERS <1, 2, or 3> QUALIFIERS RARELY USED

Selects the number of qualifying events.

USAGE

Allows you to reduce the number of qualifying events. Usually you'll use **AFTER** to set qualifiers, and would use this command only to reduce the number of qualifiers if you change your mind.

When there are qualifiers, the UniLab searches for the qualifying events before it looks for the trigger.

You will probably prefer to use **AFTER**, rather than this command.

THE ORDER OF QUALIFIERS

If you have defined three qualifiers, the UniLab looks first for Q3, then for Q2 and lastly for Q1. It must see the qualifying events one immediately after the other. If it does not see one of them, it starts searching for Q3 again.

Of course, if there are only two qualifiers, then the UniLab looks for Q2 and Q1.

AFTER THE QUALIFIERS

Unless **PEVENTS** or **PCYCLES** has been set, the UniLab will immediately start searching for the trigger after it finds the last qualifier. Of course, the trigger event does not have to follow immediately after the last qualifier.

EXAMPLE

2 QUALIFIERS S
changes the number of qualifiers, so that the third one is ignored.

RB <address> **RB**

Resumes executing program, with a breakpoint set at indicated address. Must be used with **RESET** to establish debug control.

USAGE

The first breakpoint must be in emulated ROM, and come after the stack pointer has been initialized. If your program does not initialize the stack pointer, then you cannot set a breakpoint. However, setting up the stack pointer usually only takes three or four bytes.

Available only with debugger software for your processor.

You can also use **NMI** to establish debug control-- if your processor supports the Non-Maskable Interrupt feature.

MISSED BREAKPOINTS

If the breakpoint is not reached, then the program will continue to run until you hit any key. You must then use **RESET** <address> **RB** to gain debug control. You can only set a breakpoint on the address of the first byte of an instruction.

If your processor supports **NMI**, the UniLab will, after a missed breakpoint, try to achieve debug control by asserting **NMI**.

Make certain that the address you try to set a breakpoint on gets executed by the program-- set an analyzer trigger on the same address with **NORMT** <address> **AS**.

And make sure that your program does initialize the stack pointer to point at RAM. The debugger uses the stack to save the state of your system.

EXAMPLES

RESET 123 RB

enables reset, and then restarts the target system with a breakpoint set at address 123

(continued on next page)

(continued from previous page)

1007 RB

without restarting the target system, run the program with a breakpoint set at address 1007.

COMMENTS

The second example above will work only if you have already established debugger control. The first example will establish debug control, as will an **NMI** command. **RESET** does not restart your target board-- it enables the "reset" flag, so that the **RB** which follows restarts the target.

READ no parameters

Narrows the trigger specification to read cycles only.

USAGE

Instructs the UniLab to trigger only on read cycles. Handy when you want to trigger on data memory values, not program memory opcodes. Or, when you want to trigger on reads rather than writes to some address range.

On some disassembler packages, **FETCH** instructs the UniLab to trigger only on fetches from program memory.

EXAMPLES

READ 13 DATA

sets up to trigger when microprocessor reads a 13.

NORMT READ 1000 TO 2000 ADR S

triggers when processor reads any data from address range 1000H to 2000H.

COMMENTS

A simple macro which specifies a range of CONT input values. This command, like **WRITE** and **FETCH**, gets defined for a particular processor by the optional disassembler.

RES <n> RES

Clears bit n of the stimulus generator output. The number, of course, must be between 0 and 7.

USAGE

Simulates a peripheral input going from voltage high to voltage low. The stimulus generator allows you to test how your system responds to digital signals on certain lines.

EXAMPLES

2 RES
resets output S2.

1 SET 1 RES
pulses output S1.

COMMENTS

Used to reset individual bits of the 8 stimulus outputs. See also **SET** and **STIMULUS**.

RMBP <break point #> RMBP

Resets (clears) one of the multiple breakpoints and displays new status of the multiple breakpoints.

USAGE

When you want to get rid of one of the breakpoints that you set with **SMBP**.

See also **CLRMBP**, which clears out all the multiple breakpoints.

EXAMPLE

3 RMBP
clears multiple breakpoint number 3.

COMMENTS

Multiple breakpoints are used with the debugger to break on more than one address. There are 8 multiple breakpoints available in addition to the standard (unnumbered) breakpoint set by **RB** or **GB**.

-- The Commands --

RSP no parameters F8

Re-enables the debugger, after it has been disabled by **RSP'**.

USAGE

Only when you have turned off the debugger, and now want to be able to use it again. Not the same as establishing debug control, which you do with **NMI** or **RB**. However, if you have disabled the debugger, then you cannot use either of those commands.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

RSP' no parameters F8

Turns off the debugger.

USAGE

Enables complete transparency-- no emulation memory is affected by the UniLab operation.

You will have to disable the debugger if you want to run a program from a ROM chip on your target board. See **EMCLR**.

RESERVED AREA

Allows you to use for your program the areas that Orion otherwise reserves for debugger vectors and overlays. Hit **CTRL-F3** to get a help screen that includes information telling you where the reserved bytes are for your processor.

MODE PANEL

You will not be able to use the debugger until you turn it on again from the **MODE** panel, or with **RSP**.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

RZ no parameters

Resume program from breakpoint, without any breakpoints set.
Debug control will be lost.

USAGE

When you want to run the program starting from the current address.

A handy command for exiting from the debugger. However, a better command is **GW** which waits until you start the analyzer, so that you can start the program from the breakpoint with a trigger set.

EXAMPLE

RZ Continues the program after a breakpoint.

COMMENTS

Don't try to specify a trigger event before **RZ**-- it will not work.

S no parameters

Starts the bus state analyzer. Resets the target system if automatic **RESET** is enabled.

USAGE

You do not need to start the analyzer on the same line as the command that sets up the trigger event specification, though that is the usual practice. **S** is a separate command that gets the analyzer going with whatever spec you created already in place.

You can use **TSTAT** to see what the trigger has been set up to (Trigger **STATus**).

EXAMPLES

S Starts the analyzer, with whatever trigger was last defined.

NORMT RESET 123 ADR S
clears out the trigger spec, turns on auto-reset, and then sets it to address = 123 before starting the analyzer (and restarting the target board).

S+ no parameters

Identical to **S**, except that it increases the delay cycle count by A6 counts.

USAGE

Handiest when you find that your current trace just starts getting interesting at the end. **S+** by itself will trigger on the same event, but with a new trace window that starts 3 cycles before the end of the previous one.

You should use this when your trigger spec is an event that gets regularly repeated during the program, or with **RESET** enabled. All **S+** does is change the value of **DCYCLES** and then start the analyzer again.

So if your trigger spec only happens once in the program, and **RESET** is disabled, then the UniLab will be searching a program in progress for an event that has already occurred.

EXAMPLE

S+ restarts the analyzer with an increased delay setting.

SAMP

no parameters

Samples the 48 input lines several times a second, and displays them until any key is pressed.

USAGE

A good way to get a vague idea of what is going on. It will be clear to you that the program has been stuck in an infinite loop, or that it has gone far astray. But you will not be able to tell much, as you only see one cycle out of every several thousand.

DISASSEMBLY

You will probably want to turn off the disassembler, with the **Mode Panel (F8)** or **DASM'**. When the disassembler is enabled the isolated cycles will probably be disassembled incorrectly.

EXAMPLE

SAMP

This command never used in combination with anything else.

COMMENTS

Useful when you are trying to connect analyzer inputs to something and you want to continuously monitor their state. Similar to 1 **SR** but it runs faster. Gives more detail on program execution than **ADR?**. Don't forget to start from scratch on trigger specs after using **SAMP**, because it defines its own trigger.

It also turns off the **RESET**.

SAVE-SYS

SAVE-SYS <file name>

Saves the entire UniLab system program in its present state as a named DOS file. Prompts you for file name if you do not include it on command line.

USAGE

To save a version of the system with new macros, or with default drives changed. Or, just to save the current emulator enable values, the current trace, and the trigger definition.

Warning-- does not save the symbol table. Do that with **SYMSAVE** command.

EXAMPLE

SAVE-SYS B:NEWUL

Saves the system to a new file on the B: drive.

COMMENTS

Note that the target program, which is in the UniLab itself, is not saved by this command. Use **BINSAVE**.

This command automatically makes the "file extension" **.COM**.

Since the entire program image is saved including any unintentional damage to the program, always keep backup copies.

SC <count> SC <file name>

Starts the analyzer and waits the specified maximum number of milliseconds for trigger. When trigger occurs, the trace gets compared to a previously saved trace.

USAGE

Very useful when writing test programs that compare the trace to a known good trace that you have stored away. Save traces with the **TSAVE** command. If a trace does not match, the host computer beeps and displays both a section of the previous trace and the first bad step of the new trace.

HARDWARE CHECKOUT

Probably most useful for hardware checkout. To get a vague idea of the capabilities, save a trace right now (**TSAVE test**). Then pull the RAM off your target board and execute the command below. Don't change your trigger spec between saving the good trace and getting the new one. See Appendix F for examples.

EXAMPLE

test 400 SC

Starts the analyzer board with a 400H ms trigger time limit (1 sec.) and compares the trace to the one saved in file "test."

COMMENTS

If the time limit passes with no trigger, the host displays a "NO TRIGGER" message and beeps.

SET <n> SET

Sets bit n of the stimulus generator output. The number, of course, must be between 0 and 7.

USAGE

Simulates a peripheral input going from voltage low to voltage high. The stimulus generator allows you to test how your system responds to digital signals on certain lines.

EXAMPLES

```
7 SET
    sets stimulus output 7.

1 SET 1 RES
    pulses output S1.
```

COMMENTS

Used to set individual bits of the 8 stimulus outputs. See also **RES** and **STIMULUS**.

SET-COLOR no parameters

Change the display colors for a color monitor.

USAGE

After you have issued the command **COLOR** to inform the UniLab software that you have a color monitor, you can change the display colors with this command.

You use the cursor keys to choose different colors, and see them displayed as you choose. Press the **END** key on the numeric key pad when you have completed your choices. You will need to save the system with **SAVE-SYS** if you want the colors to be permanent.

-- The Commands --

SHIFT-FKEY <# of key> SHIFT-FKEY <command>

Assigns a command to a function key hit while the SHIFT key is held down.

USAGE

Reassign the function keys on PCs and PC look-alikes. Use **SHIFT-FKEY?** (or SHIFT-F1) to find the current assignments.

The function keys allow you to execute any command or string of commands with a single keystroke. The initial assignments represent our best guess at what you will need. But you might want to change them.

To make your reassignments permanent, use **SAVE-SYS**.

EXAMPLE

6 SHIFT-FKEY TSTAT
assigns TSTAT to SHIFT-F6

COMMENTS

To execute a string of commands, define a macro first (using :) and then assign the macro to the function key.

See also **FKEY**, **CTRL-FKEY**, and **ALT-FKEY**.

SHIFT-FKEY? no parameters SHIFT-F1

Displays the current assignments of the SHIFTeD function keys.

USAGE

Whenever you want to be reminded what command will be executed when you press a function key while holding down the shift key.

See **SHIFT-FKEY** to reassign the keys.

SHOWC no parameters F8

Shows the control lines on the trace display (the default condition).

USAGE

Turn on display of the control lines, C7 through C4, as well as the high four bits of the address bus, A19 through A16.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

SHOWC' no parameters F8

Turns off display of the control lines on the trace display.

USAGE

Turn off display of the control lines, C7 through C4, as well as the high four bits of the address bus, A19 through A16.

Though the UniLab must always monitor these wires, and sometimes they give you vital information (such as that you have the wires hooked up wrong), usually you don't need to see them.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

SHOWM no parameters F8

Shows the miscellaneous lines and the HDATA lines on the trace display (the default condition).

USAGE

Turn back on display of the miscellaneous lines and the high data lines (on 8 bit processors).

You will want to see these lines when you have them hooked up to your board. Otherwise, you can ignore them.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

SHOWM' no parameters F8

Hides the miscellaneous lines and the HDATA lines on the trace display (the default condition).

USAGE

Turn off display of the miscellaneous lines and the high data lines (on 8-bit processors).

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

SMBP <addr> <breakpoint #> SMBP

Sets one of the 8 multiple breakpoints at the given address.

USAGE

Allows setting of up to 8 breakpoints, in addition to the unnumbered breakpoint that is set by **RB** or **GB**. The status of all 8 breakpoints gets displayed each time you set or clear one.

You must already have debug control before you issue this command.

To use multiple breakpoints, set all but one of your breakpoints with this command, and then use **RB** or **GB** to get the target program going again.

EXAMPLES

123 4 SMBP
sets a breakpoint #4 at address 123.

250 RB
sets a breakpoint at 250 and starts the target program going again.

COMMENTS

See also **N**, **CLRMBP**, **RMBP**.

Before using multiple breakpoints, you should examine the possibility of using the more powerful capabilities of the analyzer to do the same thing.

SOURCE SOURCE <file name>

Enables the display of source code interleaved with disassembly. You must supply the name of your source file.

USAGE

Allows you to have your high-level language source file displayed in the trace. After you issue this command, each line of your source code will be displayed just before the instructions that were generated by that line of the source code.

DISABLE

You turn this display option off again with **SOURCE'**.

EXAMPLE

SOURCE C:\ASM\TEST.C
loads in the source file TEST.C and then uses the symbol table to correlate lines of the source file to instructions in the binary file.

SOURCE' no parameters

Turns off the display of source code in trace. See **SOURCE**.

SPLIT no parameters F2

Toggles split screen mode on and off.

USAGE

Gives you the ability to compare traces, or parts of the same trace. You can also compare a trace to the assembly code (DN), or to your source text file (TEXTFILE).

WHAT WINDOWS ARE FOR

The right quadrants are reserved for the output of DN, and for the pop-up panels (MODE). TEXTFILE only works in the top window. Help screens are always shown in the top window.

MOVING AROUND

The END key moves you from one window to the other.

HISTORY

The history mechanism, which saves a record of what has happened during your session with the UniLab, only records information off of the bottom screen.

EXAMPLES

SPLIT

This command never used in combination with anything else.

-- The Commands --

SR <n> SR

ReStarts the analyzer Repeatedly. Displays n lines each time trigger occurs.

USAGE

Very useful for logging things repeatedly. You should first set up the trigger and starting point of the display with **S** and **TN**.

STOPPING

You start the infinite loop by entering **SR**. You break out by hitting any key.

HARD COPY

Use the **Mode Panel (F8)** or **PRINT** to log your output to the printer. The **Mode panel** also contains a feature that allows you to log to a file. See **TOFILE**.

RESETTING OR INTERRUPTING THE TARGET

If you use **RESET**, then the target system will be reset each time the analyzer starts.

WHEN TO USE SOMETHING ELSE

If the events you want to see occur more often than once per second and you want to see them in sequence, you can use **XAFTER** along with **A9 SR** to log bursts of the events in filtered format.

EXAMPLES

20 SR

Repeatedly displays twenty lines of trace buffer, starting the analyzer again after each display.

SST <trigger spec> SST

Starts the analyzer in the standalone mode.

USAGE

Set the analyzer looking for a bug that you think will take a while to find. After you issue this command, you can disconnect the UniLab from your host, or you can keep it plugged in but exit from the UniLab program (BYE).

Either way, the LED on the UniLab goes out when it finds the trigger. You then plug in the UniLab again, call up the UniLab program, and enter TS to display the trace.

EXAMPLE

NORMB 1200 TO 1300 ADR WRITE 3F TO FF DATA SST
Searches for this trigger in standalone mode.

COMMENTS

Handy when you want to search for an obscure bug without tying up the host computer.

SSTEP no parameters F6

Only on processors that support NMI (see appendix H), this command allows you to follow jumps, calls, and branches.

USAGE

After you have established debug control with
RESET <addr> RB

or

NMI

you can single-step through your code using a combination of **N** and **SSTEP**.

Both instructions can be used only when you are stopped at a breakpoint. **SSTEP** is appropriate when the instruction pointed to by the program counter is a jump, call, return, or branch. **N** is the correct command at all other times.

Use appendix H to check whether or not your processor supports NMI.

EXAMPLE

SSTEP

This command never used in combination with anything else.

STANDALONE STANDALONE <prom programming command>

Selects the standalone mode for the EPROM programming command that follows.

USAGE

Allows you to use the host computer for something else while the UniLab programs an EPROM. Especially handy when programming large EPROMs.

You can type in **STANDALONE** and press return, then use the PROM programming menu to program the EPROM.

When the LED next to the PROM programming socket goes out, the command has been completed. You can then enter **PROMMSG** to get the completion status message. The UniLab must remain connected to the host computer, or you will not be able to get the message.

EXAMPLES

STANDALONE

use this command and then make use of the convenient PROM programming menu to burn an EPROM in standalone mode.

STANDALONE 0 TO 1FFF P2764

you can also use **STANDALONE** along with a PROM burning command, if you know the commands.

STARTUP

no parameters

F9

Restarts the target system and gives a trace of the first 170 cycles of target system operation.

USAGE

Very useful mode at the first stages of system checkout. Allows you to check out the first few instructions, make certain that they execute properly.

The RES- wire from the analyzer cable must be properly connected to the target system, or the UniLab will not be able to reset the target processor. See the INSTALLATION chapter of the manual.

The very first cycle (cycle 0) is particularly important because if correct data is not fetched (often due to the address not being properly EMENABLEd), then the program will immediately "blow up."

MULTIPLE RESET

Some systems with simple R-C reset circuits (no hysteresis) will appear to reset intermittently many times before they finally settle down to stable operation. This is a nuisance if you want to look at a trace early in the program, but you will be able to see the program when it does finally settle down.

If your system does this, you might want to consider putting a logic element-- such as two Schmitt triggers in a row (part number LS14)-- into your reset circuit. That way your system will always get a good strong reset signal.

EXAMPLES

STARTUP

This command never used in combination with anything else.

COMMENTS

This is a target specific macro that usually looks for the reset vector address on the bus. If that address does not show up, system will wait forever. Or if a HALT instruction is fetched, will give a "NO ANALYZER CLOCK" message. See TROUBLE SHOOTING chapter.

STIMULUS

<byte> STIMULUS

Changes the 8 stimulus outputs (S0-S7) to correspond to the specified byte. Also pulses the ST- output.

EXAMPLE

10 STIMULUS

makes all stimulus outputs zero, except S4

COMMENTS

Useful for changing all stimulus outputs at once. Use **SET** or **RES** to set and reset individual signals. The stimulus outputs originate in the PROM socket on the front of the UniLab and are normally connected by the stimulus cable provided with your system. The stimulus signals are usually used to provide test inputs for the target system.

SYMB no parameters F8

Enables the symbol translation feature.

USAGE

Turns symbol translation back on, after it has been disabled with **SYMB'**. Symbols make the trace more readable, by allowing you to replace data and addresses with symbolic names.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

Symbols are entered by using **IS** or **SYMFILE**, either of which will turn on symbol translation.

SYMB' no parameters F8

Disables the symbol translation feature.

USAGE

To turn symbol translation off without clearing out the symbol table. See **CLRSYM** if you want to clear out the table.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

SYMFILE

SYMFILE <file name>

Loads a symbol table file produced by a cross assembler or by the UniLab program. Prompts for a file name if you don't include it on command line.

USAGE

Capable of loading symbol tables in almost any format. The first time you use it, **SYMFILE** presents you with a menu of predefined formats. You can choose one of those, and then save the system with **SAVE-SYS** to make that the default format.

You can change the default format with **SYMTYPE**.

After you have defined a format, **SYMFILE** will prompt you for a file name and load a symbol file into memory.

Formats not on the menu can be defined using **SYMFIX** for fixed length files. Variable length files only come in two formats: name and then value, or value and then name.

The AVOCET format on the menu is for symbol files that are name, then value.

The MANX format on the menu is for symbol files that are value, then name.

EXAMPLES

SYMFILE C:\ASM\OUT.SYM

Loads into the UniLab a symbol file created by an assembler.

SYMFIL+ SYMFIL+ <file name>

Appends the contents of a symbol file to the symbol table.

USAGE

Provides a way of adding to a symbol table that already exists. **SYMFIL**, on the other hand, automatically clears the existing symbol table.

SYMFIL+ allows you to combine several symbol tables.

See also **CLRSYM**.

EXAMPLES

SYMFIL+ A:EXTRA.SYM

Adds to the symbol table the symbols stored in a file on the A drive.

SYMFIX <a> <c> <d> <e> <f> SYMFIX

Defines symbol file parameters for formats that use fixed length records.

USAGE

Use this word to define your own **SYMFILE** format for fixed length records, if none of the predefined formats available on the **SYMFILE** menu suit your purposes. There are only two types of variable length record formats (value then name or name then value) and both appear in the menu.

The definitions of the 6 parameters:

- a = offset from start of record to start of name field.
- b = 1 if address is 4 ASCII digits or 0 if 16-bit binary.
- c = address field offset from start of record.
- d = 1 if binary address has most significant byte first.
- e = pad characters used to fill between symbols.
- f = record length.

EXAMPLES

0 0 B 1 0 E SYMFIX
defines the format for 2500AD abbreviated symbol table files. These tables follow the format:
ten bytes for the symbol name,
two bytes for the symbol value,
two pad bytes.

SYMLOAD

SYMLOAD <file name>

Loads a UniLab format symbol table file from the disk. Prompts for you for file name if you don't include it on command line.

USAGE

Loads up a symbol table that was saved with **SYMSAVE**.

These files are variable length, allowing symbols up to 255 characters long.

Warning: not compatible with symbol tables saved with pre-version 3.0 **SYMSAVE**.

EXAMPLE

SYMLOAD B:oldsyms

Loads into the UniLab a symbol table file from the B drive.

SYMSAVE

SYMSAVE <file name>

Saves the symbol table as a named DOS file. Prompts for file name.

USAGE

This command saves only the symbol table, which you will be able to load in later with **SYMLOAD**.

Use **SAVE-SYS** to save the entire system.

EXAMPLE

SYMSAVE july3.sym

Saves the current symbol table to a file called july3.sym.

SYMTYPE

no parameters

Re-defines the file format assumed by the **SYMFILE** command.

USAGE

Allows you to pick a different predefined file format, after you have chosen one with **SYMFILE**.

The first time you use the **SYMFILE** command you are presented with a menu selection of default formats. Once you have saved the default format, **SYMFILE** simply executes immediately, using the selected format. The **SYMTYPE** command allows you to get that menu again so that you can change your selection.

See also **SYMFIX**.

EXAMPLE

SYMTYPE

This command never used in combination with anything else.

-- The Commands --

T no parameters

Displays the trace from its current starting point until any key is pressed.

EXAMPLE

T displays the trace.

COMMENTS

The current starting point for the trace display is defined by the most recent TN command. (STARTUP usually sets it to -4.)

If the starting cycle # is not actually in the trace buffer, the trace is started 4 lines from the closest cycle number which is in the trace buffer.

TCOMP <n> TCOMP <file name>

Compares the present trace buffer to a previously stored trace in the named file. Compares the last <n> cycles. Aborts and indicates error if any bit fails to compare.

USAGE

Very useful for writing automatic system test programs. Use the value **AA** to compare the entire trace.

Use **TSAVE** to save the trace of a good system. You can then use that saved trace to test other systems.

If **TCOMP** finds a difference between the current trace and the one in the file, it will display 9 lines of the stored trace and the first bad line in the trace of the system under test.

You can use **TMASK** to tell **TCOMP** to ignore one or more of the columns in the trace display. See **TMASK** for details.

You can also use **SC** to compare traces.

EXAMPLE

AA march.2 TCOMP
compares the entire trace to the one stored as file "march.2."

COMMENTS

If you want to compare only part of the trace, use a smaller number. **TCOMP** will then skip over the first part of the file. This is useful for skipping over the already known discrepancies between two traces.

If **TCOMP** behaves in a confusing manner, try using it with the disassembler disabled (**DASM'** or use the mode panel, **F8**).

TD no parameters

Stops the analyzer and displays the current contents of the trace buffer.

USAGE

To see what is going on, when trigger has not occurred, or when you are producing a filtered trace that you do not think will fill up the trace buffer. Normally the trace is automatically uploaded to the host when trigger occurs.

TD skips over the first cycle in the buffer, and any other empty space (all 1's) at the top of the buffer.

EXAMPLE

TD This command never used in combination with anything else.

COMMENTS

Since the buffer is filled with 1's before the analyzer is started, a partially filled filtered trace buffer will have good data only near the end. **TD** automatically skips over the empty space.

TEXTFILE

TEXTFILE <filename>

Allows you to look over a text file from within the UniLab program.

USAGE

TEXTFILE only works from the upper window. It will take a few seconds to analyze the file, and then will show you the first window full of text.

This feature is useful for looking at your source code while you debug it-- this could replace hard copy listings.

MOVING AROUND THE FILE

Use the PgDn key or the Down Arrow to see more of the text. The PgUp key scrolls the screen back, the Up Arrow moves you up one line.

The HOME key takes you back up to the top. The END key just toggles you to the lower window.

WATCH OUT

You can't alter the file in any way-- only look it over.

EXAMPLE

TEXTFILE \memo\project1

Opens the DOS file project1, in a directory called memo.

THIST

no parameters

Time **HIST**ogram invokes the optional histogram generator that allows you to display how often the elapsed time between two addresses falls into each of up to 15 user-specified time periods. See **AHIST**.

USAGE

Allows you to examine the performance of your software. You can find out how the elapsed time between any two addresses changes, as different conditional jumps or branches are taken.

To get interesting and useful results, you will probably want to measure the time between two addresses in your main loop.

Press **F10** to exit from this command.

You must (only once) issue the command **DOHIST** to enable this optional feature. **DOHIST** performs a **SAVE-SYS**, and then causes an exit to DOS. The next time you call up the software, both **AHIST** and **THIST** will be enabled.

MENU DRIVEN

You produce a histogram by first specifying the upper and lower limits of each time "bin" that you want displayed, then starting the display.

When you give the command **THIST**, you get the histogram screen with the cursor positioned at the first bin. You can then start typing in the lower and upper limits of each bin. Use return, tab, or an arrow key after you enter each number, to move to the next entry field.

Press function key 1 (**F1**) to start displaying the histogram.

SAVE TO A FILE

You can save the setup of a histogram as a file with the **HSAVE** <file>. Issue this command after you exit from the histogram.

You load the histogram back in with **HLOAD** <file>. Issue this command before invoking the histogram.

TMASK <byte value> **TMASK**

Set up a mask which tells **TCOMP** which columns to compare.

USAGE

The lower six bits of the byte value tell **TCOMP** which groupings of the trace display to use when comparing traces. The default is 3F (00111111 binary) which tells **TCOMP** to check all columns.

Used when comparing traces to filter out erroneous error messages-- due, for example, to different wiring of the **MISC** lines.

MASK VALUES

Each of the six bits corresponds to one of the groupings. If the bit is one, then **TCOMP** will include that grouping:

BINARY	GROUPING	HEXADECIMAL
0000 0001	LADR	1
0000 0010	HADR	2
0000 0100	CONT	4
0000 1000	DATA	8
0001 0000	HDATA	10
0010 0000	MISC	20

TN <n> TN

Displays the trace buffer, starting at cycle n. Sets the starting point for future trace displays.

USAGE

For random access to the trace buffer, when you also want to reset the starting point used by **T**. To access the buffer without changing the default value of the point where the display starts, use **TNT**.

EXAMPLE

12 TN

Displays the trace, starting 12 cycles after the trigger. The rest of the traces this session will also be initially displayed starting 12 cycles after the trigger.

COMMENTS

You will usually want to use **TNT**. Use **TN** when you think that you will want to display from the same point on future trace displays.

TNT <n> TNT

Displays the trace buffer, starting at cycle n.

USAGE

Allows you to immediately look at any point in the trace buffer. **TN** does the same thing, but also changes the default trace starting point used by **T**. The default trace starting point is set to -5, until you change it.

EXAMPLE

-7 TN

displays the trace starting 7 cycles before the trigger.

TO <number> TO <number> <command>

Sets a flag that indicates that a range of numbers is being entered.

USAGE

Used with all of the trigger event description commands to define a trigger on a range of numbers. See **ADR**, **CONT**, **DATA**, **HADR**, **HDATA**, **LADR**, and **MISC**.

EXAMPLE

12 TO 34 DATA

Tells the analyzer to look for any data on the range 12 to 34 on the data inputs.

COMMENTS

In the example above omitting the TO would result in a trigger spec that would accept only data = 34.

TOFILE TOFILE <filename> F8

Use to start sending screen output to a DOS textfile as well as to screen.

USAGE

Use for toggling on the logging of information to a file. You can include that command on the DOS command line as a "command tail." For example:

C> ULZ80 TOFILE A:JUNE7.LOG

The usual DOS rules for naming files apply.

You will be prompted for the file name if you do not include it.

Turn off logging to the file with **TOFILE'**.

You can use the **MODE** panel (function key 8) to toggle logging to a file on and off, but you have to use the command to open the file in the first place.

COMMENTS

Files produced in this way can then be edited with a word processor, or shown on the screen using the DOS command: **TYPE** file name.

TOFILE' no parameters F8

Use to stop sending screen output to DOS textfile as well as to screen.

USAGE

Use for toggling off the logging of information to a file.

Normally you will use the **MODE** panel (function key 8) when you want to change this feature.

TOP/BOT no parameters END key

Moves you from top window to bottom, or from bottom window to top.

USAGE

You will usually want to just use the END key, which is the number one key of the numeric key pad.

Only active when the screen has been split with **SPLIT** (function key 2). See that word for details about windows.

EXAMPLE

TOP/BOT

This command never used in combination with anything else.

TRIG TRIG <event description>

The event description that follows will be a trigger event.

USAGE

As opposed to **Q1**, **Q2**, and **Q3**, which tell the analyzer that the following description is a qualifying event. Useful if you want to alter the trigger event without altering the qualifiers.

EXAMPLE

TRIG 123 ADR
searches for 123 on the address lines.

COMMENTS

Used to select the TRIG truth table context again after **AFTER**, **Q1**, **Q2**, or **Q3** has caused another truth table to be selected. Useful if you want to change your mind about the trigger step after you have just defined a qualifier. The 4 truth tables are **Q3**, **Q2**, **Q1**, and **TRIG**.

TS no parameters

Displays trace after standalone mode trigger.

USAGE

To retrieve the trace from the UniLab's trace buffer, after you start the analyzer in standalone mode with **SST**.

When you use **SST** to start the analyzer, you can disconnect your host computer from the UniLab and run other programs on the computer. When the analyzer sees the trigger, the light next to the analyzer goes out. You can retrieve the trace at anytime after that.

To retrieve the trace you must start up the UniLab program while the UniLab is disconnected from the host. Use **CONTROL - BREAK** to break out of the **Initializing UniLab...** message. Then reconnect the UniLab and issue the **TS** command.

EXAMPLE

TS
This command never used in combination with anything else.

COMMENTS

TS begins by sending a "wake-up" code to the UniLab. Since this does not fit into the normal UniLab communications protocol, don't enter **TS** unless you have previously entered **SST**. If you do, the system will hang.

-- The Commands --

TSAVE TSAVE <filename>

Saves the current trace buffer as a file.

USAGE

A good way to save information about a trace for later review with **TSHOW** or for automatic comparison to another trace with **TCOMP** or **SC**.

EXAMPLE

TSAVE good.trc
saves current trace as a file called good.trc.

TSHOW TSHOW <file name>

Displays a previously saved trace.

USAGE

A useful way to examine traces saved while in the field, or by an automatic testing program. **TSAVE** saves the trace in the first place.

TCOMP will compare the present trace to the numbered trace, and let you know if they differ. That will probably, most of the time, serve your purposes better than looking over a trace.

EXAMPLES

TSHOW good.trc

Displays the trace saved by **TSAVE** into a file called `good.trc`.

COMMENTS

If you are tracking down a problem you can save interesting traces as you go so that you can look at them again later or even print them out (by using control-P to turn on the printer).

Note that after you use **TSHOW** the trace image in the host contains the recalled trace, so you can use **T**, **TN**, or **TNT** to view it from various points.

When you want to load the UniLab's trace buffer back into the host, enter **TD**. Since **TSHOW** changes the setting of **DCYCLES**, the cycle numbers will be incorrect unless the changed delay setting is the same as the previous one.

TSTAT no parameters F7

Displays the complete status of the current trigger specification including qualifiers, delay and pass counts filtering, and auto reset.

USAGE

A good way to determine what the current settings are. Also a good way to check on how the UniLab interprets your trigger specifications.

EXAMPLE

TSTAT

This command never used in combination with anything else.

WORDS WORDS <command>

Displays an alphabetical listing of the UniLab's commands, starting with the command or characters you include on the command line.

USAGE

To remind you of the names of some UniLab commands. Hit any key to stop the listing.

EXAMPLE

WORDS INIT

shows a list of commands, starting with INIT.

WSIZE no parameters SHIFT-F8

Allows you to redefine the size of the windows.

USAGE

Once you enter this command, only the cursor keys are active. Use the "END" key (numeric pad key 1) to exit.

Use whenever you want to set the window size to something other than the standard setup.

EXAMPLE

WSIZE

This command never used in combination with anything else.

Chapter Eight: Target Notes

Introduction

This chapter consists of separate writeups for each of the processor specific software packages. The chapter serves as both:

- 1) a guide to connecting your cable properly,
and
- 2) an overview of the special features of each disassembler/debugger package and the quirks of the microprocessor chips.

Be sure to read the **General Information** on pages 2 through 4 for important information common to all the processor-specific software packages.

Consult Appendix C to double check the wiring of your connection between the UniLab and your target board.

Each software package is shipped with a separate Disassembler/Debugger Note. That document includes the printout of a full demonstration of the analyzer and debugger for each processor. It also includes any special information on the processor specific on-line assembler (**ASM**).

Contents

General Information	8-2
1802/4/5/6 (disassembler only).....(DIS-18)...	8-5
6301/3	(DDB-63)...8-7
6500 series where the SYNC output exists..(DDB-65)...	8-10
6500 series piggyback divices w/o SYNC....(DDB-65P)...	8-14
6800/2/8 with external memory at page 0 ..(DDB-68)...	8-18
6801/3	(DDB-681)..8-21
6802 without external RAM at page 0	(DDB-682)..8-24
6805	(DDB-685)..8-25
6809	(DDB-689)..8-30
68000	(DDB-68K)..8-32
68008	(DDB-688)..8-36
68HC11	(DDB-611)..8-38
8048/35/39/40/49/50.....	(DDB-48)...8-40
8051/31/32/52 & 8051P	(DDB-51) & (DDB-51P)..8-45
8080/85	(DDB-85)...8-50
8086/186/286 & 8088/188.....	(DDB-86) & (DDB-88)...8-53
8094/5/6/7	(DDB-96)...8-61
SUPER 8.....	(DDB-S8)...8-65
Z8	(DDB-Z8)...8-68
Z80 and NSC-800 and HD64180	(DDB-Z80)..8-72
Z8000	(DDB-Z8K)..8-76

GENERAL INFORMATION

All debugger packages have certain common requirements and characteristics. Read this section to get a quick briefing.

Watch out

All Orion debuggers make use of some of your processors resources. All of them require a several-byte "reserved area" in emulation ROM that you cannot put code into, and a larger "overlay area" that you can use.

Some debuggers also make use of a trap vector or a single internal register of your processor.

And all of them use your stack.

Be certain to check the section on your processor for the **Reserved Area** or **Reserved Resources**. You can also get information on-line by pressing **CTRL-F3** (hold down the control key and tap function key three). This **Help Screen** also appears in the section on your processor. And Appendix H includes this information for all processors.

How the debuggers work

All debuggers work by downloading Orion software routines to your emulation ROM (in the overlay area) and using your processor to execute routines that display target registers, alter target RAM, etc.

While you can put code into the overlay area, most of the debugger routines will not work on that area of memory.

Required code in user program

All the routines that are downloaded to your processor require a working stack. Your code must initialize the stack pointer, or the Orion routines will not work. You cannot set a breakpoint in your code until the stack pointer is initialized.

Some processors automatically initialize their stack pointer to use internal RAM.

A few debuggers, such as that for the 8088 and 8086 processors, require that your code initialize other resources, such as interrupt vectors. Look for the heading **Required Code in User Program**, in the following pages.

Some debuggers require you to issue a UniLab command that specifies the areas of RAM or ROM you want the debugger to use. Look for the heading **Commands Required**.

Patch words

Many packages support several slightly different processors. If the differences are big enough, you have to enter a patch word to configure the software package for your processor. The patch words are always mentioned on the first page of the writeup on your debugger; as well as in Appendix J of this manual.

Once you enter the patch word, your software is ready for use.

Either use **SAVE-SYS** to save the configured software, or enter the patch word at the start of each UniLab session.

Some packages now have a "Patch Menu." This menu is invoked with the UniLab command **PATCH**, and gives you a choice of all the processors supported by the package.

Connections

Each section of this chapter includes a diagram that shows you how to connect your UniLab to your processor. You can get a diagram on-line by typing in the command **PINOUT**. Some software packages support more than one processor, and require different cable connections for different processors.

You can also consult Appendix C to double-check your wiring.

Access to RAM, to internal registers, etc.

You cannot look at or alter RAM or internal registers until you have first established debug control. You establish debug control by setting a breakpoint (**RESET <addr> RB**) or by asserting a non-maskable interrupt while your target program is running (**NMI**). Not all processors support **NMI**-- check appendix H, or check whether the cable diagram calls for the NMI wire to be connected to a processor pin.

For more information on debug control, check the entries for **NMI** and **RB** in the command reference chapter.

Trace and breakpoint display

You will probably want to compare the trace you get with the sample program to the printout that appears at the end of the section on your processor. These sample sessions also include a sample of the breakpoint display.

Enter **LTARG** to load the sample program, then follow the steps shown in the printout. A more extensive demo session for your software appears in the separate Disassembler/Debugger writeup that is shipped with the Disassembler/Debugger diskette.

ROM Cables and Address Eleven (A11)

Many processors show a connection to A11. This is necessary only with the 24-pin ROM cable (which is the usual ROM cable).

180X DISASSEMBLER (DIS-18)

This version of the UniLab software disassembles 1802, 1804, 1805, and 1806 instruction sets. It is the only Orion package that does not yet include a debugger.

However, remember that you can use the UniLab's NMI output or generic breakpoint capability to set multiple breakpoints. If you write your own breakpoint routine to put the desired registers or memory location contents on the bus, you will have a simple debug capability.

The 1802 family requires the G analyzer cable.

g Cable		1802		g Cable
	CLOCK	1	40	Udd
	<u>WAIT</u>	2	39	<u>XTAL</u>
RES	CLEAR	3	38	<u>DMA IN</u>
	Q	4	37	<u>DMA OUT</u>
C7	SC1	5	36	<u>INTERRUPT</u> NMI
C6	SC0	6	35	MWR WR-
RD-	<u>MRD</u>	7	34	TPA
	BUS 7	8	33	TP8 ALE
	BUS 6	9	32	MA7
	BUS 5	10	31	MA6
	BUS 4	11	30	MA5
	BUS 3	12	29	MA4
	BUS 2	13	28	MA3
	BUS 1	14	27	MA2
	BUS 0	15	26	MA1
	Vcc	16	25	MA0
	N2	17	24	<u>EF1</u>
	N1	18	23	<u>EF2</u>
	NO	19	22	<u>EF3</u>
GND	Vss	20	21	<u>EF4</u>

Trace display

(Note user words entered at terminal are underlined.)

LTARG (load sample program, set default area of emulation memory)

Emulator Memory Enable Status:

F =EMSEG

0 TO 7FF EMENABLE

ok

STARTUP resetting (issue reset to target, capture first cycles of bus activity)

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
0	3F	0000	71	DIS	11111111	11111111
1	7F	0001	00	read	11111111	11111111
2	3F	0002	F800	LDI 0	11111111	11111111
4	3F	0004	A1	PLO R1	11111111	11111111
5	3F	0005	F808	LDI 8	11111111	11111111
7	3F	0007	B1	PHI R1	11111111	11111111
8	3F	0008	51	STR R1	11111111	11111111
9	5F	0800	08	write	11111111	11111111
A	3F	0009	01	LDN R1	11111111	11111111
B	7F	0800	08	read	11111111	11111111
C	3F	000A	11	INC R1	11111111	11111111
D	3F	000B	F6	SHR	11111111	11111111
E	3F	000C	3008	BR 8	11111111	11111111
10	3F	0008	51	STR R1	11111111	11111111
11	5F	0801	04	write	11111111	11111111
12	3F	0009	01	LDN R1	11111111	11111111
13	7F	0801	04	read	11111111	11111111
14	3F	000A	11	INC R1	11111111	11111111
15	3F	000B	F6	SHR	11111111	11111111
16	3F	000C	3008	BR 8	11111111	11111111
18	3F	0008	51	STR R1	11111111	11111111

Pg Dn Home (top) n TN (from step n) T (from n=-5) ok

(NOTE: no breakpoint display, because this package does not include a debugger).

630X DISASSEMBLER/DEBUGGER (DDB-63)

This version supports the 6303R, 6303X, and 6301 processors. The 6301 is a piggyback chip that was discontinued in 1986.

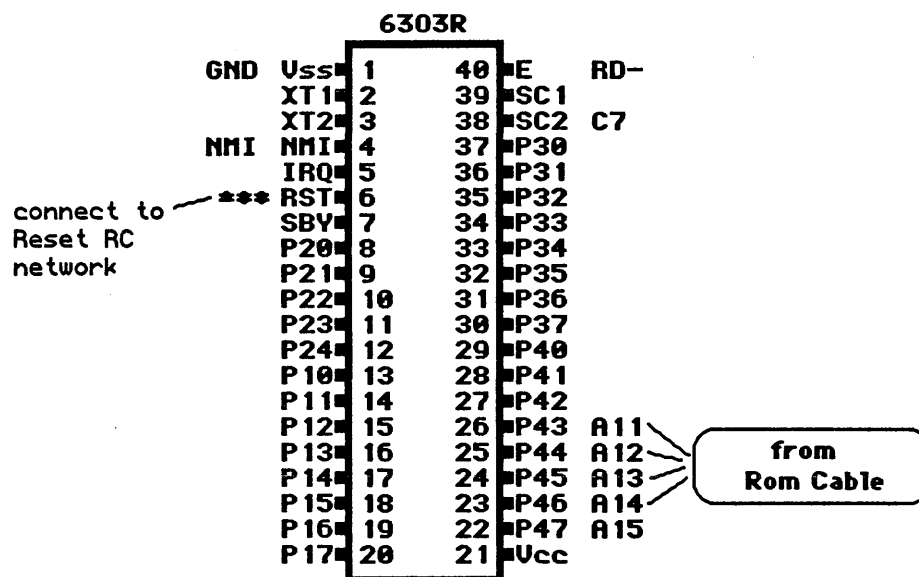
This package is similar to the Orion software for the 6800. Please refer to the 6800 writeup for important details.

The 6303R and 6303X processors require the B analyzer cable.

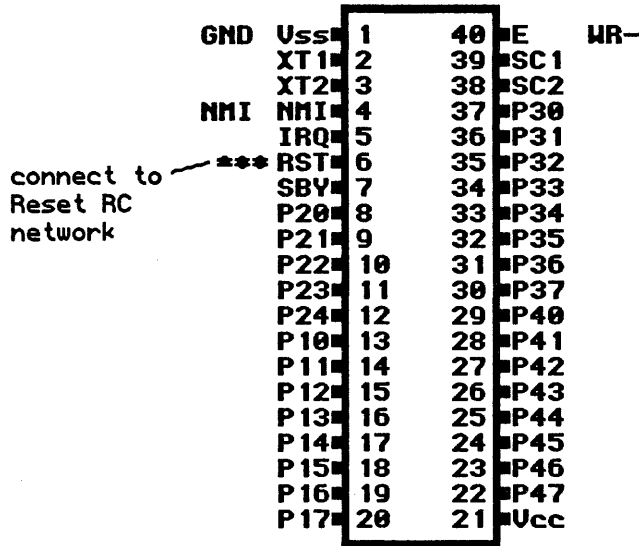
The piggyback 6301 requires the N cable.

Patch word

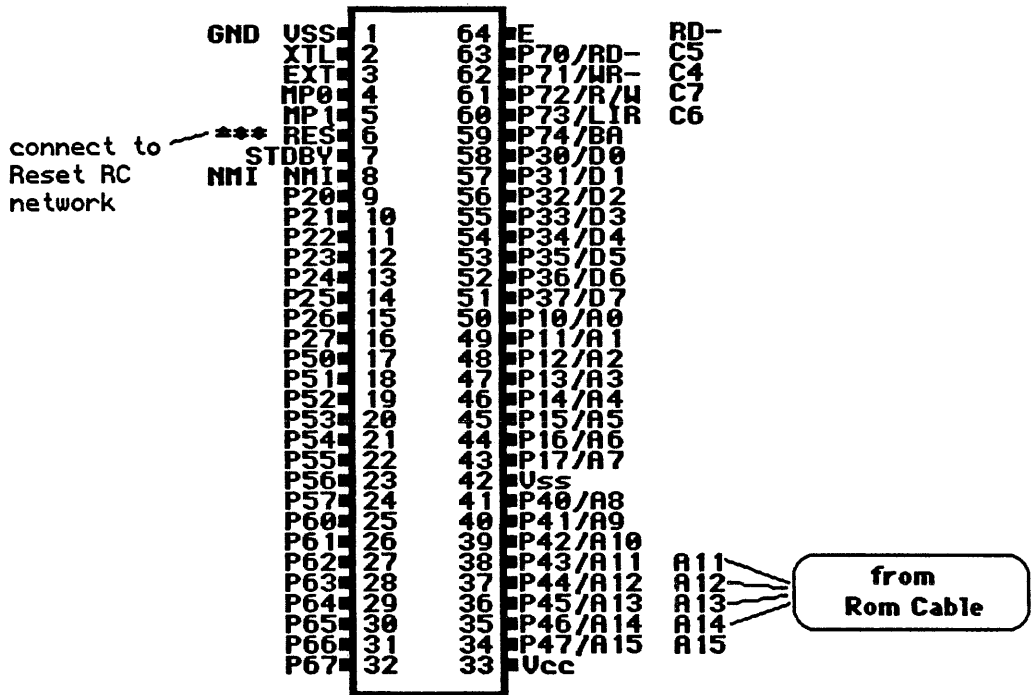
For operation with the piggyback 6301, you enter **PBACK** to patch the program, then **SAVE-SYS** to save the newly configured software.



6301 Piggyback



6303X



Help Screen

Press CTRL-F3 to get the help screen that includes this target-specific information:

Register change: n =A n =B n =CC nn =X

See also HD . \$FFB9,A reserved. Overlay area = \$FFBC to reset

Trace and Breakpoint Display

LTARG (Load built-in sample program, set typical emulation memory area)

Emulator Memory Enable Status:

F =EMSEG

F800 TO FFFF EMENABLE

ok

STARTUP resetting (Reset target system, and start analyzer,)

(from top of buffer) (triggering on reset vector...)

cy#	CONT	ADR	DATA	HDATA	MISC
-1	7F	FFFE	FF (reset vector)	11111111	11111111 (note)
0	7F	FFFF	00interrupted	11111111	11111111 (vectors)
1	7F	LTARG.PGM	FF00 8E00FF LDS #FF	11111111	11111111 (and data)
4	7F	PGM.LOOP	FF03 8612 LDAA #12	11111111	11111111
6	7F		FF05 C634 LDAB #34	11111111	11111111
8	7F		FF07 CE5678 LDX #X.INIT	11111111	11111111
B	7F		FF0A 3C PSHX	11111111	11111111
E	7F		00FF FF (write)	11111111	11111111
F	7F		00FE FF (write)	11111111	11111111
10	7F		FF0B 38 PULX	11111111	11111111
13	7F		00FE FF (read)	11111111	11111111
14	7F		00FF FF (read)	11111111	11111111
11	7F	INC.ST	FF0C 4C INCA	11111111	11111111
15	7F		FF0D 4C INCA	11111111	11111111
16	7F		FF0E 4C INCA	11111111	11111111
17	7F		FF0F 4C INCA	11111111	11111111
18	7F		FF10 4C INCA	11111111	11111111
19	7F		FF11 4C INCA	11111111	11111111
1A	7F		FF12 4C INCA	11111111	11111111
1B	7F		FF13 4C INCA	11111111	11111111
1C	7F		FF14 4C INCA	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-1) ok

RESET FF0C RB resetting

CC=11 (--hInzvC) B=34 A=12 X=5678 SP=00FF

FF0C 4C INCA (next step) ok

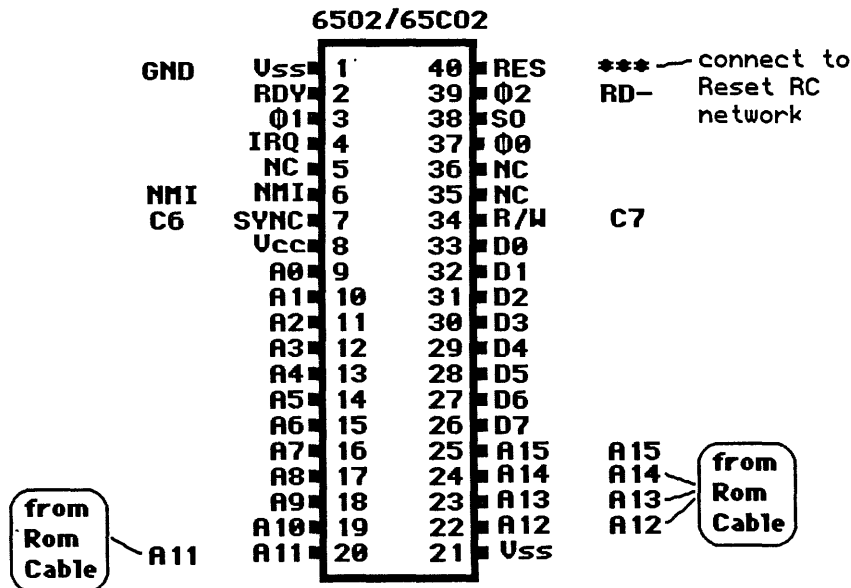
6500 DISASSEMBLER/DEBUGGER (DDB-65)

This version supports the 6500 series chips which have a SYNC output signal. This includes the 6502, and 65C02.

The 6500 series processors require the B analyzer cable.

Debugger Patches

You cannot use the debugger until you have told the UniLab software what location in RAM it can use to store the A register, and where your interrupt routines are located. See the subsection on **The debugger**, below.



Displaying all cycles

Sometimes the trace of your 6500 series processor will show extra bus cycles. You can chose whether or not to display the extra bus cycles by entering **ALLCY** or **ALLCY'**.

Since the extra cycles of the 6502 sometimes contain useful information, you should normally keep them enabled with that processor.

Whether you have **ALLCY** enabled or not, the UniLab can trigger on the extra cycles.

Remember that these extra cycles can cause triggering. For example, you may trigger on the address just after a conditional branch that doesn't actually branch because that byte is fetched but not executed. (Breakpoints at such an address will work.)

Special **STARTUP**

STARTUP usually shows the first cycles of a program after it starts. However, **STARTUP** is specially redefined for the 6500 family because most 6500 series chips fetch a single location hundreds of times after reset, before they actually start program execution.

If you really want to see those first fetches after reset, just enter **RESET NORMT S** .

The debugger

You have to tell the debugger:

- 1) where it can save the A register
- 2) where you have your interrupt handling routine.

After you do that, as described below, you must **SAVE-SYS** to preserve these changes.

Commands required: Where to save the A register

The debugger will save the A register at a reserved RAM location in zero page which you must specify by entering `<addr> =ASAVE`.

Commands required: Where to vector your interrupts

The debugger will install its own pointer at vector location `FFFE-F` and vector your interrupts to a location that you must specify by entering `<addr> =INTADR`.

The system can be saved to the disk with these 2 locations specified by entering **SAVE-SYS** .

Reserved areas: Locations automatically taken by the debugger

An interrupt routine is automatically installed by the debugger at the reserved locations `FFAD-FFBC`. These locations must be **EMENABLE**d for the debugger to work.

Addresses `FE00` to `FEFF` are used by the debugger in such a way that you cannot look at external RAM at those addresses. If this is a problem, you can reassign this area to another starting address by entering `xx00 =DADR` .

Increasing the speed of the breakpoint routine

When your program reaches a breakpoint, your microprocessor executes an Orion breakpoint routine. The breakpoint routine takes 21 machine cycles.

If speed is critical, interrupt response time can be improved by five cycles by patching in a BEQ instruction to the interrupt routine in place of the BNE in the listing below and covering the following 2 bytes with NOOP's. Note that the interrupt routine must then restore the A register from the ASAVE location before returning.

The address of the breakpoint routine image in the UniLab program is **RSTADRIMAGE 4 +**. The routine is listed below. See the Disassembler/Debugger writeup for further details.

```
FFAC      8510          STA ASAVE          (location set by =ASAVE)
FFAE      68           PLA
FFAF      48           PHA
FFB0      2910          AND #10
FFB2      D005          BNE FFBB
FFB4      A510          LDA ASAVE
FFB6      4D1100       JMP INTADR          (address set by =INTADR)
FFB9      8510          STA ASAVE
FFBB      B8           CLV
FFBC      .            (... overlay area begins here)
FFBE      .
```

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

register change: n =A n =X n =Y n =P (status)
n =ASAVE reserved adr page 0 where A is saved on int. n =INTADR
see also HD .(locations FFAE-BC reserved, overlay starts at FFBC)

Trace and breakpoint display

LTARG (Enable emulation memory, set up reset vectors, and load built-in sample program)

Emulator Memory Enable Status:

F =EMSEG

F800 TO FFFF EMENABLE

ok

STARTUP resetting (Reset target system, and show trace of first cycles after reset)

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
0	BF	FFFC	00	read	11111111	11111111 (note reset vector data
1	BF	FFFD	FF	read	11111111	11111111 read from FFFC,D)
2	FF	FF00	A912	LDA #12	11111111	11111111
4	FF	FF02	A2FF	LDX #FF	11111111	11111111
6	FF	FF04	9A	TXS	11111111	11111111 (initialize stack ptr)
8	FF	FF05	A056	LDY #56	11111111	11111111
A	FF	FF07	A9AB	LDA #AB	11111111	11111111
C	FF	FF09	85E0	STA E0	11111111	11111111 (store FEAB at E0)...
E	3F	00E0	AB	write	11111111	11111111 see valid data to
\$00E0)						
F	FF	FF0B	A9FE	LDA #FE	11111111	11111111
11	FF	FF0D	85E1	STA E1	11111111	11111111
13	3F	00E1	FE	write	11111111	11111111
14	FF	FF0F	B1E0	LDA (E0),Y	11111111	11111111
16	BF	00E0	AB	read	11111111	11111111
17	BF	00E1	FE	read	11111111	11111111
18	BF	00E1	FE	read	11111111	11111111 (extra read)
19	BF	FF01	12	read	11111111	11111111 (address is FEAB+Y)
1A	FF	FF11	C8	INX	11111111	11111111
1C	FF	FF12	48	PHA	11111111	11111111
1D	BF	FF13	B1	read	11111111	11111111 (Again, note stack
1E	3F	01FF	12	write	11111111	11111111 address and data)
Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)						

RESET FF16 RB resetting (reset target system, run to breakpoint at \$FF16)

PC=FF16 A=12 P=34(nv-BdIzc) X=FF Y=57 SP=1FF
FF16 E8 INX (next step) ok

65XX PIGGYBACK DISASSEMBLER/DEBUGGER (DDB-65P)

This version supports the R65/11EB, R65/41, and other variations of piggyback ROM chips, as well as the R6511Q. This version is derived from the DDB-65 version but is modified for chips that do not have a SYNC output signal.

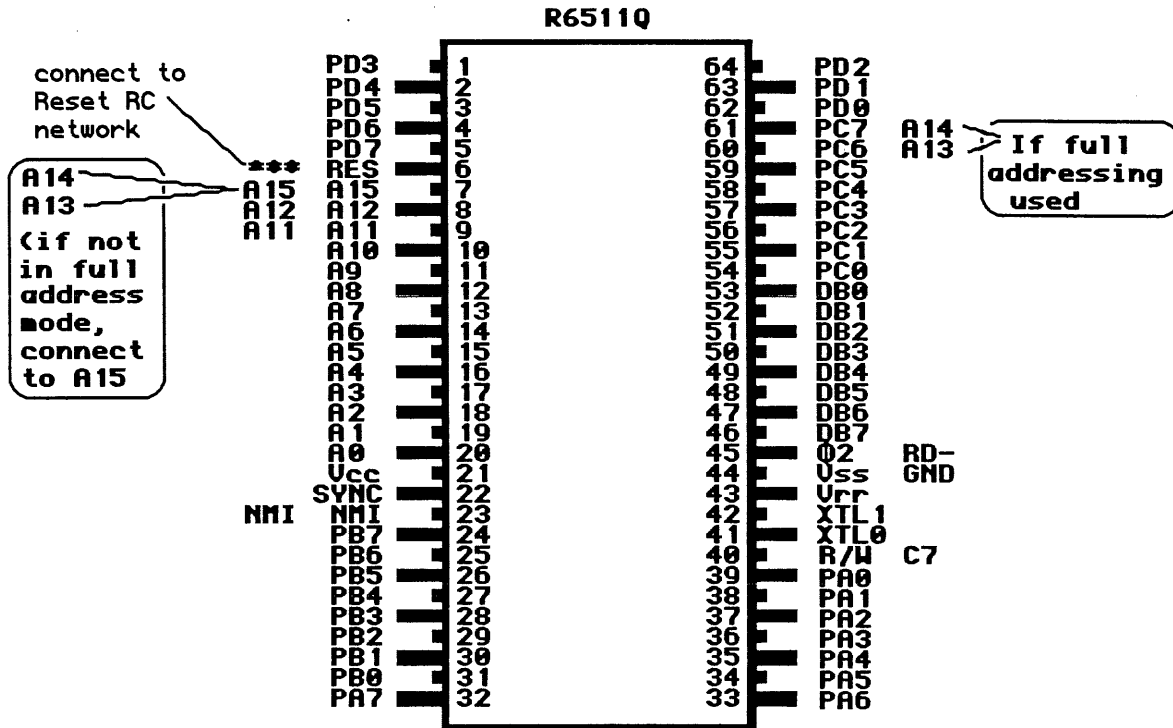
Please see the section on the DDB-65 for other important details, including **Commands required**.

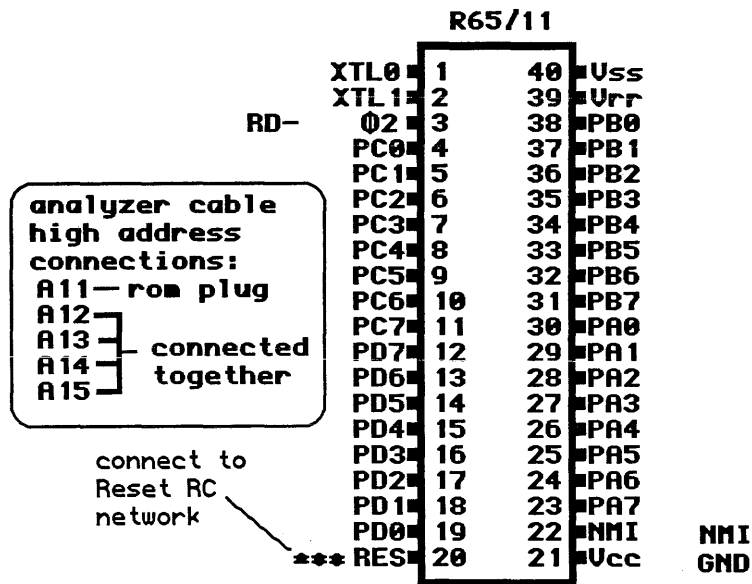
The R65/11EB requires the K analyzer cable.

The R65F11Q requires the B analyzer cable.

Patch word

For operation with the R6511Q you enter **R65Q** to patch the program, then **SAVE-SYS** to save the newly configured software.





Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

register change: n =A n =X n =Y n =P (status)
n =ASAVE reserved adr page 0 where A is saved on int. n =INTADR
see also HD .(locations FFAD-BC reserved, overlay starts at FFBC)

Trace and breakpoint display

R65/11 Trace display

LTARG

ok

STARTUP resetting (reset, and trace. Note that many more cycles are hidden)
(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
0	7F	FFFC	00	fetch	11111111	11111111 (note that there are
1	7F	FFFD	FF	fetch	11111111	11111111 no special values in
2	7F	FF00	A912	LDA #12	11111111	11111111 the CONT column for
4	7F	FF02	A2FF	LDX #FF	11111111	11111111 READ, WRITE or FETCH)
6	7F	FF04	9A	TXS	11111111	11111111
8	7F	FF05	A056	LDY #56	11111111	11111111
A	7F	FF07	A9AB	LDA #AB	11111111	11111111
C	7F	FF09	85E0	STA E0	11111111	11111111
F	7F	FF0B	A9FE	LDA #FE	11111111	11111111
11	7F	FF0D	85E1	STA E1	11111111	11111111
14	7F	FF0F	B1E0	LDA (E0),Y	11111111	11111111
1A	7F	FF11	C8	INY	11111111	11111111
1C	7F	FF12	48	PHA	11111111	11111111
1F	7F	FF13	B1E0	LDA (E0),Y	11111111	11111111
25	7F	FF15	68	PLA	11111111	11111111
29	7F	FF16	E8	INX	11111111	11111111
2B	7F	FF17	E8	INX	11111111	11111111
2D	7F	FF18	E8	INX	11111111	11111111
2F	7F	FF19	E8	INX	11111111	11111111
31	7F	FF1A	E8	INX	11111111	11111111
33	7F	FF1B	E8	INX	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)

R6511Q Trace display

R650 ok (configure disassembler for R6511Q target)
(We've disconnected the piggyback chip and hooked up the R6511Q:)

STARTUP resetting

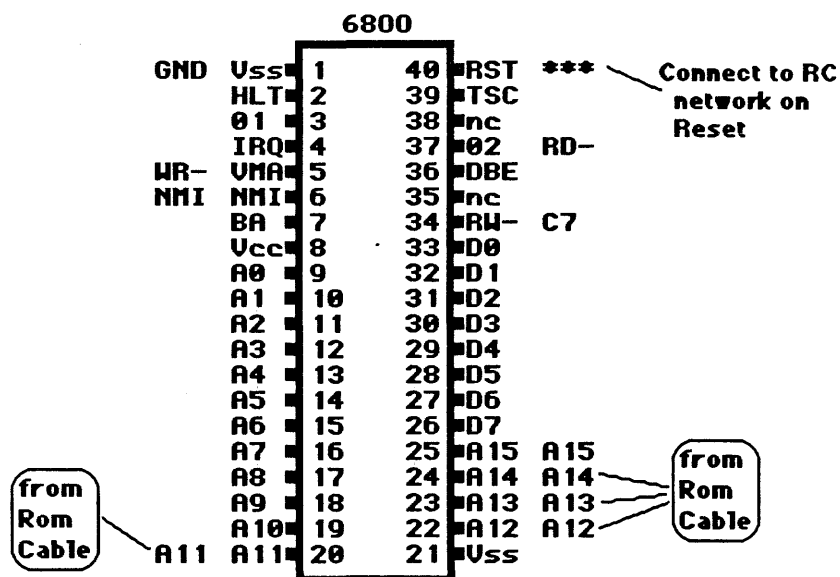
cy#	CONT	ADR	DATA		HDATA	MISC
0	BF	FFFC	00	read	11111111	11111111
1	BF	FFFD	FF	read	11111111	11111111
2	FF	FF00	A912	LDA #12	11111111	11111111
4	FF	FF02	A2FF	LDX #FF	11111111	11111111
6	FF	FF04	9A	TXS	11111111	11111111
8	FF	FF05	A056	LDY #56	11111111	11111111
A	FF	FF07	A9AB	LDA #AB	11111111	11111111
C	FF	FF09	85E0	STA E0	11111111	11111111
F	FF	FF0B	A9FE	LDA #FE	11111111	11111111
11	FF	FF0D	85E1	STA E1	11111111	11111111
14	FF	FF0F	B1E0	LDA (E0),Y	11111111	11111111
1A	FF	FF11	C8	INY	11111111	11111111
1C	FF	FF12	48	PHA	11111111	11111111
1F	FF	FF13	B1E0	LDA (E0),Y	11111111	11111111
25	FF	FF15	68	PLA	11111111	11111111
29	FF	FF16	E8	INX	11111111	11111111
2B	FF	FF17	E8	INX	11111111	11111111
2D	FF	FF18	E8	INX	11111111	11111111
2F	FF	FF19	E8	INX	11111111	11111111
31	FF	FF1A	E8	INX	11111111	11111111
33	FF	FF1B	E8	INX	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)

6800 DISASSEMBLER/DEBUGGER (DDB-68)

This version supports the 6800 processor. It also works with 6808 systems that have an external RAM chip which works at address zero.

The 6800 requires the B analyzer cable.



Extra cycles

The 6800 family often performs extra fetch cycles during instruction execution. The UniLab program hides these cycles from you, to make the trace easier to read.

However, if the disassembler is out of sync, it will interpret extra cycles as instructions, and hide cycles that are not extras

Keeping the disassembler in sync

If the disassembler is out of sync, it will usually fall back into sync after a few cycles.

One way to make sure the disassembler is in sync: always start the trace a few cycles before the area of interest.

A better way to keep the disassembler in sync: start the disassembly at the first cycle of an instruction. Use <n> TN, where n is the cycle number in the trace.

You can look at all cycles by turning off the disassembler with **DASM'**.

Reserved area

When the debugger is engaged by entering

RESET <adr> RB

a NOOP instruction is patched into the target program at location FFB9 and FFBA. These are the only reserved program memory locations. Overlays are installed above location FFBA and automatically restored.

You can thus use the area above FFBA, but most debug operations won't work properly in the 10-30 locations above this address.

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

register change: n **=A** n **=B** n **=CC** n **=IX**
See also **HD** . FFB9-A reserved. Overlay above.

Trace and breakpoint display

LTARG

Emulator Memory Enable Status:

F =EMSEG

F000 TO FFFF EMENABLE

STARTUP resetting

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
0	FF	FFFE	F0	read	11111111	11111111
1	FF	FFFF	00	read	11111111	11111111
2	BF	F000	8E00FF	LDS # FF	11111111	11111111 (stack in page 0)
5	FF	F003	8601	LDAA #1	11111111	11111111
7	FF	F005	36	PSHA	11111111	11111111
9	7F	00FF	01	write	11111111	11111111 (note address & data)
A	FF	F006	C678	LDAB #78	11111111	11111111
C	FF	F008	37	PSHB	11111111	11111111
E	7F	00FE	78	write	11111111	11111111 (note address -1)
F	FF	F009	32	PULA	11111111	11111111
11	FF	00FE	78	read	11111111	11111111 (note address & data)
12	FF	F00A	33	PULB	11111111	11111111
14	FF	00FF	01	read	11111111	11111111 (address & data!
15	FF	F00B	CE1234	LDX # 1234	11111111	11111111
18	FF	F00E	4C	INCA	11111111	11111111
1A	FF	F00F	4C	INCA	11111111	11111111
1C	FF	F010	4C	INCA	11111111	11111111
1E	FF	F011	4C	INCA	11111111	11111111
20	FF	F012	4C	INCA	11111111	11111111
22	FF	F013	4C	INCA	11111111	11111111
24	FF	F014	4C	INCA	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)

RESET F010 RB (reset, run to break at address F010)

PC=F010 A=7A B=01 IX=1234 CC=F1(--HInzvC) SP= FF

F010 4C INCA (next step)

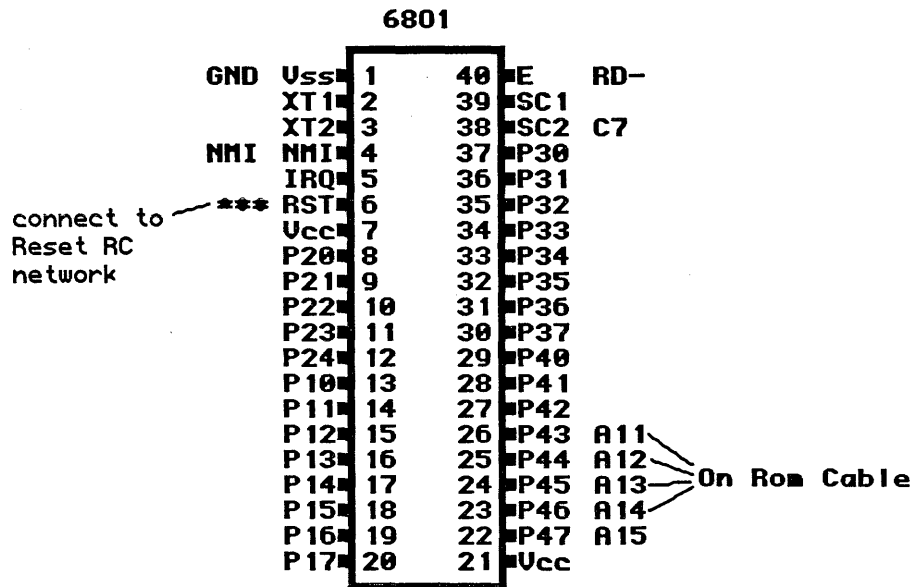
6801 DISASSEMBLER/DEBUGGER (DDB-681)

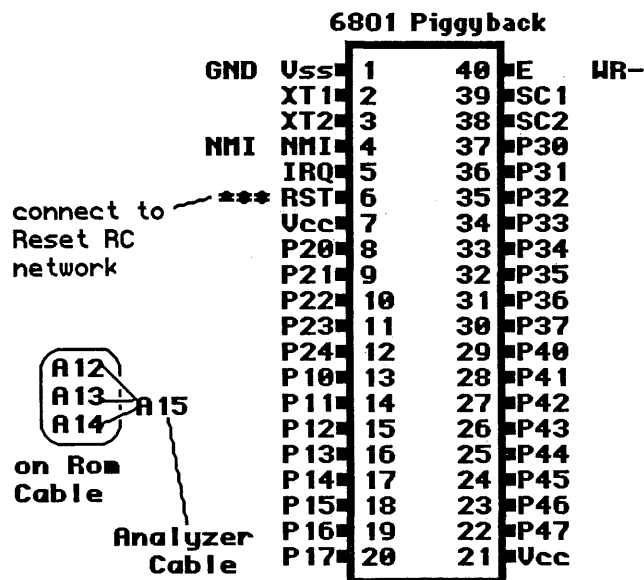
This version supports the 6801 and 6803 processors. It supports the piggyback chips without requiring a patch word.

The debugger is capable of displaying and changing internal memory, even though read and write cycles to this memory do not show on the bus.

This software package is similar to the DDB-68. Please read the DDB-68 writeup for important information.

The 6801 requires the B analyzer cable. The piggyback requires the N analyzer cable.





Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:
 Register change: n =A n =B n =CC nn =X
 See also **HD** . \$FFB9,A reserved. Overlay area = \$FFBC to reset vectors

ALLCY shows hidden bus cycles, **ALLCY'** hides them.

Trace and breakpoint display

LTARG (Load built-in sample program)

STARTUP resetting (Reset target system, start analyzer, show trace from reset vector)

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
-1	FF	FFFE	FF	(read)	11111111	11111111 (Note reset vector)
0	FF	FFFF	00interrupted	11111111	11111111 FFFE, FFFF come up
1	FF	FF00	8E00FF	LDS #FF	11111111	11111111 first on bus)
4	FF	FF03	86FF	LDAA #FF	11111111	11111111
6	FF	FF05	9705	STAA \$5	11111111	11111111
8	7F	0005	FF	(write)	11111111	11111111 (note address and data)
9	FF	FF07	8600	LDAA #0	11111111	11111111
B	FF	FF09	9707	STAA \$7	11111111	11111111
D	7F	0007	00	(write)	11111111	11111111
E	FF	FF0B	8616	LDAA #16	11111111	11111111
10	FF	FF0D	C627	LDAB #27	11111111	11111111
12	FF	FF0F	CE0E49	LDX #E49	11111111	11111111
15	FF	FF12	A100	CMPA \$0,X	11111111	11111111
18	FF	0E49	49	(read)	11111111	11111111 (Note value in address
is						
19	FF	FF14	4C	INCA	11111111	11111111 contents of X
register!)						
1B	FF	FF15	4C	INCA	11111111	11111111
1D	FF	FF16	4C	INCA	11111111	11111111
1F	FF	FF17	4C	INCA	11111111	11111111
21	FF	FF18	4C	INCA	11111111	11111111
23	FF	FF19	4C	INCA	11111111	11111111
25	FF	FF1A	4C	INCA	11111111	11111111
Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-1)						

RESET FF14 RB resetting (Reset target board, and break at address FF14)

CC=39 (--HINzVC) B=27 A=16 X=0E49 SP=00FF
FF14 4C INCA (next step) ok

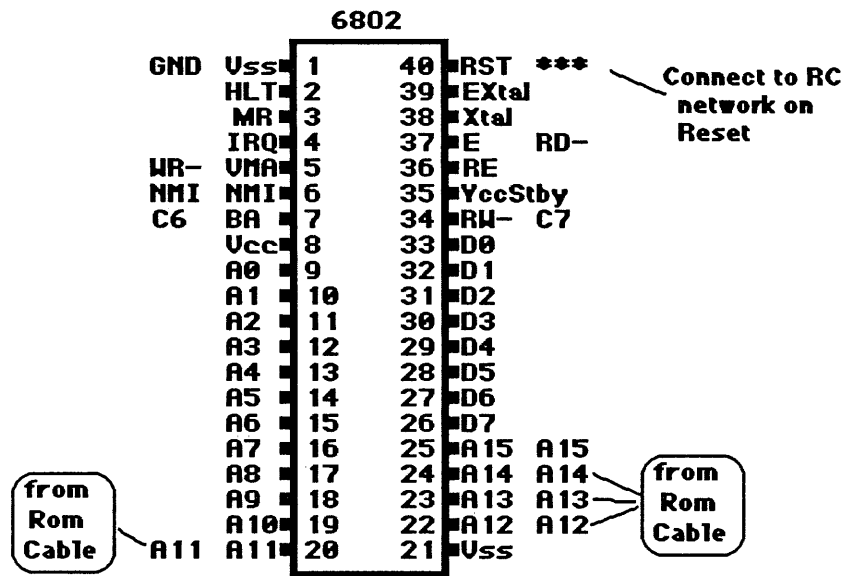
6802 DISASSEMBLER/DEBUGGER (DDB-682)

This version supports the 6802 processor.

The debugger is capable of displaying and changing internal memory, even though read and write cycles to this memory do not show on the bus.

This software package is derived from the DDB-68 version. Please read the DDB-68 writeup for more important details, and refer to the **Trace and breakpoint display** in that section.

The 6802 requires the B analyzer cable.



DDB-682 additional reserved resources

The DDB-682 version debugger requires 4 reserved locations in the zero page on-chip RAM. The RAM is used by the UniLab software to save information while reading or writing the internal RAM.

The default assignment for these locations is 50-53, but you can change the starting address of that area with `<address> =ZP`.

See the writeup on DDB-68 for more information on **Reserved areas**.

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

```
$FFB9,A reserved for breakpoint, $FFBA-$FFF7=overlay
4 bytes at $50-$53 reserved, change with xx =ZP .
Register change: n =A n =B n =IX n =CC
Also see HD
```

6805 DISASSEMBLER/DEBUGGER (DDB-685)

This version supports the 6805 family including the 6805E2, the 6805E3, and piggyback chips as well.

The 6805 requires the B analyzer cable. The piggyback requires the M analyzer cable.

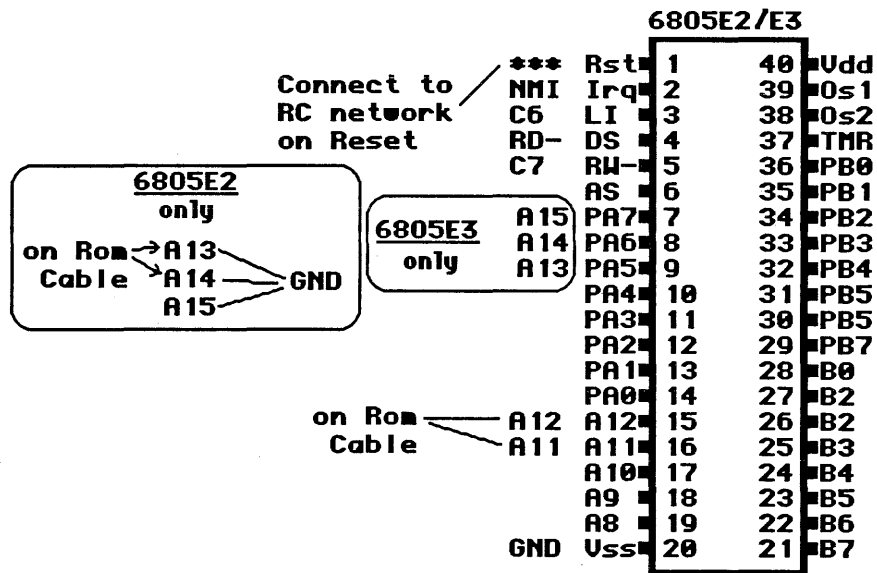
Patch word

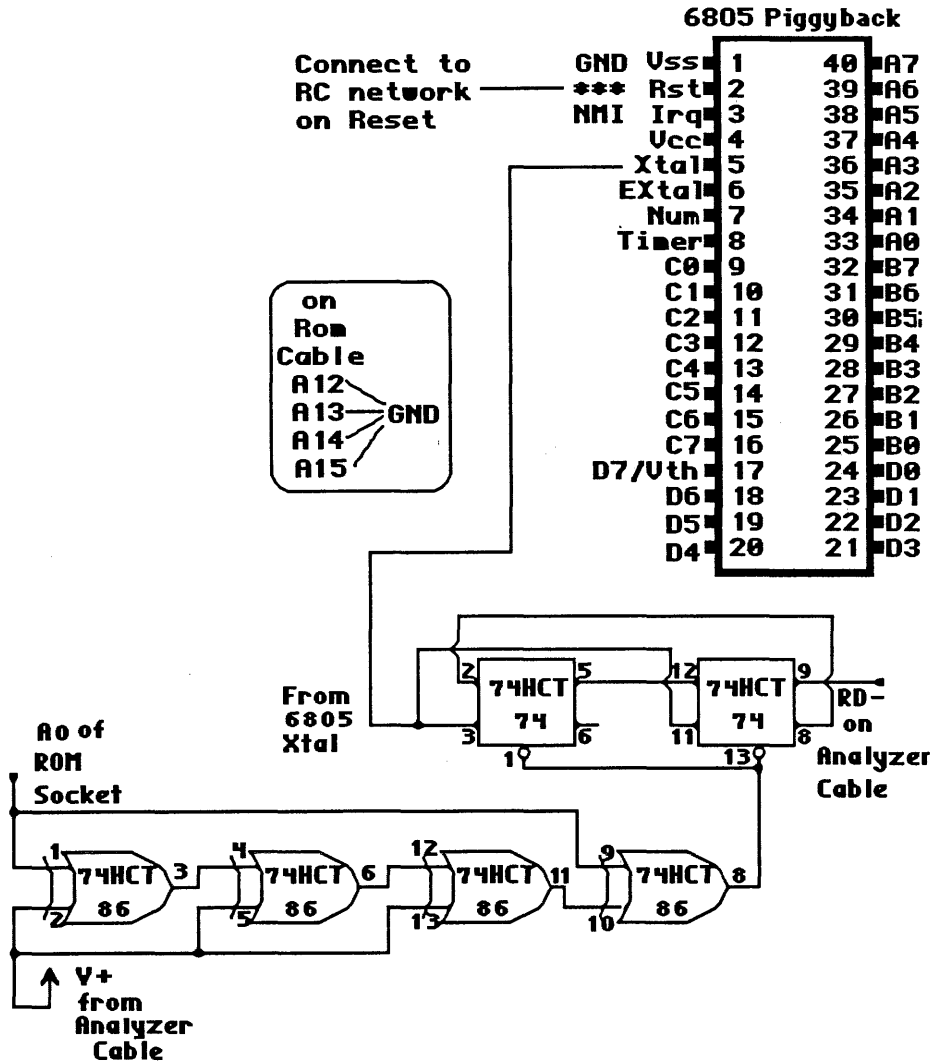
For operation with piggyback chips, you enter **PBACK** to patch the program.

For the HD6305 processor, you enter **HD6305**.

For the 6805E3, you enter **RCA-E3**.

Use **SAVE-SYS** to save the newly configured software.





Note: the logic gates shown above are part of the M cable, not additional hardware for you to put on your board.

Reserved area

When the debugger is engaged by entering

RESET adr **RB**

a NOOP instruction is patched into the target program at locations:

1FB9 for the 6805E2
0FB9 for the piggyback
FFB9 for the 6805E3

Overlays are installed above the reserved area, in location \$xFBA (x equals: 0 for 6805E2, 1 for piggyback, F for 6805E3). You can use the area above \$xFBA but most debug operations won't work properly in the 30 locations above this address.

Other locations used by the debugger

Location 3F is used by the debugger and should not be used. You may change this location by entering <addr> **=ZP**, where nn is the location in RAM reserved for the debugger. This address must be greater than 20 and less than 6F.

Showing all cycles

Sometimes the trace of your processor will show extra bus cycles, recorded while your processor was busy executing code. You can choose whether or not to display the extra bus cycles by entering **ALLCY** or **ALLCY'**.

Since the extra cycles contain useful information you can choose to show them, or you can show only instruction cycles if you prefer.

Enter **ALLCY** to show most useful extra cycles in the trace, **ALLCY'** turns this mode off and shows only instruction cycles. RAM may be added to your circuit to "shadow" the operation of the internal RAM and allow correct data to be seen on the trace data column for internal RAM r/w.

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

Register change: n **=X** n **=A** n **=CC**
\$xFB9,A reserved for breakpoint, \$xFBA-\$xFDA=overlay
(x=1 for 6805E2, x=0 for piggyback)
\$3F reserved, change with xx **=ZP**
Also see **HD**

Trace and breakpoint display

LTARG (Load built-in sample program, set up emulation memory)

Emulator Memory Enable Status:

F =EMSEG
1800 TO 1FFF EMENABLE

STARTUP resetting

(from top of buffer)

cy#	CONT	ADR	DATA	HDATA	MISC
-1	BF	1FFE	1F (read)	11111111	11111111
0	BF	1FFF	00 (read)	11111111	11111111
1	BF	1F00	AEinterrupted	11111111	11111111
2	FF	1F00	AE0E LDX #E	11111111	11111111
4	FF	1F02	D60800 LDA \$800,X	11111111	11111111
8	BF	080E	00 (read)	11111111	11111111 (note address
9	FF	1F05	B74E STA \$4E	11111111	11111111 and control
C	3F	004E	00 (write)	11111111	11111111 values for
D	FF	1F07	6C40 INC \$40,X	11111111	11111111 fetch ,r/w)
11	BF	004E	4E (read)	11111111	11111111 (note address,
12	3F	004E	01 (write)	11111111	11111111 good data?)
13	FF	1F09	4C INCA	11111111	11111111
16	FF	1F0A	4C INCA	11111111	11111111
19	FF	1F0B	4C INCA	11111111	11111111
1C	FF	1F0C	4C INCA	11111111	11111111
1F	FF	1F0D	4C INCA	11111111	11111111
22	FF	1F0E	4C INCA	11111111	11111111
25	FF	1F0F	4C INCA	11111111	11111111
28	FF	1F10	4C INCA	11111111	11111111
2B	FF	1F11	4C INCA	11111111	11111111
2E	FF	1F12	4C INCA	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-1)

RESET 1F0A RB resetting (Reset target, trigger analyzer on RESET vec)

SP= 7F X=0E A=01 CC=19(---HInzC)

1F0A 4C INCA (next step) ok

6805 Piggyback Trace display

Here is a listing of a trace using the 6805 piggyback μ P to show the differences from the 6805E2 version:

PBACK ok (convert to piggyback mode)

LTARG (enable emulation memory, load built-in test program)

Emulator Memory Enable Status:

F =EMSEG

800 TO FFF EMENABLE

ok

STARTUP resetting

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
-1	FF	0FFE	0F (read)		11111111	11111111
1	FF	8000	FFinterrupted	11111111	11111111
2	FF	0F00	AE0E	LDX #E	11111111	11111111
4	FF	0F02	D60800	LDA \$800,X	11111111	11111111
A	FF	0F05	B74E	STA \$4E	11111111	11111111
F	FF	0F07	6C40	INC \$40,X	11111111	11111111
16	FF	0F09	4C	INCA	11111111	11111111
1A	FF	0F0A	4C	INCA	11111111	11111111
1E	FF	0F0B	4C	INCA	11111111	11111111
22	FF	0F0C	4C	INCA	11111111	11111111
26	FF	0F0D	4C	INCA	11111111	11111111
2A	FF	0F0E	4C	INCA	11111111	11111111
2E	FF	0F0F	4C	INCA	11111111	11111111
32	FF	0F10	4C	INCA	11111111	11111111
36	FF	0F11	4C	INCA	11111111	11111111
3A	FF	0F12	4C	INCA	11111111	11111111
3E	FF	0F13	4C	INCA	11111111	11111111
42	FF	0F14	4C	INCA	11111111	11111111
46	FF	0F15	4C	INCA	11111111	11111111
4A	FF	0F16	4C	INCA	11111111	11111111
4E	FF	0F17	4C	INCA	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-1)

RESET F10 RB resetting

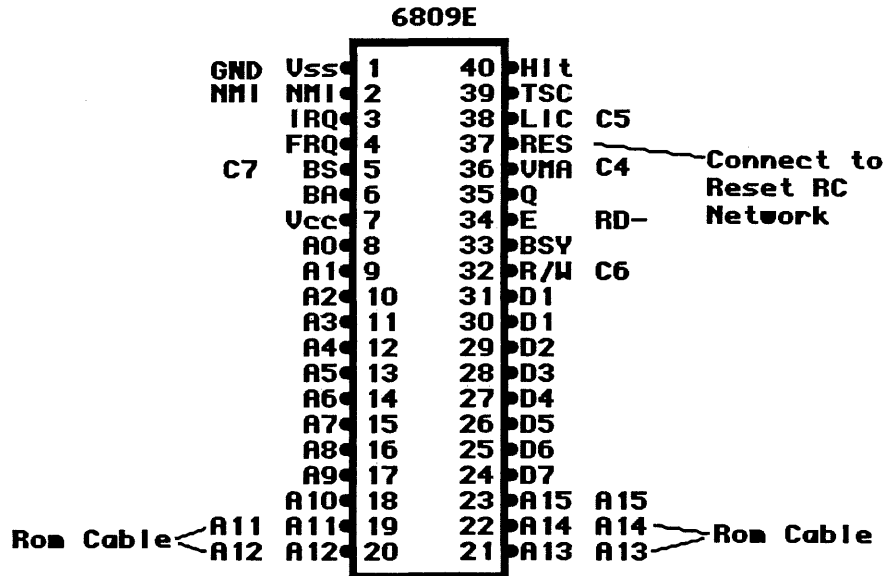
SP= 7F X=0E A=36 CC=09(---hInzC)

0F10 4C INCA (next step) ok

6809E DISASSEMBLER/DEBUGGER (DDB-689)

This version supports the 6809E.

The 6809E requires the B analyzer cable.



The disassembler and hidden cycles

The 6809E performs extra fetch cycles during instruction execution. Since these extra cycles are predictable, the disassembler hides them from you.

The analyzer connections to the LIC, and AVMA pins are used to distinguish opcode fetch instruction cycles from extra cycles. Most non-interesting ones are filtered out, and only useful read/write will show in the trace. Turn off the disassembler with **DASM'** to view all cycles.

Reserved area

When the debugger is engaged by entering **RESET** adr **RB**, a NOOP instruction is patched into the target program at location FFB9. This is the only reserved location.

Other areas used by Debugger

Overlays are installed above location FFBA and automatically restored. You can thus use the area above FFBA but most debug operations won't work properly in the 10-30 locations above this address.

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

register change: n =A n =B nn =X nn =Y n =U n =CC n =DP
 FFB9-A reserved. Overlay above. See also **HD** .

Trace and breakpoint display

LTARG (load sample program, set up default emulation memory area)

Emulator Memory Enable Status:

F =EMSEG

F000 TO FFFF EMENABLE

STARTUP resetting (Reset the target processor, and trace the first cycles after it starts up)

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
-1	DF	FFFE	FF	(interrupt)	11111111	11111111 (Note Reset
vector						
1	7F	FFFF	00	(read)	11111111	11111111 fetch from FF00)
2	5F	FF00	10CE00FF	LDS #FF	11111111	11111111
6	5F	FF04	8612	LDA #12	11111111	11111111
8	5F	FF06	8E0000	LDX #0	11111111	11111111
B	5F	FF09	A780	STA ,X+	11111111	11111111
D	4F	FF0B	08	(read)	11111111	11111111 (extra cycle..)
10	3F	0000	12	(write)	11111111	11111111(note good
address						
11	5F	FF0B	0800	ASL 0	11111111	11111111 and data)
14	4F	0000	12	(read)	11111111	11111111(
Read-Modify-Write						
16	3F	0000	24	(write)	11111111	11111111 clearly shown)
17	5F	FF0D	A780	STA ,X+	11111111	11111111
19	4F	FF0F	10	(read)	11111111	11111111
1C	3F	0001	12	(write)	11111111	11111111
1D	5F	FF0F	109E00	LDY 0	11111111	11111111
21	5F	0000	24	(read)	11111111	11111111
22	7F	0001	12	(read)	11111111	11111111
23	5F	FF12	3420	PSHS Y/	11111111	11111111
27	5F	00FF	12	(read)	11111111	11111111
28	1F	00FE	12	(write)	11111111	11111111(note stack
address						
29	3F	00FD	24	(write)	11111111	11111111 and write data)
2A	5F	FF14	3520	PULS Y/	11111111	11111111
2E	5F	00FD	24	(read)	11111111	11111111(data comes off..
2F	5F	00FE	12	(read)	11111111	11111111 ..indicates good
30	7F	00FF	12	(read)	11111111	11111111 stack)
31	5F	FF16	4C	INCA	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-1)

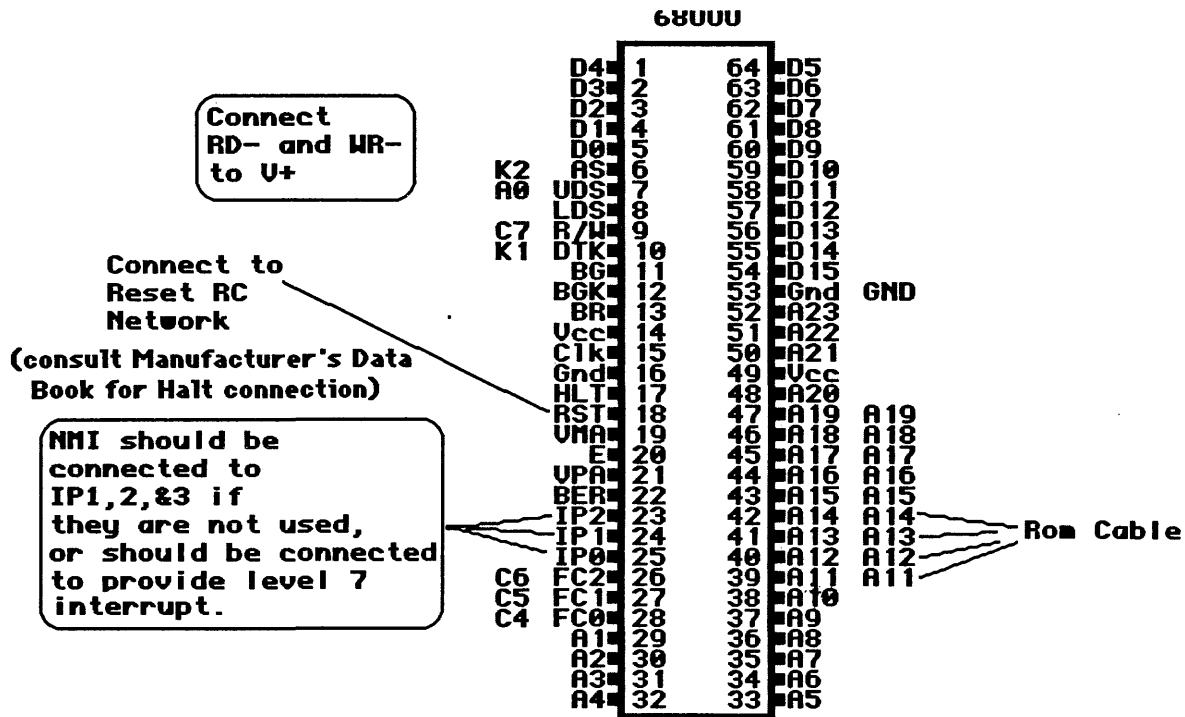
RESET FF2A RB resetting (Reset target, set breakpoint to occur just before opcode at address FF2A is executed)

PC=FF2A A=26 B=02 X=0002 Y=2412 U=FEFE DP=00 CC=D0(EFhInzvc) SP=00FF
FF2A 4C INCA (next step) ok

68000 DISASSEMBLER/DEBUGGER (DDB-68K)

This version supports the 68000 processor.

The 68000 requires the P analyzer cable.



Reserved resources

The 68000 debugger uses one trap vector for its operation, as well as a reserved area. Initially it uses trap 0, but this can be changed by entering <hex digit> =TRAP, where the digit is the new trap vector number to be used (from 0 to F).

The reserved area is 07A8 through 07AB.

The debugger needs the first 2K of emulation ROM enabled.
Use:

0 TO 7FF EMENABLE

then use **ALSO** <range> **EMENABLE** to enable any other areas you need.

Consult separate Disassembly/Debugger writeup if you need to use RAM at these addresses, 0 to 7FF.

Other resources

The overlay area is above the reserved area, at 07AC through 07FC. You can use this area, but most debug operations won't work properly in the 10-40 locations above this address. The debugger operates from the supervisor mode.

Addresses interpreted as interrupts

If your microprocessor tries to access an area below 400 hex occurs, it will be interpreted as an interrupt. That area is designated as an interrupt vector area. If you want to use any of these addresses for your program, you can move the boundary down to 100 by typing in:

```
100 ' INTMAX !
```

and then use **SAVE-SYS** to save the newly configured software.

32-bit addresses and . . .

The UniLab software works with the 32-bit external address space of the 68000 by using several different commands to set the upper 16 bits of the address. The different commands affect different groups of UniLab words, as described below.

. . . Register change words

All of the register change words such as **=D3** and **=USP** can accept 32-bit values if the number ends in a decimal point.

For instance, entering **12345678. =A0** will send a full 32-bit long word into register A0.

. . . Memory access words

The words to examine and modify target memory (**MDUMP**, **MMOVE**, **MM!**, **M?**, etc.) can, indirectly, make use of 32-bit addresses.

Initially the value of the upper 16 bits of the full address is set to 0, but can be changed. The command **<value> =TOMV** will change the upper 16 bits of the destination address, while **<value> =FROMMV** will change the upper 16 bits of the source address.

Those values will remain constant until you use **=TOMV** or **=FROMMV** again.

The values of the lower 16 bits are entered as arguments to the **MDUMP**, **MM?**, etc. commands.

. . . Breakpoint words

The command **<value> =TOGO** sets the upper 16 bits of the 32-bit address used by the debugger words such as **G**, **GB**, or **RB**.

The CONT column

The upper 4 bits in the CONT column in the analyzer trace represent the state of the R/W line and the FC1, FC2, and FC3 pins. These are used by the disassembler to give the following cycle types:

0 ?cy=0 (unassigned)	8 ?cy=8 (unassigned)
1 write (write user data)	9 read (read user data)
2 pwrit (write user program)	A fetch (read user program)
3 ?cy=3 (unassigned)	B ?cy=B (unassigned)
4 ?cy=4 (unassigned)	C ?cy=C (unassigned)
5 sdwrt (write supervisor data)	D sread (read sup. data)
6 spwrt (write supervisor program)	E sftch (read supervisor pgm)
7 ?cy=7 (unassigned)	F iack (interrupt acknow.)

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

Register change: n =R0 n =A0 (use nnn. for 32bit).

Also n =USP n =SR

n =TRAP changes trap vector.

n =TOGO n =TOMV n =FROMMV for long word addr's.

SHOWM and **SHOWM'** to turn MISC column on/off.

\$07A8,B reserved for breakpoint, \$07AC-\$07FC=overlay. Also see **HD**.

Trace and breakpoint display

LTARG (Load built-in sample program, set up default emulation memory)

Emulator Memory Enable Status:

0 =EMSEG
0 TO 7FF EMENABLE

STARTUP resetting (Issue reset to target, trace first few cycles)

(from top of buffer)

cy#	CONT	ADR	DATA	MISC
0	E0	0000	0000	iread (note stack ptr) 11111111
1	E0	0002	1100	iread 11111111
2	E0	0004	0000	iread (and program ctr) 11111111
3	E0	0006	0400	iread 11111111
4	E0	0400	46FC2700	MOVE \$2700,SR 11111111
6	E0	0404	91C8	SUB.L A0,A0 11111111
8	E0	0406	223C00021706	MOVE.L \$21706,D1 11111111
B	E0	040C	4E71	NOP 11111111
C	E0	040E	4E71	NOP 11111111
D	E0	0410	207C12345678	MOVE.L \$12345678,A0 11111111
10	E0	0416	48E700C0	MOVEM.L /A1/A0,-(A7) 11111111
12	E0	041A	4CDF0300	MOVEM.L (A7)+,/A0/A1 11111111
13	50	10FE	FFFF	sdwrt (write to supervisor stack)
14	50	10FC	FFFF	sdwrt 11111111
15	50	10FA	5678	sdwrt (note good data on this write and)
16	50	10F8	1234	sdwrt (read on cycles below...)
18	E0	041E	5400	ADDQ.B \$2,D0 11111111
19	D0	10F8	1234	read 11111111
1A	D0	10FA	5678	read 11111111
1B	D0	10FC	FFFF	read 11111111
1C	D0	10FE	FFFF	read 11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)

RESET 41E RB resetting (Reset target, get debug control just before address 41E is executed)

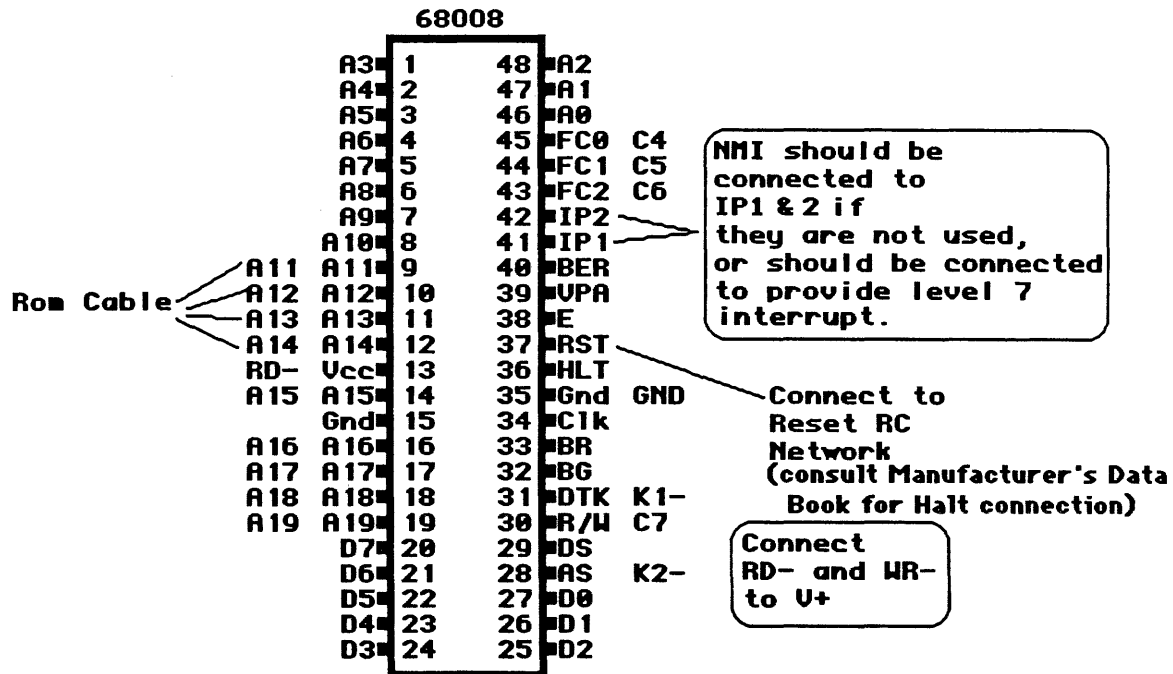
D0= FFFF 00E4 D1= 0002 1706 D2= 0000 007F D3= FFFF FFFF D4= FFFF FFFF
D5= FFFF FFFF D6= FFFF 7FFF D7= FFFF FFFF A0= 1234 5678 A1= FFFF FFFF
A2= FFFF FFFF A3= FFFF FFFF A4= FFFF FFFF A5= FFFF FFFD A6= 0000 0008
PC= 0000 041E USP= FFFF FFFF SSP= 0000 1100 SR= 2700 (t.S..III...xnzvc)
041E 5400 ADDQ.B \$2,D0 (next step) ok

68008 DISASSEMBLER/DEBUGGER (DDB-688)

This version supports the 68008 processor.

The 68008 requires the P analyzer cable.

Please refer to the writeup on the 68000 for all details except for the **Trace and breakpoint display** and the cable wiring diagram.



Trace and breakpoint display

LTARG

Emulator Memory Enable Status:

0 =EMSEG

0 TO 7FF EMENABLE

ok

STARTUP resetting

(from top of buffer)

cy# CONT ADR DATA

0	E0	0000	00	iread
1	E0	0001	00	iread
2	E0	0002	0F	iread
3	E0	0003	00	iread
4	E0	0004	00	iread
5	E0	0005	00	iread
6	E0	0006	04	iread
7	E0	0007	00	iread
8	E0	0400	46FC2700	MOVE #\$2700,SR
C	E0	0404	91C8	SUB.L A0,A0
E	E0	0404	91C8	SUB.L A0,A0
10	E0	0406	223C00021706	MOVE.L #\$21706,D1
16	E0	040C	4E71	NOP
18	E0	040E	4E71	NOP
1A	E0	0410	207C12345678	MOVE.L #\$12345678,A0
20	E0	0416	48E700C0	MOVEM.L /A1/A0,-(A7)
24	E0	041A	4CDF0300	MOVEM.L (A7)+,/A0/A1
26	50	0EFE	FF	sdwrt
27	50	0EFF	FF	sdwrt
28	50	0EFC	FF	sdwrt
29	50	0EFD	FF	sdwrt

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)

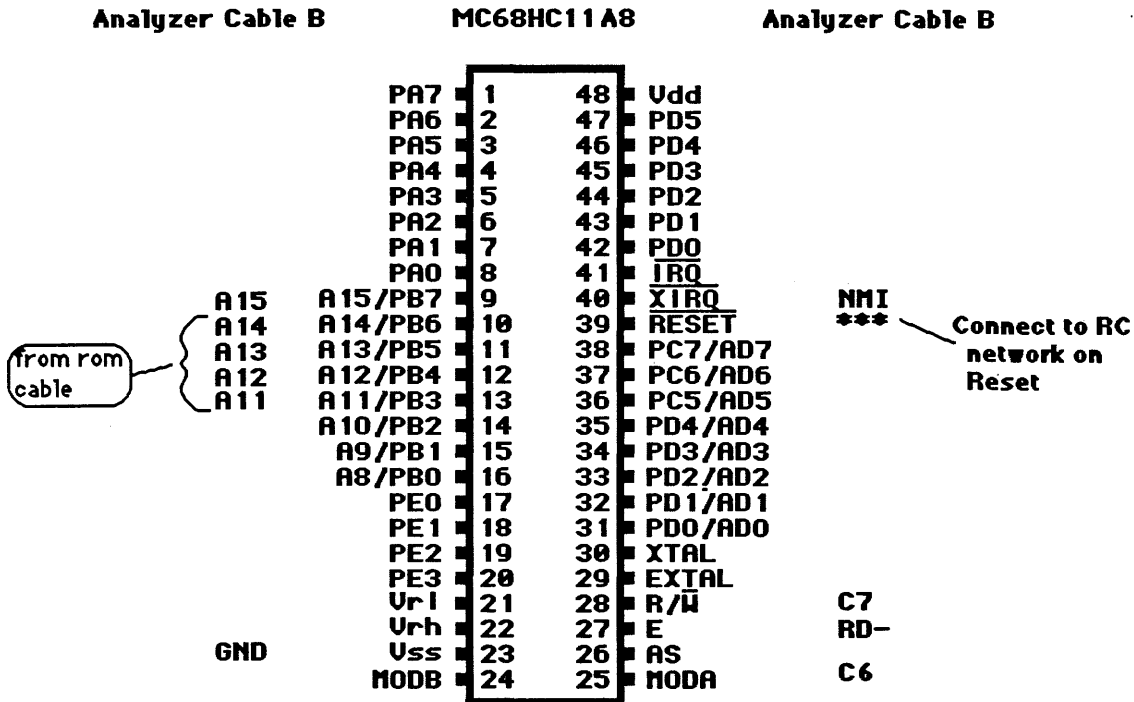
RESET 424 RB resetting

D0= FFFF FF41 D1= 0002 1706 D2= FFFF FFFF D3= FFFF FFFF D4= FFFF FF07
D5= FFFF FFFF D6= FFFF FFFF D7= FFFF FFFF A0= 1234 5678 A1= FFFF FFFF
A2= FFFF FFFF A3= FFFF FFFF A4= FFFF FFFF A5= FFFF FFFF A6= FFFF FFFF
PC= 0000 0424 USP= FFFF FFFF SSP= 0000 0F00 SR= 2700 (t.S..III...xnzvc)
0424 5400 ADDQ.B #\$2,D0 (next step) ok

68HC11 DISASSEMBLER/DEBUGGER (DDB-611)

This version supports the 68HC11 processor.

The 68HC11 requires the B analyzer cable.



Reserved area

The reserved area is FF71 to FF73.

Overlay area

The overlay area is from FF74 to FFBF.

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

Register change: n =**A** n =**B** n =**CC** nn =**XX** nn =**Y**
 See also **HD**. \$FF71-73 reserved. Overlay area = \$FF74-FFBF.
 Display special function register names: **SHOW-REGS**
 Show - don't show reg. names: **SHOW-NAMES / HIDE-NAMES**
 n =**SFPAGE** sets page where special function registers are mapped.

Trace and breakpoint display

LTARG

STARTUP resetting (reset the target, show first cycles of trace)

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
-1	FF	FFFE	FF	(read)	11111111	11111111
0	FF	FFFF	00interrupted	11111111	11111111
1	BF	FF00	8E00FF	LDS #FF	11111111	11111111 (set stack)
4	BF	FF03	07	TPA	11111111	11111111
6	BF	FF04	84BF	ANDA #BF	11111111	11111111 (enable nmi)
8	BF	FF06	06	TAP	11111111	11111111
A	BF	FF07	86FF	LDAA #FF	11111111	11111111
C	BF	FF09	9705	STAA 5	11111111	11111111 (note valid
E	7F	0005	FF	(write)	11111111	11111111 data)
F	BF	FF0B	8600	LDAA #0	11111111	11111111
11	BF	FF0D	9707	STAA 7	11111111	11111111
13	7F	0007	00	(write)	11111111	11111111
14	BF	FF0F	8616	LDAA #16	11111111	11111111
16	BF	FF11	C627	LDAB #27	11111111	11111111
18	BF	FF13	CE1234	LDX #1234	11111111	11111111
1B	BF	FF16	18CE5678	LDY #5678	11111111	11111111
1F	BF	FF1A	A100	CMPA 0,X	11111111	11111111 (note
<i>address</i>						
22	FF	1234	31	(read)	11111111	11111111 = x
<i>contents</i>						
23	BF	FF1C	4C	INCA	11111111	11111111
25	BF	FF1D	4C	INCA	11111111	11111111
27	BF	FF1E	4C	INCA	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-1)

RESET FF1C RB resetting (Set first breakpoint at FF1C, display regs)

A = 16 B = 27 X = 1234 Y = 5678 SP = 00FF CC = B9 (SxHINzvc)
 FF1C 4C INCA (next step) ok

8048 DISASSEMBLER/DEBUGGER (DDB-48)

This version supports the 8048 family including the 8035, 8039, 8040, 8049, and 8050. The processor may be a piggyback version or normal chip operating with a separate ROM (in expanded mode).

In expanded mode all members of the 8048 family require the E analyzer cable. The piggyback chips require the F analyzer cable.

Patch word

If you are using the National piggyback ROM processor (NS87P50), enter **PIGGYBACK** to patch the program. Use **SAVE-SYS** to make this change permanent.

If you have no external memory in your piggyback system, only the reset wire and the ROM plug need be connected. However, you will need to ground **K1** and all unused address lines.

e Cable

8048

	TO	1	40	Ucc
	XT1	2	39	T1
	XT2	3	38	P27
RES	RST	4	37	P26
	SS	5	36	P25
NMI	INT	6	35	P24
	EA	7	34	P17
K1	RD-	8	33	P16
RD-	PSN	9	32	P15
WR-	WR-	10	31	P14
	ALE	11	30	P13
	D0	12	29	P12
	D1	13	28	P11
	D2	14	27	P10
	D3	15	26	Udd
	D4	16	25	PRG
	D5	17	24	P23
	D6	18	23	P22
	D7	19	22	P21
	Uss	20	21	P20

The disassembler and hidden cycles

The 8048 performs extra fetch cycles during instruction execution. Since these extra cycles are predictable, the disassembler hides them from you.

You can look at these cycles by turning the disassembler off with **DASM'**.

Reserved locations

The debugger requires 5 reserved bytes (for example, 7FB-7FF or FFB-FFF) at the top of each 2K memory bank where the debugger is to be used.

These locations are used to disable timer and external interrupts before the processor is stopped at a breakpoint and also to re-enable them (if you have selected that mode by entering **ENI** or **ENTCTI**) and return to the program. The required instructions are patched in when you enter **RESET <adr> RB** for any address in that bank.

Overlay areas

The top 21 (hex) bytes of the page are used for overlaying the debug routines. Since this area is restored between debug steps, you can use it for program storage but you cannot use the debugger on that part of the program.

Block writes to memory also use and restore target locations 7BD-7DD, FBD-FDD, etc. so these addresses can be debugged, but block moves to target memory cannot be done from these locations.

How to set breakpoints

If you want to debug in more than one 2K bank, you must enter **RESET adr RB** for an address in each bank to install the reserved bytes at the top of the page. Once this is done, you can use **n RB** to set breakpoints in any of the active banks until the program is reloaded. While you are stopped at a breakpoint, you can use **GB**, **G**, or **GW** to transfer control within that bank.

Naming registers

The 7 working registers and A, PSW, and T are displayed automatically at the breakpoint. If you are not changing register banks much, you can assign mnemonic names to R0-R7.

For example, if you enter 6 **RNAME** USER# the breakpoint display will print USER# instead of R6.

Altering registers

You can change any of these registers while stopped at a breakpoint by entering the desired contents followed by a space then the register name with an equal sign. For example, 12 **=A** will put 12 in the A register.

Accessing external RAM

If you have external memory, you can access it with any of the commands used to read and write emulation memory such as **MDUMP**, **MFILL**, **MMOVE**, **M?**, **MM?**, **M!**, and **MM!**. They will go to external memory if the address isn't enabled for emulation.

If you have external memory that uses the same addresses as emulated program memory, you can fool the UniLab by setting a bit that is not decoded by the hardware. For example, if your hardware does not look at A15, (while the UniLab's emulation ROM does) use 8000 to address external location 0. The UniLab's emulation ROM then sees address 8000 and so does not respond, while the hardware sees address 000.

You can also use **INTRAM** or **EXTRAM** to select whichever memory you want. This will patch the debugger to use LDC or LDE instructions for looking at non-emulated memory. Notice that if you try to look at the internal memory locations that are used as R0, R1, and PSW as memory (e.g., **MDUMP**) you will not get the right results. These registers are used by the downloaded Orion routine, which displays memory.

Special STARTUP

STARTUP is specially redefined for the 8048 family because most chips fetch a single location hundreds of times after reset before they actually start program execution. If you really want to see the first fetches after reset, just enter **RESET NORMT S.**

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

m n **OUT** writes m to port n. n **INP** reads port n.
register change: n **=A** n **=R0** n **=R1** ... n **=R7** n **=T**
n **RNAME** name assigns 10 char name to reg n. op **EXEC** executes opcode.

ENI enables interrupts. **DISI** disables. **ENTCTI**, **DISTCTI** for counter.

Type **EXTRAM** or **INTRAM** to select ram mode.

Use 80nn or **TRAM** nn to access location nn with **MDUMP**, etc.

See also HD .(locations 7FA-F reserved, overlay starts at 7DE)

Trace and breakpoint display

LTARG (set up default emulation memory, load sample program)

Emulator Memory Enable Status:

E =EMSEG

0 TO 7FF EMENABLE

ok

STARTUP resetting (issue reset to target, and capture cycles after reset on bus)

(from top of buffer)

cy#	CONT	ADR	DATA	HDATA	MISC
-1	FE 0000	2312	MOV A,#12	11111111	11111111
1	FE 0002	17	INC A	11111111	11111111
2	FE 0003	A8	MOV R0,A	11111111	11111111
3	FE 0004	17	INC A	11111111	11111111
4	FE 0005	A9	MOV R1,A	11111111	11111111
5	FE 0006	17	INC A	11111111	11111111
6	FE 0007	AA	MOV R2,A	11111111	11111111
7	FE 0008	17	INC A	11111111	11111111
8	FE 0009	AB	MOV R3,A	11111111	11111111
9	FE 000A	17	INC A	11111111	11111111
A	FE 000B	AC	MOV R4,A	11111111	11111111
B	FE 000C	17	INC A	11111111	11111111
C	FE 000D	AD	MOV R5,A	11111111	11111111
D	FE 000E	17	INC A	11111111	11111111
E	FE 000F	AE	MOV R6,A	11111111	11111111
F	FE 0010	17	INC A	11111111	11111111
10	FE 0011	AF	MOV R7,A	11111111	11111111
11	FE 0012	17	INC A	11111111	11111111
12	FE 0013	A8	MOV R0,A	11111111	11111111
13	FE 0014	17	INC A	11111111	11111111
14	FE 0015	17	INC A	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5) ok

RESET 3A RB resetting (set a breakpoint at address 003A, run to get breakpoint control)

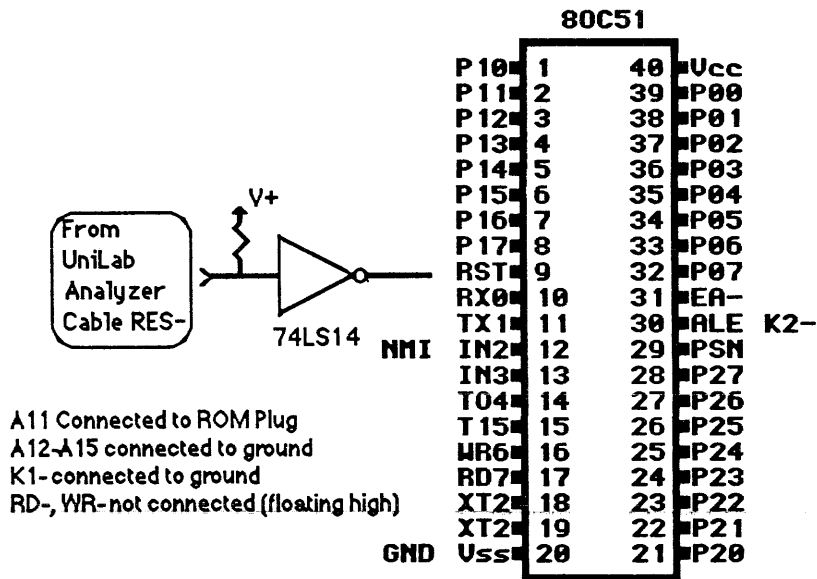
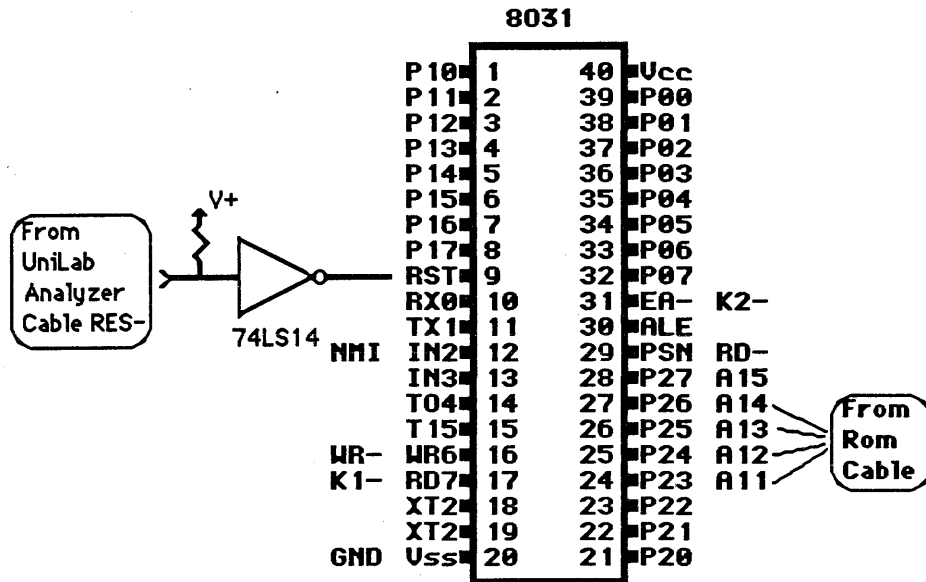
A=41 PSW=48 (cAfb) R0=1B R1=14 R2=15 R3=16 R4=17 R5=18 R6=19 R7=1A T=00
003A 17 INC A (next step) ok

8051 DISASSEMBLER/DEBUGGER (DDB-51) & (DDB-51P)

DDB-51 supports any of the following processors with separate ROM sockets (in expanded mode): 8051, 80C51, 8031, 8032, or 8052.

DDB-51P supports the OKI piggyback M80C51VS.

Both the piggyback and non-piggyback processors require the E analyzer cable.



Reset circuit

The reset circuit on the 8051 is non-standard (high-going rather than low-going), so the UniLab's open-collector active low RES- output on the analyzer cable will not work if connected directly. We suggest that you build the adapter circuit shown in the pinout diagram above or add the components to your prototype system. (The debugger will not work without a working automatic reset.)

The disassembler and hidden cycles

The 8051 disassembler automatically skips extra fetches that occur during instruction execution, and pre-fetches that occur before branches. If you want to see all the cycles, you can turn the disassembler off with **DASM'**.

Hidden cycles and the analyzer

Since the analyzer sees all cycles you should be careful about setting a trigger on the address immediately after a conditional jump. The UniLab will trigger on that address when it is pre-fetched, whether the branch occurs or not. Breakpoints at such an address, however, will be effective only if the instruction is actually executed by the microprocessor.

Reserved area

The 8051 debugger requires 8 reserved memory bytes (FFC0-FFC7) at the top of memory for saving and restoring interrupt enable status, and returning to the program. This memory space must be enabled.

Enter **ALSO FF00 EMENABLE** after the **EMENABLE** statements you would normally make for your system.

The size of the breakpoint

The breakpoint is set by patching a 3-byte LCALL instruction into your program. If your program tries to jump to or otherwise use either of the 2 bytes following the address of a breakpoint, there will be trouble.

Each time a new breakpoint is set, the previous breakpoint locations have their original contents restored.

Overlay area

The program memory locations above FFC0 are used for overlaying the debug routines. You can use them for programs, if necessary, as they are restored between debug steps, but you cannot use the debugger on them.

BY 4130
FF00 EMENABLE
FF00 EMENABLE

All sorts of memory

The 8051 family makes use of a somewhat confusing scheme of overlapping areas of memory. When you refer to address 83, for example, you could be trying to access internal RAM, external RAM, external ROM, or the special function registers. See the chart on the following page.

The UniLab software deals with this quirk by providing you with commands that "set the context." After you issue the command **EXTRAM**, the software knows that you are trying to access external RAM. The other "context" words are **INTRAM** and **PMEM**. For more information, read the discussion that follows, and the separate writeup on the DDB-51.

Internal registers

The 7 working registers and A, PSW, SP, DPTR, and IE are displayed automatically at the breakpoint.

Naming internal registers

If you are not changing register banks much, you can assign mnemonic names to R0-R7. For example, if you enter 6 **RNAME** **USER#**, the breakpoint display will print **USER#** instead of R6.

Pre-assigned names

The internal registers of the 8051 are assigned symbolic names automatically when the symbol table is turned on.

Altering internal registers

You can change any of the registers on the breakpoint display (except SP) while stopped at a breakpoint. Enter the desired contents followed by a space, then the register name preceded by an equal sign.

For example, 12 **=A** will put 12 in the A register.

Accessing internal memory

The internal memory locations can be written to by entering

<data> <adr> **R!**

and displayed with
<adr> **R?**

All of the named locations such as P0, IP, TH0, etc are defined as constants, so you can, for example, look at IP by entering **IP R?**. Enter 23 **P0 R!** to write 23 to port 0.

Accessing external memory

If you have external memory, you can access it with any of the commands used to read and write emulation memory such as **MDUMP**, **MFILL**, **MMOVE**, **M?**, **MM?**, **M!**, and **MM!**. They will go to external memory if the address isn't enabled for emulation.

Enter **EXTRAM** to access external RAM that occupies the same addresses as emulation ROM.

Access to internal RAM

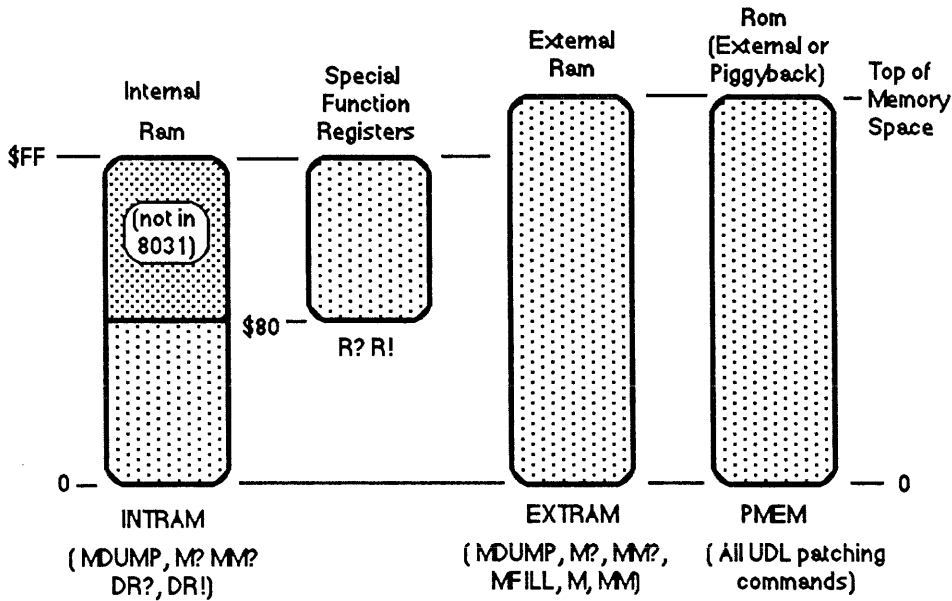
Enter **INTRAM** to use **MDUMP** on the internal RAM (addresses 0000 through 00FF).

PMEM resets to normal use of **MDUMP**.

You use <byte> <address> **DR!** to store a byte into internal RAM.

Type <address> **DR?** to display contents of internal RAM.

CHART OF 8051 FAMILY OVERLAPPING MEMORY MAP AND THE UniLab COMMANDS TO ACCESS EACH



Help Screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

n **R?** displays, m n **R!** writes ram 0-\$7F and special reg's \$80-\$FF.
 n **DR?** displays, m n **DR!** writes ram \$80-\$FF.
INTRAM allows mem read of internal RAM, 0-\$FF
PMEM for norm or **EXTRAM** for external ram read.
 n **RNAME** name assigns 10 char name to working reg n on bp display
 n **=A** , n **=IE** , n **=DPTR** , n **=R0** , n **=R1** , n **=R7** changes registers.
BPEX word executes word (macro) at breakpoint.

Trace and breakpoint display

LTARG (Load in sample program, set up emulation memory)

Emulator Memory Enable Status:

F =EMSEG

0 TO 7FF EMENABLE (enable rom area)

ALSO F800 TO FFFF EMENABLE (enable overlay area)

STARTUP resetting (reset target, show first cycles of operation)

cy#	CONT	ADR	DATA		HDATA	MISC
0	7F	0000	020030	LJMP 30	11111111	11111111
4	7F	0030	758168	MOV 81,#68	11111111	11111111 (set stack ptr)
8	7F	0033	7412	MOV A,#12	11111111	11111111
A	7F	0035	7834	MOV R0,#34	11111111	11111111
C	7F	0037	7956	MOV R1,#56	11111111	11111111
E	7F	0039	7A78	MOV R2,#78	11111111	11111111
10	7F	003B	7B9A	MOV R3,#9A	11111111	11111111
12	7F	003D	7C04	MOV R4,#4	11111111	11111111
14	7F	003F	04	INC A	11111111	11111111
16	7F	0040	04	INC A	11111111	11111111
18	7F	0041	04	INC A	11111111	11111111
1A	7F	0042	04	INC A	11111111	11111111
1C	7F	0043	04	INC A	11111111	11111111
1E	7F	0044	04	INC A	11111111	11111111
20	7F	0045	04	INC A	11111111	11111111
22	7F	0046	04	INC A	11111111	11111111
24	7F	0047	04	INC A	11111111	11111111
26	7F	0048	04	INC A	11111111	11111111
28	7F	0049	04	INC A	11111111	11111111
2A	7F	004A	04	INC A	11111111	11111111
2C	7F	004B	04	INC A	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5) ok

RESET 40 RB resetting (set a breakpoint at address \$0040, reset target and run until we break to get debug control)

PC= 40 A=13 PSW=01(cafbbv-P) R0=34 R1=56 R2=78 R3=9A R4=04 R5=03 R6=FF R7=FF
 DPTR= 0 SP=68 IE=60
 0040 04 INC A (next step) ok

8085 DISASSEMBLER/DEBUGGER (DDB-85)

This version supports the 8085 and 8080 processors.

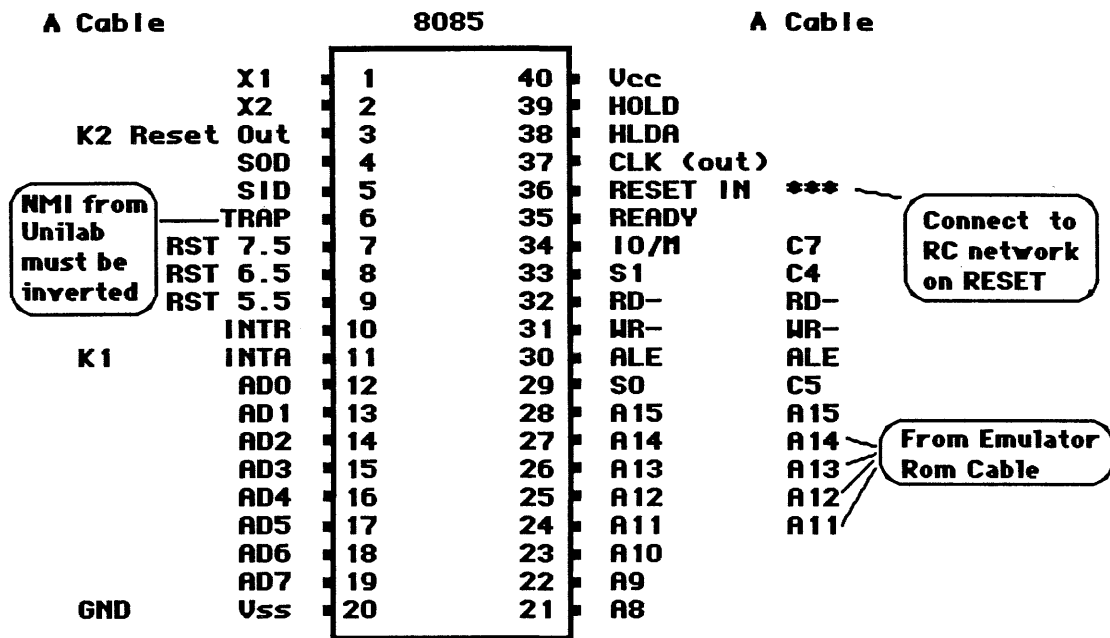
It works almost exactly the same as the Z80 package.

The 8085 requires the A analyzer cable.

The 8080 requires the H analyzer cable.

Patch word

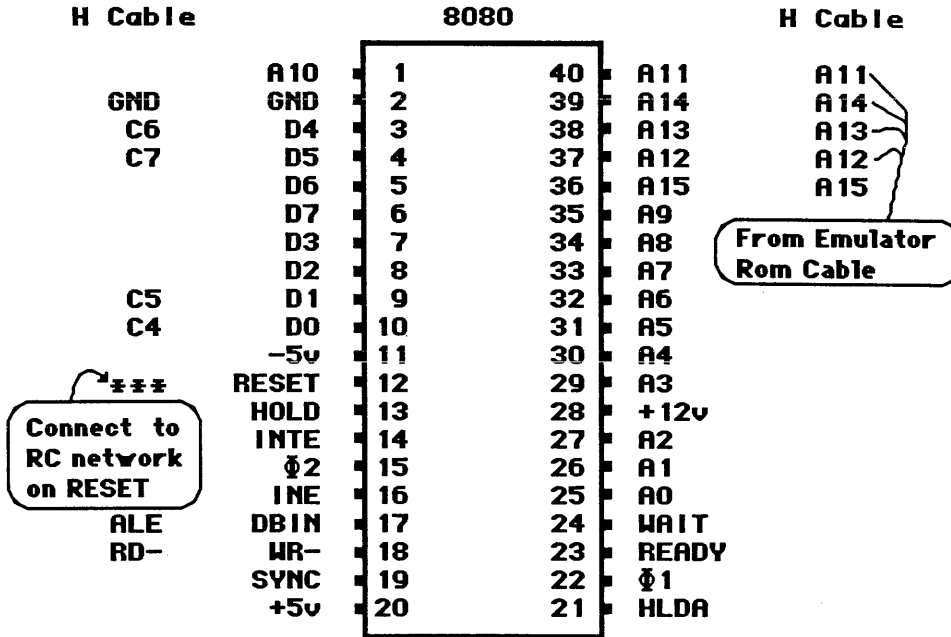
For operation with the 8080, you enter **8080PATCH** to patch the program, then **SAVE-SYS** to save the newly configured software.



NMI note

The **NMI** wire from the UniLab must be feed through an inverting circuit before connecting to the **TRAP** pin of the 8085.

The 8080 does not have a **TRAP** pin. So the NMI features will not work for the 8080 (**NMI, SSTEP, RI, SI**).



About "..."

Note that since the second byte is not fetched on conditional jumps that don't actually jump, you will see a ... on the disassembled display in place of the destination address.

Reserved area

Bytes eight through eleven are reserved. The overlay is above there.

Reserved resources

In additon, 3 bytes of RAM at addresses 24-26 are used for the 8085 NMI vector.

Help screen

Press CTRL-F3 to get the help screen that includes this target-specific information:

m n **OUT** writes m to port n. n **INP** reads port n.
EINT reenables target interrupts after bp, **DINT** leaves disabled
 register change: n =**AF** n =**BC** n =**DE** n =**HL**
 See also **HD** . (locations 8-11 reserved, overlay above.)

Trace and breakpoint display

LTARG (set up default emulation memory, load sample program)

Emulator Memory Enable Status:

F =EMSEG

0 TO 7FF EMENABLE

ok

STARTUP resetting (issue reset to target, show first cycles upon start up)

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
0	7F	0000	C38000	JMP 80	11111111	11111111
3	7F	0080	31001A	LXI SP,1A00	11111111	11111111
6	7F	0083	3E12	MVI A,12	11111111	11111111
8	7F	0085	015634	LXI B,3456	11111111	11111111
B	7F	0088	119A78	LXI D,789A	11111111	11111111
E	7F	008B	21DEBC	LXI H,BCDE	11111111	11111111
11	7F	008E	C5	PUSH B	11111111	11111111
12	6F	19FF	34	write	11111111	11111111
13	6F	19FE	56	write	11111111	11111111
14	7F	008F	C1	POP B	11111111	11111111
15	5F	19FE	56	read	11111111	11111111
16	5F	19FF	34	read	11111111	11111111
17	7F	0090	3C	INR A	11111111	11111111
18	7F	0091	3C	INR A	11111111	11111111
19	7F	0092	3C	INR A	11111111	11111111
1A	7F	0093	3C	INR A	11111111	11111111
1B	7F	0094	3C	INR A	11111111	11111111
1C	7F	0095	3C	INR A	11111111	11111111
1D	7F	0096	3C	INR A	11111111	11111111
1E	7F	0097	3C	INR A	11111111	11111111
1F	7F	0098	3C	INR A	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5) ok

RESET 8F RB resetting (set a breakpoint at 008F, reset target, run to breakpoint)

AF=1214 (sz-A-Pnc) BC=3456 DE=789A HL=BCDE SP=8795
008F C1 POP B (next step) ok

DISASSEMBLER/DEBUGGER (DDB-86) & (DDB-88)

DDB-86 supports the 8086, 80186, and 80286 processors.

DDB-88 supports the 8088 and 80188 processors.

The 8086/88 in min mode requires the A analyzer cable.

The 8086/88 in max mode requires the L analyzer cable.

The 80186/188 requires the A analyzer cable.

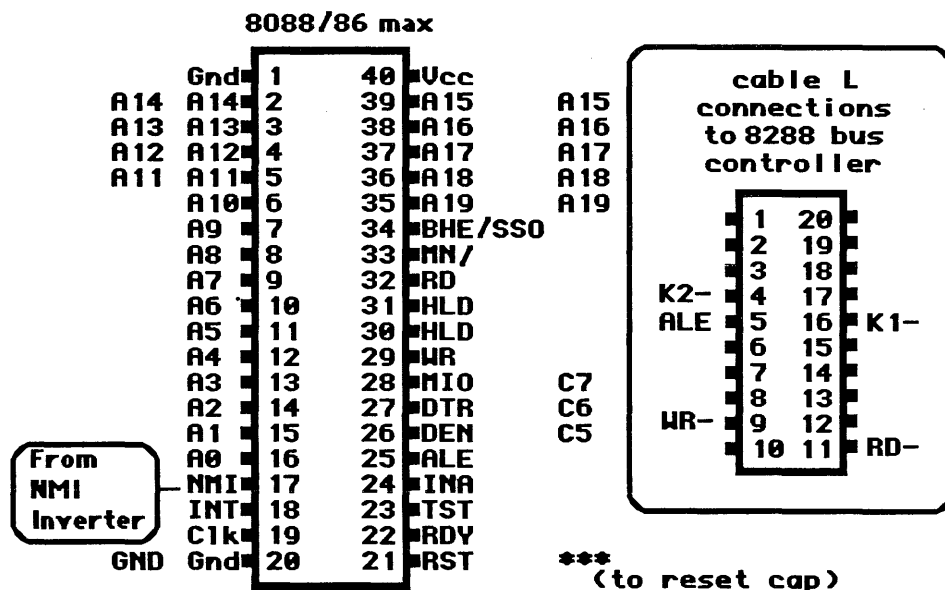
The 80286 requires the I analyzer cable.

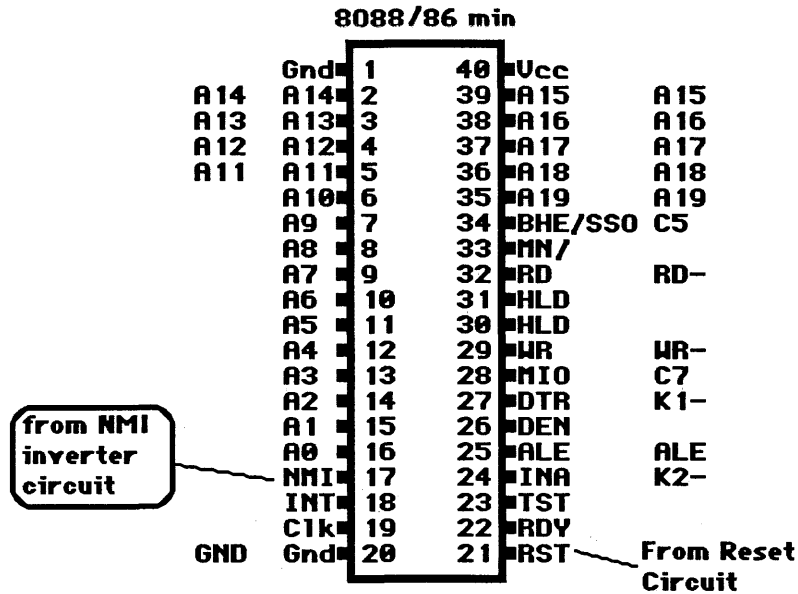
Patch words

To configure DDB-86 properly you must first enter one of the following: **86MIN**, **186PATCH**, or **286PATCH**, depending upon which processor you will be using.

To configure DDB-88 properly, you must initially enter one of the following commands: **88MIN**, **88MAX**, or **188PATCH**.

. Then you must use **SAVE-SYS** to save the newly configured software.





Special note on 86MIN

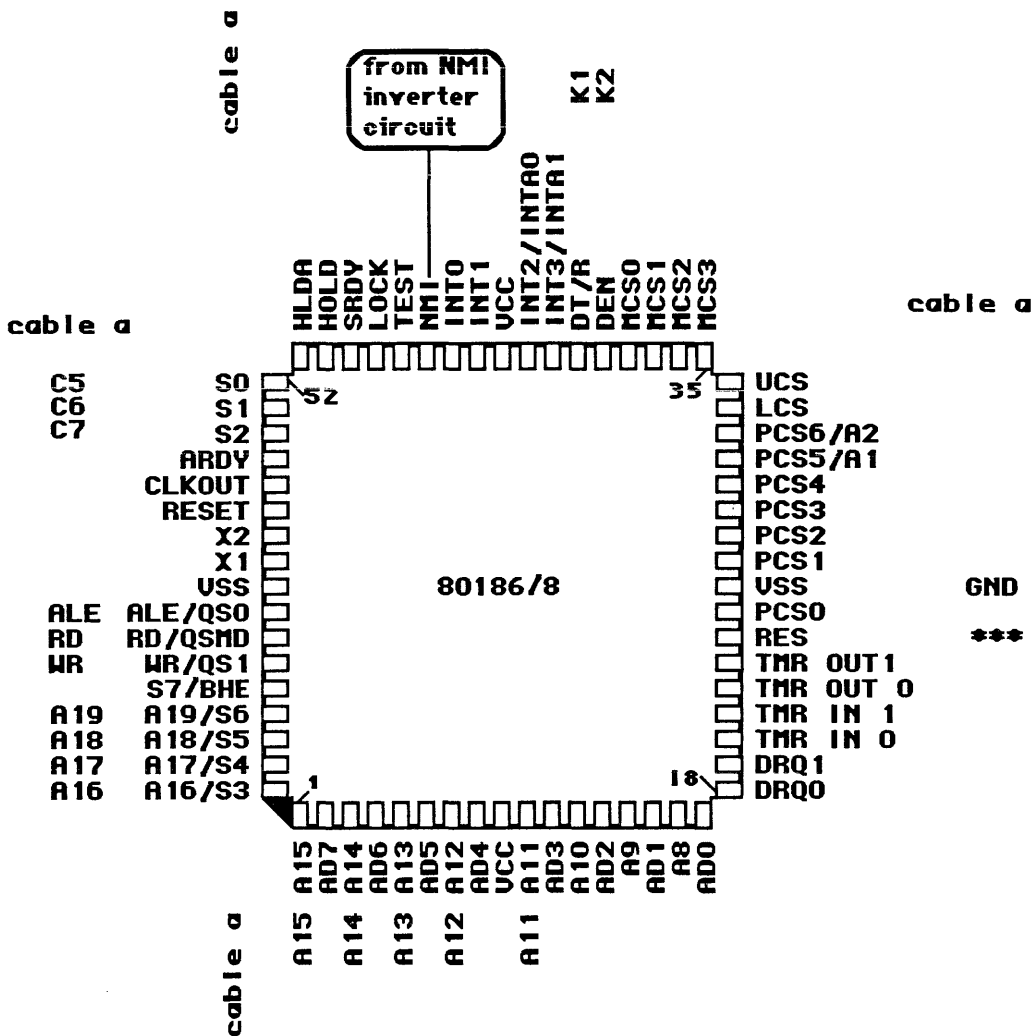
The S0 signal is not brought out, so address bits A16-19 are used to identify fetch cycles. The code is assumed to be in segment Fxxxx. You can include a larger memory area by changing system constants **FMASK** and **F=**, which are now set to CF and 8F respectively.

For example, to allow code fetches in addresses Exxxx to Fxxxx, change **FMASK** to CE by entering **CE ' FMASK !**. (FMASK is ANDed with the CONT inputs and the result is tested for equality to **F=** to determine if a cycle is a fetch cycle.)

NMI connection

The NMI input on the 8088 and 8086 family members, requires a positive - going transition, the opposite of most processors. You cannot connect the UniLab's negative-going NMI output directly to the NMI pin of the 8088 or 8086 (pin 17).

Instead, you must put the signal through an inverting circuit first.



Connections

The DIP-clip is not practical for the 80188, 80186, and 80286 packages. Normally this is not a problem, however, since you can pick up the required signals by placing DIP clips on the bus controller and address latch ICs.

Disassembler

Data memory reads and writes

The 8088/6 disassemblers automatically unscramble the bus cycles into two separate streams: if a data memory read or write occurs during the decoding of an instruction its display is deferred until the instruction's disassembly is complete.

The fetch and non-fetch cycles are thus treated as two separate streams. The read and write cycles that result from an instruction will usually appear on the trace display one or two instructions later due to prefetching.

You can cause the disassembler to group data memory access cycles with the commands that cause them, by using the command **ALIGN**. Turn this back off with **ALIGN'**. Or, use the Mode panel to toggle this option off and on (**F8**).

The disassembler and hidden cycles

Extra prefetches that are not executed due to a branch are automatically thrown away by the disassembler. You can look at the true sequence of cycles by turning the disassembler off with **DASM'**.

Hidden cycles and the analyzer

Don't try to trigger the analyzer on the 2 bytes immediately after a conditional jump. They will be pre-fetched even if they aren't executed. (Breakpoints in these bytes are OK however.)

Special MISC display

The MISC display column of the disassembled trace display shows the state of the MISC inputs during the first cycle of the disassembled instruction. (Most other UniLab disassemblers show the MISC bits during the last cycle of the disassembled instruction.)

Debugger

A demo program is included with your system so that you can familiarize yourself with the debug commands while working on a program that is known to work. To load this program just enter **LTARG**. If you then enter **STARTUP** you will see a trace of the program, which simply initializes the stack pointer to 100, installs the breakpoint vector, then sets all registers to known values. If you then enter **RESET F842 RB** you will see a breakpoint display.

Reserved Area

The reserved area is at FFFB1 and B2. The overlay area is above.

Overlay area

The locations above FFFB2 are used as an overlay area but are restored between operations, so you can use them but the debugger may not work on them. Note that this top end of memory must be emulated for the debugger to work.

Required code in user program

The INT3 instruction is used as a breakpoint code so your program must install the correct pointer at locations 0000C-0000F. Your program must install the vector at 00004-7 if you want to be able to use the single step function. The vector at 00008-B is used for the **NMI** feature.

Most programs will initialize the interrupt pointers from an array in ROM, or you can include this code:

```
C7060400B1FF      MOV 04H,0FFB1H
C706060000F0      MOV 06H,0F000H
C7060800B1FF      MOV 08H,0FFB1H
C7060A0000F0      MOV 0AH,0F000H
C7060C00B1FF      MOV 0CH,0FFB1H
C7060E0000F0      MOV 0EH,0F000H
```

If your target program doesn't initialize these locations, you can run the demo program LTARG to initialize them, then load in and run your program.

Segments and addresses

All operations that require an address expect a 16-bit offset address from the "current user segment."

You can change the current segment with the **CS:** or **DS:** or **ES:** or **SS:**. The segment will remain selected until another segment is selected. Other commands allow you to change the value of the segments.

Note that the absolute address is calculated by multiplying the current segment by 10 (hex) and then adding the value of the offset.

Use **SEGS** to find the current value of all the segments, and the current segment.

You can put the segment selection on the same line as the command. For example:

```
CS: 1234 100 MDUMP
```

Consult the Disassembler/Debugger writeup for more details.

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

```
Register change: n =F  n =AX  n =BX  n =CX  n =DX
                  n =BP  n =SI  n =DI  n =CS  n =DS  n =ES  n =IP
CS: DS: ES: SS: select segments for M operations.
b p OUTB  w p OUTW  p INB  p INW  for port I/O
see also HD  FFFB1-FFFB2 reserved, overlay above.
```

Trace and breakpoint display

8088 Trace display

A>UL88 (User entry into program... Note user words entered at terminal are underlined.)

UniLab
II
Version 3.00

Copyright 1986
Orion Instruments
Redwood City, CA

8088 disassembler installed - with debugger. (Note log on display)

LTARG (load built-in sample program, set up emulation area)

STARTUP resetting (Reset target, show first cycles of operation)

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
0	9F	CS:07F0	EA000080FF	JMP 0,FF80:	11111111	11111111
6	9F	CS:0000	B80000	MOV AX,0	11111111	11111111
9	9F	CS:0003	8ED0	MOV SS,AX	11111111	11111111
B	9F	CS:0005	BC0001	MOV SP,100	11111111	11111111
E	9F	CS:0008	C7060C00B1FF	MOV [C],FFB1	11111111	11111111
15	D0	DS:000C	B1	write interrupt #3	11111111	11111111
16	D0	DS:000D	FF	write interrupt #3	11111111	11111111
14	9F	CS:000E	C7060E0000F0	MOV [E],F000	11111111	11111111
1D	D0	DS:000E	00	write interrupt #3	11111111	11111111
1E	D0	DS:000F	F0	write interrupt #3	11111111	11111111
1C	9F	CS:0014	C7060800B1FF	MOV [8],FFB1	11111111	11111111
25	D0	DS:0008	B1	write interrupt #2	11111111	11111111
26	D0	DS:0009	FF	write interrupt #2	11111111	11111111
24	9F	CS:001A	C7060A0000F0	MOV [A],F000	11111111	11111111
2D	D0	DS:000A	00	write interrupt #2	11111111	11111111
2E	D0	DS:000B	F0	write interrupt #2	11111111	11111111
2C	9F	CS:0020	C7060400B1FF	MOV [4],FFB1	11111111	11111111
35	D0	DS:0004	B1	write interrupt #1	11111111	11111111
36	D0	DS:0005	FF	write interrupt #1	11111111	11111111
34	9F	CS:0026	C706060000F0	MOV [6],F000	11111111	11111111
3D	D0	DS:0006	00	write interrupt #1	11111111	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)

RESET 50 RB resetting (set a breakpoint at address 50)

IP=0050 F=F002(---oditsz-a-p-c) AX=1237 BX=5678 CX=9ABC DX=DEF0
CS=FF80 DS=3000 SS=0000 ES=2000 SP=00FE BP=1111 DI=3333 SI=2222
CS:0050 40 INC AX (next step) ok

8086 Trace display

A><u>UL86</u> (Hook up 8086 target system to show trace differences from 8088...)

UniLab	Copyright 1986
II	Orion Instruments
Version 3.00	Redwood City, CA

8086 disassembler installed - with debugger. (Note log on display)

LTARG (load built-in sample program, set up emulation area)

STARTUP resetting (Reset target, show first cycles of 8086 operation)
(from top of buffer)

cy#	CONT	ADR	DATA	MISC
0	BF	CS:07F0	EA000080FF JMP 0,FF80:	11111111
4	BF	CS:0000	B80000 MOV AX,0	11111111
5	BF	CS:0003	8ED0 MOV SS,AX	11111111
6	BF	CS:0005	BC0001 MOV SP,100	11111111
8	BF	CS:0008	C7060C00B1FF MOV [C],FFB1	11111111
D	F0	DS:000C	FFB1 write interrupt #3	11111111
B	BF	CS:000E	C7060E0000F0 MOV [E],F000	11111111
11	F0	DS:000E	F000 write interrupt #3	11111111
F	BF	CS:0014	C7060800B1FF MOV [8],FFB1	11111111
15	F0	DS:0008	FFB1 write interrupt #2	11111111
13	BF	CS:001A	C7060A0000F0 MOV [A],F000	11111111
19	F0	DS:000A	F000 write interrupt #2	11111111
17	BF	CS:0020	C7060400B1FF MOV [4],FFB1	11111111
1D	F0	DS:0004	FFB1 write interrupt #1	11111111
1B	BF	CS:0026	C706060000F0 MOV [6],F000	11111111
21	F0	DS:0006	F000 write interrupt #1	11111111
1F	BF	CS:002C	B80020 MOV AX,2000	11111111
20	BF	CS:002F	8EC0 MOV ES,AX	11111111
22	BF	CS:0031	B80030 MOV AX,3000	11111111
24	BF	CS:0034	8ED8 MOV DS,AX	11111111
25	BF	CS:0036	B83412 MOV AX,1234	11111111
Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)				

RESET 50 RB resetting (Reset target, set breakpoint at 50 in code segment)

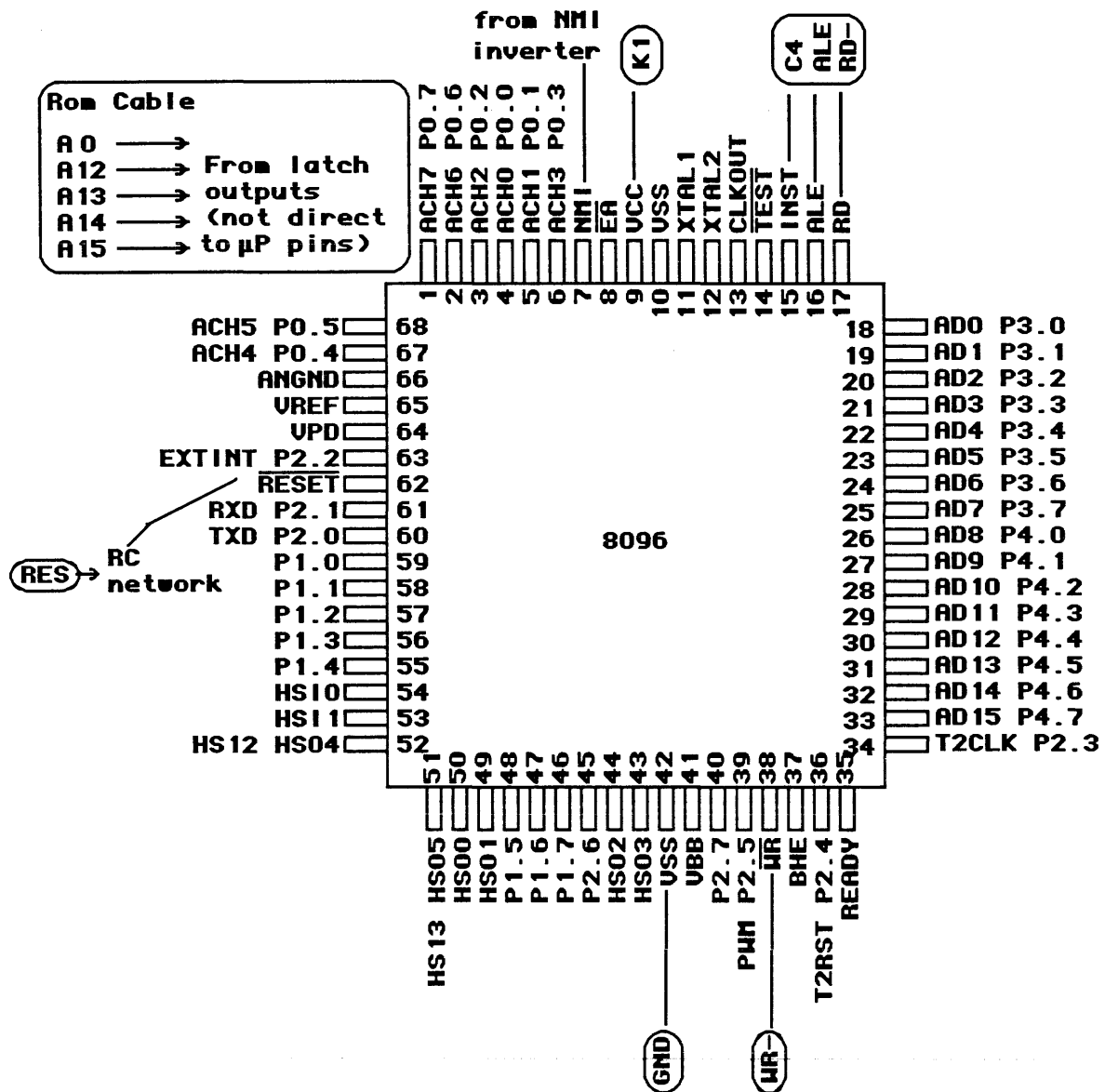
IP=0050 F=F002(---oditsz-a-p-c) AX=1237 BX=5678 CX=9ABC DX=DEF0
 CS=FF80 DS=3000 SS=0000 ES=2000 SP=0100 BP=1111 DI=3333 SI=2222
 CS:0050 40 INC AX (next step) ok

8096 DISASSEMBLER/DEBUGGER (DDB-96)

This version supports the ROM-less versions of the MCS-96 family of Intel processors. We fully support the 64 pin packages, the 8096 and the 8097.

We cannot provide the **NMI** command nor related features on the 48 pin packages, the 8094 and 8095, since those do not have an NMI pin.

All members of the MCS-96 family require the R analyzer cable.



The MCS-96 family:

	I/O		ROMLESS		ROM
48	PIN	d	8094		8394
		d&a	8095		8395

64	PIN	d	8096		8396
		d&a	8097		8397

Caution

SSTEP is only appropriate for following branches. Otherwise it tends to "double-step."

Use **N** to single-step on instructions other than jumps-- or to "fall through" jumps.

Reserved resources

Note that INTEL reserves 2012H through 2079H for "Factory test code." And they reserve NMI for use only by their development system. NMI jumps to location 00.

Since the internal RAM cannot be used for program code, any fetches from the address range 00 to 255 go to external memory. INTEL reserves use of that range for their development code.

The Orion Debugger uses all the above resources anyway.

We reserve locations 2016H to 201CH for our debugger vector. The overlay area-- the place we put the routines that read your processor's internal state-- takes up 2016H and above.

You can change the starting address of overlay area with <addr> **=OVERLAY**. But right now it occupies the hole in the memory map that INTEL uses for factory test code, so you are advised to not move the overlay area.

The Orion debugger also uses register 50H. You can change which register is used with the command <adr> **=DBG**.

Internal registers

The named registers of the 8096 family (AX, BX, CX, and DX) are symbolic names assigned to an arbitrary eight byte region of the register file. The debugger assumes that your program uses the region 1CH through 23H.

The **IS-xx** commands can reassign the names somewhere else. For example, **44 IS-AX** would tell the Orion debugger that your target program uses register 44 (hex) as register AX.

See the separate document on the 8096 debugger/disassembler for more details.

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

Register change: n **=F** n **=AX** n **=BX** n **=CX** n **=DX**
Store to any register: n reg# **R!** Display any register. reg# **R?**
Change register assignment: n **IS-AX** n **IS-BX** n **IS-CX** n **IS-DX**

Display special function register names: **SHOW-REGS**
Assign a symbolic name to a register: n **RNAME** name
Show - don't show reg. names: **SHOW-NAMES** / **HIDE-NAMES**
Adjust - don't adjust memory cycles. **ALIGN** / **ALIGN'**
See also **HD**. 2016-201c reserved, overlay above.

Trace and breakpoint display

LTARG (load built in sample program, set up default emulation memory area)

Emulator Memory Enable Status:

F =EMSEG

0 TO 2FFF EMENABLE

ok

STARTUP (Reset target, capture first cycles of operation)

resetting

(from top of buffer)

cy#	CONT	ADR	DATA		MISC
0	FF	2080	A1004118	LD SP,#4100	11111111
2	FF	2084	A100A01C	LD AX,#A000	11111111
4	FF	2088	A100B01E	LD BX,#B000	11111111
6	FF	208C	A100C020	LD CX,#C000	11111111
8	FF	2090	A100D022	LD DX,#D000	11111111
A	FF	2094	CA1C	PUSH [AX]	11111111
C	EF	A000	A000	read	11111111
D	CF	40FE	A000	write	11111111
B	FF	2096	CE1C	POP [AX]	11111111
F	EF	40FE	A000	read	11111111
10	CF	A000	A000	write	11111111
E	FF	2098	071C	INC AX	11111111
11	FF	209A	071C	INC AX	11111111
12	FF	209C	071C	INC AX	11111111
13	FF	209E	071C	INC AX	11111111
14	FF	20A0	071C	INC AX	11111111
15	FF	20A2	071C	INC AX	11111111
16	FF	20A4	071C	INC AX	11111111
17	FF	20A6	071C	INC AX	11111111
18	FF	20A8	071C	INC AX	11111111
19	FF	20AA	071C	INC AX	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)

RESET 209E RB resetting (enable reset, run to address 209E and break)

SP = 4100 AX = A003 BX = B000 CX = C000 DX = D000

FLAGS = 4000 (zNvtc.isrmmmmmmmm)

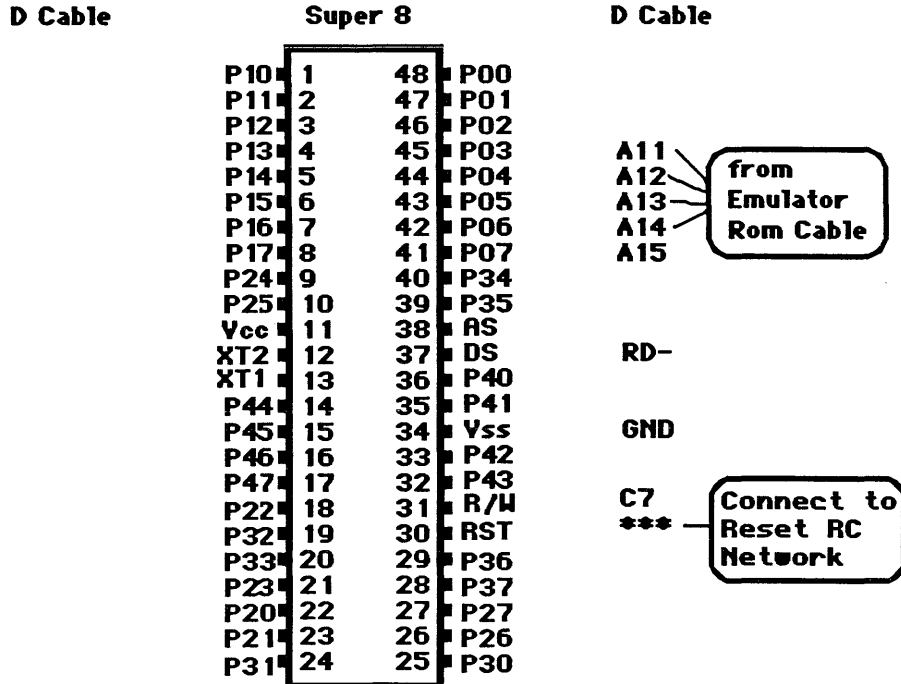
209E 071C INC AX (next step) ok

SUPER 8 DISASSEMBLER/DEBUGGER (DDB-S8)

This version supports the Super 8 family, including all ROMless and piggyback chips. As of July, 1986, Zilog has released one 48-pin ROMless chip, the 8800, and one prototype chip, the 8822.

The Super 8 requires the D cable.

Cable D Connections to Super 8



Reserved area

The area from 0781 to 0785 is reserved. The overlay starts at 0786. You can alter the location of reserved and overlay areas with `<adr> =OVERLAY`.

Reserved resources

The debugger also requires 3 bytes of RAM from addresses 24 to 26. You can alter the location of that area with `<adr> =PTR`.

Accessing external RAM

After you have established debug control (see **RB** or **NMI** in the **Command Reference** chapter) you can access internal RAM with any of the commands used to read and write emulation memory such as **MDUMP**, **MFILL**, **MMOVE**, **M?**, **MM?**, **M!**, and **MM!**. These commands automatically go to external RAM if the address isn't enabled for emulation.

Enter **EXTRAM** to access external RAM that occupies the same addresses as emulation ROM. You use the command **EXTROM** when you want to switch back to accessing emulation ROM at those addresses.

Help screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

n R? displays register 0-FF. **m n R!** writes m to register n
n DR? displays data reg C0-FF. **m n DR!** writes m to data reg n
n RNAME name assigns 10 char name to working reg n on bp
display
=PTR defines 3 bytes ram used by debug. Must load before bkpt.
EXTRAM or **EXTROM** select target ram/rom read mode.
(locations 0781-0785 reserved, overlay above there)

Trace and breakpoint display

STARTUP resetting (reset target, show trace of startup routine)

cy#	CONT	ADR	DATA		HDATA	MISC
-2	FF	0020	E66007	LD 60,#7	11111111	11111111 (load ptr with address
1	FF	0023	E66181	LD 61,#81	11111111	11111111 of overlay...)
4	FF	0026	C6D80000	LDW D8,#0	11111111	11111111
8	FF	002A	C6C01234	LDW C0,#1234	11111111	11111111
C	FF	002E	C6C25678	LDW C2,#5678	11111111	11111111
10	FF	0032	C6C4ABCD	LDW C4,#ABCD	11111111	11111111
14	FF	0036	C6C6EF12	LDW C6,#EF12	11111111	11111111
18	FF	003A	C6C80123	LDW C8,#123	11111111	11111111
1C	FF	003E	C6CA4567	LDW CA,#4567	11111111	11111111
20	FF	0042	C6CC89AB	LDW CC,#89AB	11111111	11111111
24	FF	0046	C6CECDEF	LDW CE,#CDEF	11111111	11111111
28	FF	004A	EE	INC RE	11111111	11111111
29	FF	004B	EE	INC RE	11111111	11111111
2A	FF	004C	EE	INC RE	11111111	11111111
2B	FF	004D	EE	INC RE	11111111	11111111
2C	FF	004E	EE	INC RE	11111111	11111111
2D	FF	004F	EE	INC RE	11111111	11111111
2E	FF	0050	EE	INC RE	11111111	11111111
2F	FF	0051	EE	INC RE	11111111	11111111
30	FF	0052	EE	INC RE	11111111	11111111
31	FF	0053	EE	INC RE	11111111	11111111

PgDn Home (top) n TN (from step n) T (from n=-2)

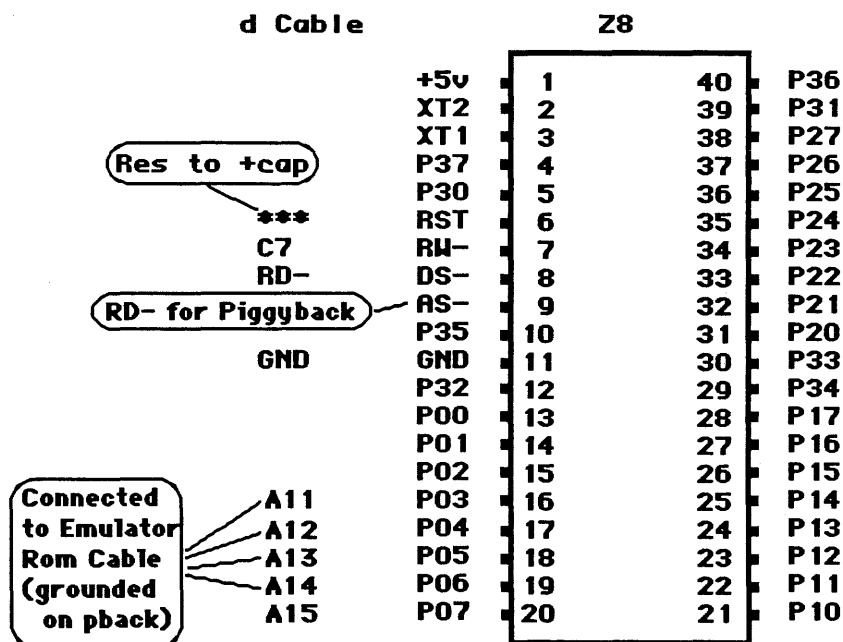
RESET 50 RB resetting (set a breakpoint at address 50 and run to get debug control)

PC=0050 FLAGS=A0 (CzSodhfb) RP0=C0 RP1=C8 SP=0000 IMR=6E
R0=12 R1=34 R2=56 R3=78 R4=AB R5=CD R6=EF R7=12
R8=01 R9=23 RA=45 RB=67 RC=89 RD=AB RE=D3 RF=EF
0050 EE INC RE (next step) ok

Z-8 DISASSEMBLER/DEBUGGER (DDB-Z8)

This version supports the Z-8 with separate ROM or the piggyback version.

The Z8 requires the D analyzer cable.



Note: The above diagram is for a separate ROM configuration. See Appendix C for piggyback wiring.

The disassembler and hidden cycles

The Z-8 disassembler automatically skips extra fetches that occur during instruction execution, and pre-fetches that occur before branches. You can look at these cycles by turning the disassembler off with **DASM'**.

Special STARTUP

Since the Z-8 does repeated fetches during reset that are not actually executed, the **STARTUP** command is redefined to look for address D immediately after C.

If you want to look at the actual first cycles after reset, enter **RESET NORMT S** .

Reserved resources

The debugger requires one register pair and 5 reserved memory locations for breakpoints. The register pair must be specified by entering <value> **=PTR** where the value is the address of the lower numbered register in hex, and is between 3 and 80.

You can make this assignment permanent by entering **SAVE-SYS**.

Required code in user program

Normally the debugger uses a 2-byte interrupt by assuming that your program has loaded the address of the Orion breakpoint routine into the pointer register pair. You must add code to the beginning of your program to initialize the register pair that you choose with **=PTR**.

For example, since the breakpoint address is 7AF, if you have chosen register 60 as your pointer (by entering **60 =PTR**), your program must include the code:

```
LD 60,#7
LD 61,#AF
```

While installation of the pointer is a bit of a nuisance, it reduces the length of the breakpoint instruction to 2 bytes. This greatly reduces the possibility of weird effects due to your program trying to use the third byte of a breakpoint, say by jumping to the start of the instruction that was there before you set a breakpoint.

Of course when you exit from the debugger, the code that was overwritten by the breakpoint code is restored.

As it is you must not place breakpoints on 1-byte instructions if the program might jump to the instruction following.

Internal registers

The 16 working registers are displayed automatically at the breakpoint. If you are not changing the register pointer much, you can assign mnemonic names. For example, if you enter **4 RNAME USER#**, the breakpoint display and disassembler will print USER# instead of R4.

Once you have established debug control, any internal register can be altered by entering <data> <address> **R!** or examined by entering <address> **R?**. For example, entering **12 34 R!** will write 12 into register 34. (Register 34 will be called R4 on the debug display if the bank of working registers starts at 30.)

To look at the same register you just enter **34 R?**. Note that the Zilog-defined register names can be used in place of numbers. For example **SIO R?** will display the SIO register, and **50 SIO R!** will write 50 to it.

Access to external RAM

If you have external memory, you can access it with any of the commands used to read and write emulation memory, such as **MDUMP**, **MFILL**, **MMOVE**, **M?**, **MM?**, **M!**, and **MM!**. They will go to external memory if the address isn't enabled for emulation.

If you have external memory which uses the same addresses as emulated program memory, you can fool the UniLab by setting a bit which is not decoded by the hardware. For example, if your hardware does not look at A15 (while the UniLab's emulation ROM does) use 8000 to address external location 0.

You can also use **PMEM** or **EXTRAM** to select which memory you mean. This will patch the debugger to use LDC or LDE instructions for looking at non-emulated memory.

Address lines and the UniLab

Remember that if you have a ROM that is external to the Z-8, the most significant 8 bits of the Z-8 address output float until enabled by the program. You must have pulldown or pullup resistors on these signals so that their initial state is defined.

If you have used pulldown resistors that are greater than 3K, you will have to replace U14 and U15 in the UniLab with 74HCT373's so that input loading won't cause indeterminate levels. Orion will supply these chips free of charge if you need them (the chips are socketed).

If you have pullup resistors, you can ensure proper startup by entering **ALSO FFOC EMENABLE C 15 FFOC MMOVE** to put a copy of the start of your program up at address FFOC.

HELP Screen

Press **CTR-F3** to get the help screen that includes this target-specific information:

n R? displays register 0-FF. **m n R!** writes m to register n
n RNAME name assigns 10 char name to working reg n on bp display
=PTR defines bp pointer register pair. Must load before bkpt.
EXTRAM or **PMEM** select ram mode. use 8nnn to access location nnn.
EXTDATA or **INTDATA** for stack in ext or int ram
PBACK for piggyback chip, **PBACK'** for regular.
(locations 7AF -7B3 reserved, overlay above there)

Trace and breakpoint display

LTARG (set up default emulation memory, load built-in sample program..)

Emulator Memory Enable Status:

F =EMSEG

0 TO FFF EMENABLE

STARTUP resetting (issue reset to target and capture trace of first bus cycles)

(from top of buffer)

cy#	CONT	ADR	DATA		HDATA	MISC
-2	FF	000C	E60000	LD 0,#0	11111111	11111111
1	FF	000F	E6F896	LD F8,#96	11111111	11111111
4	FF	0012	E66007	LD 60,#7	11111111	11111111
7	FF	0015	E661AF	LD 61,#AF	11111111	11111111
A	FF	0018	E6FD10	LD FD,#10	11111111	11111111
D	FF	001B	E6FE00	LD FE,#0	11111111	11111111
10	FF	001E	E6FF40	LD FF,#40	11111111	11111111
13	FF	0021	E6FC00	LD FC,#0	11111111	11111111
16	FF	0024	0C01	LD R0,#1	11111111	11111111
18	FF	0026	1C23	LD R1,#23	11111111	11111111
1A	FF	0028	2C45	LD R2,#45	11111111	11111111
1C	FF	002A	3C67	LD R3,#67	11111111	11111111
1E	FF	002C	4C89	LD R4,#89	11111111	11111111
20	FF	002E	5CAB	LD R5,#AB	11111111	11111111
22	FF	0030	BC12	LD RB,#12	11111111	11111111
24	FF	0032	CC34	LD RC,#34	11111111	11111111
26	FF	0034	DC56	LD RD,#56	11111111	11111111
28	FF	0036	EC78	LD RE,#78	11111111	11111111
2A	FF	0038	FC9A	LD RF,#9A	11111111	11111111
2C	FF	003A	EE	INC RE	11111111	11111111
2D	FF	003B	EE	INC RE	11111111	11111111

PgDn Home (top) n TN (from step n) T (from n=-5)

RESET 3A RB resetting (set breakpoint at address 003A and run to get debug control)

PC=003A FLAGS=00(czsodhQR) RP=10 SP=40 IMR=3F
 R0=01 R1=23 R2=45 R3=67 R4=89 R5=AB R6=00 R7=0E
 R8=4C R9=00 RA=FF RB=12 RC=34 RD=56 RE=78 RF=9A
 003A EE INC RE (next step) ok

Z-80 DISASSEMBLER/DEBUGGER (DDB-Z80)

This version supports the Z-80, the Hitachi HD64180 and the NSC-800 microprocessors.

The Z80 and HD64180 processors require the E analyzer cable.

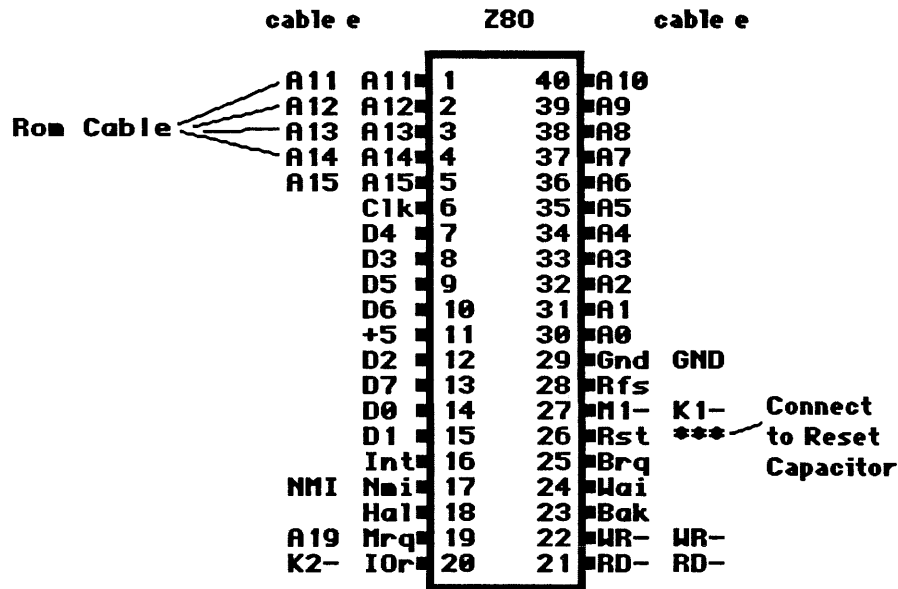
The NSC-800 requires the Q analyzer cable.

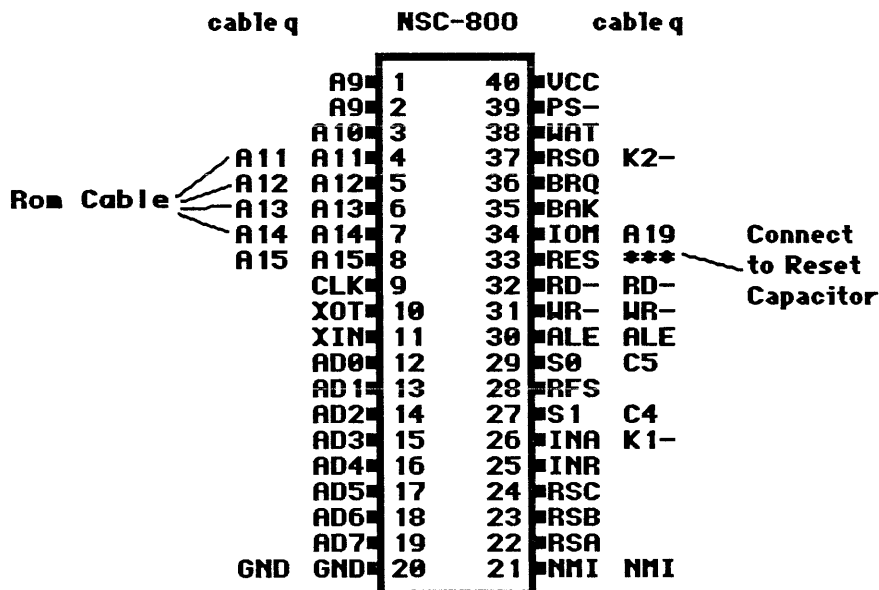
Patch Words

For operation with the HD64180, you enter **HD64180** to patch the program.

For the NSC800, you enter **NSC-800**.

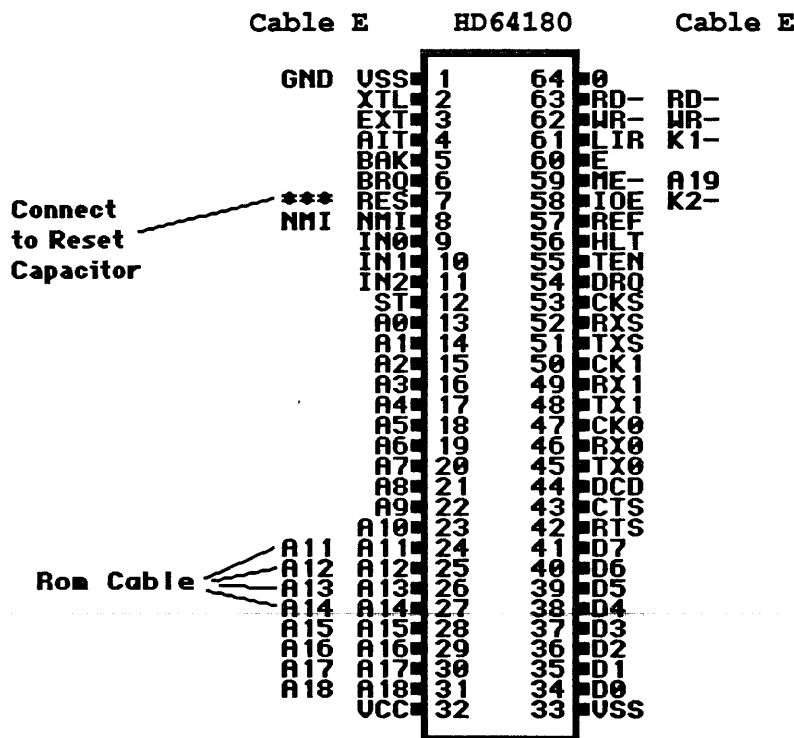
Then **SAVE-SYS** to save the newly configured software.





The HD64180's 64-pin package presents some difficulty in making the cable connections. Until 64-pin DIP-clips become available, the connections to the HD64180 will have to be made in other ways.

You can connect wires to wire-wrap posts if the target is a prototype board, or to a special bus strip of wire-wrap posts which are connected to the appropriate pins of the HD64180. The signals can also be picked up with jumpers at electrically equivalent points in the circuit.

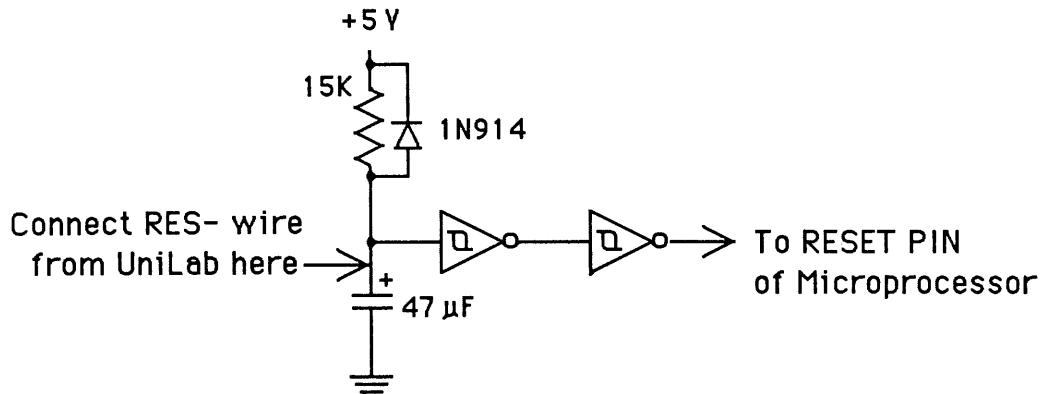


Reset Circuit

Unless you include logic gates in the reset circuit, your processor might be subject to "multiple resets" each time you try to start it up. We recommend that you include two Schmitt triggers in your power-on reset circuit, as in the example below.

Reserved Area

The reserved area is from 38 to 3D. The overlay starts at 3D.



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

Help Screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

m n **OUT** writes m to port n. n **INP** reads port n.
EINT reenables target interrupts after bp, **DINT** leaves disabled register change: n =**AF** n =**BC** n =**DE** n =**HL** n =**IX** n =**IY**
see also **HD** (locations 38-3D reserved, overlay starts at 3D)

Trace and breakpoint display

LTARG (Load sample program and set up
 Emulator Memory Enable Status a typical emulation memory
 area)
 7 =EMSEG (Again, LTARG establishes a MSEG and
 0 TO 7FF EMENABLE area for emulation rom)
 ok (operating system response, telling us that command was executed,
 now it's waiting for next command)

STARTUP resetting (Reset target system and show startup traces.)

cy#	CONT	ADR	DATA		HDATA	MISC	
0	B7	0000	310019	LD SP,1900	11111111	11111111	(Patch this to agree
3	B7	0003	3E12	LD A,12	11111111	11111111	with your ram
5	B7	0005	015634	LD BC,3456	11111111	11111111	area....)
8	B7	0008	119A78	LD DE,789A	11111111	11111111	
B	B7	000B	21DEBC	LD HL,BCDE	11111111	11111111	
E	B7	000E	C5	PUSH BC	11111111	11111111	
F	D7	18FF	34	write	11111111	11111111	(note stack
10	D7	18FE	56	write	11111111	11111111	address and
11	B7	000F	C1	POP BC	11111111	11111111	data written
12	F7	18FE	56	read	11111111	11111111	also shows up
13	F7	18FF	34	read	11111111	11111111	when stack is
14	B7	0010	3C	INC A	11111111	11111111	read. Change
15	B7	0011	3C	INC A	11111111	11111111	stack address in
16	B7	0012	3C	INC A	11111111	11111111	location 1,2 to
17	B7	0013	3C	INC A	11111111	11111111	your target ram
18	B7	0014	3C	INC A	11111111	11111111	area.)
19	B7	0015	3C	INC A	11111111	11111111	
1A	B7	0016	3C	INC A	11111111	11111111	

RESET 10 RB resetting (Reset target and break at address 0010)
 AF=1241 (sZ-a-pnC) BC=3456 DE=789A HL=BCDE IX=1234 IY=EFA8 SP=1900 PC= 10
 0010 3C INC A (next step) ok

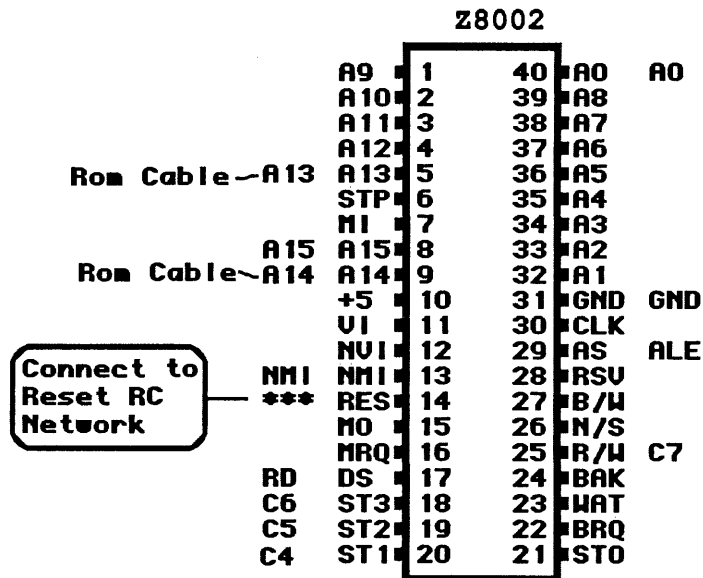
Z8000 DISASSEMBLER/DEBUGGER (DDB-Z8K)

This version supports the Z8000 family of CPUs. The Z8002 and Z8004 are unsegmented versions of this chip and may be run directly. The Z8001 and Z8003 are segmented versions.

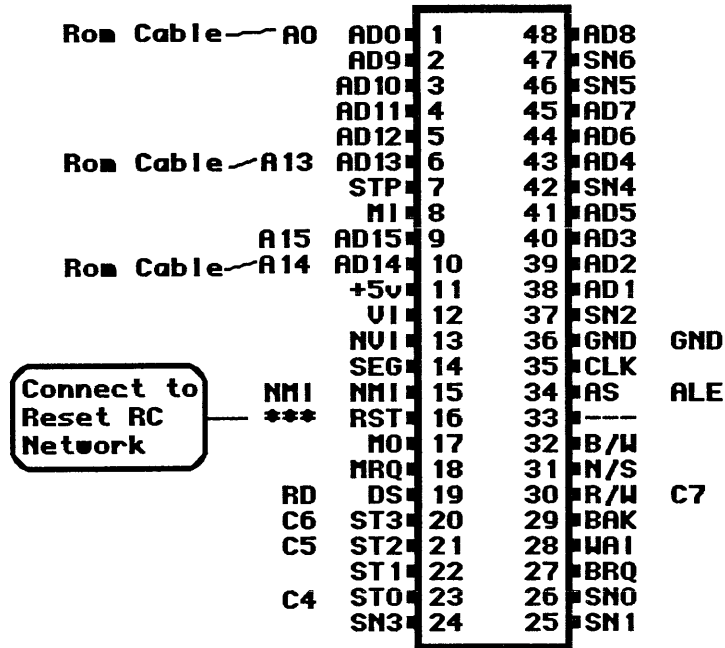
All members of this family require the C analyzer cable.

Patch Words

To use the debugger and disassembler with the segmented chips the segment number for both ram and rom must be set using the **=RAM.SEGMENT** and **=ROM.SEGMENT** commands. Then use **SAVE-SYS** to save the newly configured program.



ANALYZER CABLE C Z8001 ANALYZER CABLE C



Reserved Area

The debugger needs locations 700-4.

The location of the reserved area may be changed by typing in <address> **=OVLAY**.

The overlay area starts at 705.

Help Screen

Press **CTRL-F3** to get the help screen that includes this target-specific information:

Reg chng: n =R0 n =FCW
LTARG uses RAM at 4000. n =RAM to change.
 Overlay area begins at 700 n =OVLAY to change.
 n =SEGMENT sets segment on Z8001 and Z8003.

Trace and breakpoint display

0 =RAM.SEGMENT ok (set initial segments for rom and ram on Z8001 target board)
0 =ROM.SEGMENT ok

LTARG (Load built-in sample program, set up default emulation area)

Emulator Memory Enable Status:

F =EMSEG

0 TO FFF EMENABLE

Breakpoint opcode set to 0F00. ok

STARTUP (Reset the target, and show the first cycles after target starts up)

resetting

(from top of buffer)

cy#	CONT	ADR	DATA		MISC
0	EF	0002	C000	fetch	11111111 (flag Control
1	EF	0004	0000	fetch	11111111 Word)
2	EF	0006	0010	fetch	11111111 (reset PC)
3	FF	0010	210E0000	LD RE, #0	11111111
5	FF	0014	210F4800	LD RF, #4800	11111111 (stack ptr)
7	FF	0018	21024600	LD R2, #4600	11111111 (program
9	FF	001C	7D2D	LDCTL PSAPOFF, R2	11111111 status
area)					
A	FF	001E	7DEC	LDCTL PSAPSEG, RE	11111111
B	FF	0020	21019E00	LD R1, #9E00	11111111
D	FF	0024	7D1B	LDCTL REFRESH, R1	11111111
E	FF	0026	2101C000	LD R1, #C000	11111111
10	FF	002A	6F218000000A	LD 0:A(R2), R1	11111111
13	4F	460A	C000	write	11111111
14	FF	0030	4D258000000C0000	LD 0:C(R2), #0	11111111
18	4F	460C	0000	write	11111111
19	FF	0038	4D258000000E0700	LD 0:E(R2), #700	11111111 (set
exception					
1D	4F	460E	0700	write	11111111 debug vector)
1E	FF	0040	6F218000002A	LD 0:2A(R2), R1	11111111
21	4F	462A	C000	write	11111111
22	FF	0046	4D258000002C0000	LD 0:2C(R2), #0	11111111
26	4F	462C	0000	write	11111111

Pg Dn (trace resume) Home (top) n TN (from step n) T (from n=-5)

RESET 70 RB resetting (reset target, set breakpoint at address 0070H, and run to that address and break, displaying registers, flags.)

R0= 0000 R1= 1111 R2= 4600 R3= D6F9 R4= 4444 R5= FDFB R6= 0008 R7= 000E
R8= 0123 R9= 000D RA= DB8C RB= 3B1B RC= CCCC RD= DDD9 RE= 0000 RF= 47FE
FCW= C0A0 (.Meii...CzSvdh..)
0070 ABD0 DEC RD, #1 (next step) ok

Chapter Nine: TroubleShooting

Overview

We designed this chapter to help you get the **LTARG** sample program running on your target board. The use of a standard program makes it much easier to pinpoint and solve any problem that you have.

A sample trace of the **LTARG** for your processor can be found in the Disassembler/Debugger writeup and is often distributed on your software diskette as well.

Back to Basics

If you run into problems when trying to run your own program, we recommend that you first go back to the **LTARG** program. Check whether your system functions with that simple program.

The **LTARG** program will help you pinpoint the problem-- after all, you have to walk before you learn to run.

When the simple program runs, the solution to the problem with your software is usually clear.

Contents

Explanation	9-2
Solutions in Depth:	
Program hangs up on "Initializing UniLab. . . " message . .	9-3
Program hangs on initialization some of the time, not all of the time	9-5
RS-232 error message "RS-232 Error #XX"	9-6
STARTUP does not work -- never get to see trace, or see trace filled with garbage	9-8
Error message: "NO ANALYZER CLOCK"	9-10
Program runs, UniLab traces, but reads bad data from stack	9-12
Program runs and UniLab traces, but does not disassemble properly	9-13
Program runs, UniLab traces properly, but cannot set a breakpoint-- gives a "Debug Control not Established" message	9-14
Program runs, UniLab traces properly, but cannot set a breakpoint-- hangs with red light next to Analyzer socket on until key pressed	9-15
Bad Input buffers on the UniLab, as if an IC has been blown.	9-16
Screen flickers when you use PgUp key to look at line history.	9-17

Explanation

Symptoms

Find the symptom on the previous page that most nearly matches your problem. Then turn to the page that covers that symptom, to find the solution.

SUB-SYSTEMS OF THE UniLab

When you have a problem, you can usually trace it to only one of the four functional "subsystems" of the UniLab. These four systems handle:

- 1) communications between UniLab and the host computer,
- 2) emulation of target ROM,
- 3) analysis of target board program, setting triggers and capturing traces,
- 4) debugger features, such as setting breakpoints in target board program, single stepping, using **MDUMP** on RAM, etc.

The above list forms a hierarchy of dependence. That is, the other three all depend on the communications subsystem-- if it does not work, then the other three will act bizarrely, or not act at all.

And if you cannot trace properly, then don't bother trying to get a breakpoint.

FIRST THINGS FIRST

Similarly, if the trace show your target board starting to execute at the wrong address, then you can ignore the rest of the trace. You've already found the vital information-- the first opcode address is wrong.

When to call

If you cannot get the **LTARG** program to run, or you cannot figure out where the problem lies, then the next step is Orion Technical Support. Call (415) 361-8883 for assistance.

Solutions in Depth

Program hangs up on "Initializing UniLab. . . " message

Quick Check:

- UniLab plugged in and turned on? Turn it on, enter **INIT**, and try again.
- UniLab connected to COM1? See below.
- Do you have two serial ports? If so, is the second one properly "jumpered" as COM2? See page 9-7.
- Is your serial port set up to work with a printer? See below.

WHY

If the system freezes right after "Initializing UniLab..." is displayed, it means that the program is waiting to receive a character from the UniLab. You can unlock the program by pressing the CONTROL and BREAK keys at the same time.

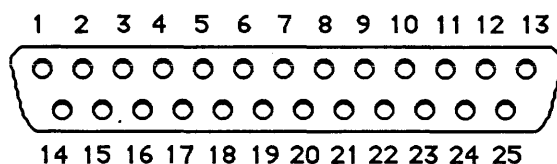
It might be that the UniLab simply has not been hooked up to the correct port.

Another possibility is that the UniLab is on the correct port, but the computer is looking at the wrong pin. The UniLab looks like Data Communications Equipment (DCE) to the host computer. That is, the UniLab expects to receive data on line 2 and send it on line 3 of the RS-232 connection.

WHAT TO DO

Check that the UniLab has been plugged into the correct DB-25 pin connector on your computer. If you have two 25 pin sockets on your computer, you should unlock the program with CONTROL-BREAK and move the UniLab cable to the second socket. Then use the **INIT** command to initialize the UniLab.

DB-25 Connector



If that does not work, use **AUX2** to reconfigure your software, so that it expects the UniLab to be connected

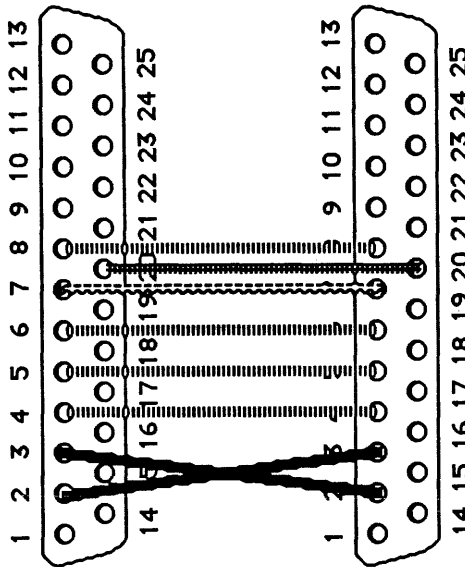
-- Solutions in Depth --

to communications port 2. Again, try to initialize with the UniLab on each of the two serial ports.

If that doesn't work, check that the connector on the outside of your computer has actually been connected to the circuitry inside.

The jumpers on the serial board must be configured for operation with a modem or other DCE (Data Communications Equipment), rather than for operation with a printer or other DTE (Data Terminal Equipment). All serial port boards have jumpers that allow you to change the port to connect to a DTE or a DCE.

Only if you want to keep the port configured for a DTE, you should make or buy a standard "null modem" connector, or the non-standard null modem shown here.



Cable Configuration for
Connection between
UniLab and DTE serial port
(Or use standard null modem)

Program hangs on initialization some of the time, not all of the time

Quick Check:

- Are your cables poorly connected? Check the UniLab to host cable.
- Are you running a background task or a program that tries to write to the screen? See below.
- Does your AUTOEXEC.BAT file set up a background task when you turn on the computer? See below.

WHY

Your real-time clock interrupt might cause missed characters by taking too much time to process interrupts. One common cause of such problems is on-screen clock display utilities from programs such as Side-kick. If the clock interrupt routine does anything time-consuming, like writing to the screen, it can affect communications with the UniLab.

WHAT TO DO

Do not run desk accessories or background tasks such as a print spooler, on-screen clock display, alarm clocks, or multitasking, if you find that they affect the communications with the UniLab.

Take a look at your autoexec file, with the DOS command **TYPE AUTOEXEC.BAT**, to see if a background task has been set up to start automatically.

-- Solutions in Depth --

RS-232 error message "RS-232 Error #XX"

Quick Check:

- Poorly connected cable? Check the UniLab to host cable.
- Running a background task or desk accessory program? See previous page.

WHY

The program running on your host sees incorrect data coming back when it tries to talk to the UniLab. The hex number following the # sign indicates what the error is.

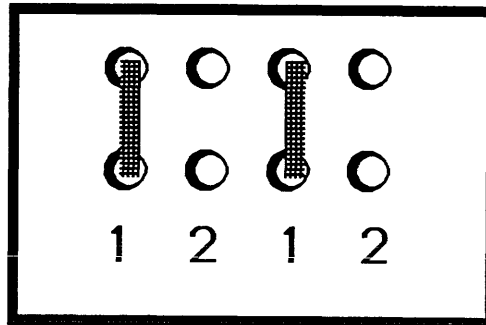
All RS-232 messages are checked with a 16-bit checksum and acknowledged with a single ACK (06) byte. Errors are signaled with other single-character responses as follows:

9B	Timeout error
A1	Overrun or framing error
69,70,75, 7C	Checksum error
2D	Length error
54	Load address error.

Other error codes indicate that the host is not receiving the UniLab transmissions correctly. For example, a 9600-baud host will usually get error CC or FC from a 19,200-baud UniLab, while a 19,200 baud host connected to a 9600-baud UniLab will usually get error 9E or FE.

WHAT TO DO

If you have two ports that might be "jumpered" to the same address, open up your computer and look at the boards that connect to the DB-25 connections on your computer. One or both boards should have a group of eight pins, as shown on the next page. This set of jumpers is different from the set that determines whether your port tries to talk to a DTE or a DCE, which are explained on page 9-4.



Typical pin arrangement on Serial port board

When the pins are jumpered as shown, then the port will be COM1. If the pins labeled 2 are joined, then the port will be COM2. If both ports are trying to be the same port, then you will have a bus contention problem.

If you want to run a diagnostic test on your RS-232 port, first disconnect the UniLab from the host. Try sending single characters from within the UniLab program. You will have to jumper pins 2 and 3 on the DB-25 connector of your computer. Then type **INITRS232** to get the port ready. **30 SEND** will then send out a 30 on the serial port. Since pins 2 and 3 are jumpered together, the same port should receive a 30. Type **RCV .** to see what was received.

Try sending other numbers as well. If the port works fine, then you should suspect that pins two and three are reversed on that port. You can take care of this by putting in a null modem that switches pin 2 to pin 3 and pin 3 to pin 2. Pin 7 carries ground, and no other connections are necessary. See diagram below.

Another way to check the port. Put an oscilloscope on pin 2 of the DB-25 connector, and then follow the procedure above for sending characters. You will see on the oscilloscope whether or not signals are being sent on pin 2..

-- Solutions in Depth --

STARTUP does not work -- never get to see trace, or see trace filled with garbage

Quick check:

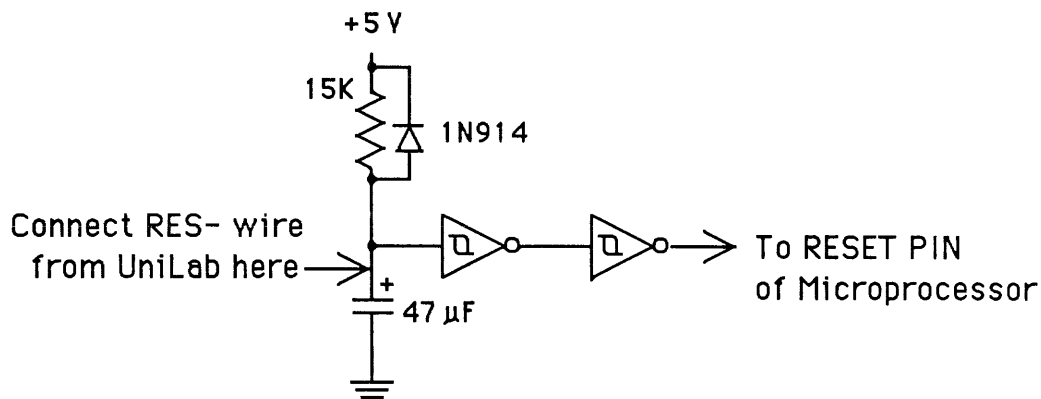
- UniLab turned on and initialized? Turn it on now, then issue command **INIT**.
- **LTARG** has been loaded? Issue the command **LTARG**.
- **RES-** wire not connected properly? See below and explanation in section 3 of chapter 2.
- Are address lines properly hooked up? See below.
- Error message: "NO ANALYZER CLOCK" ? See page 9-10.
- Do you have an **8051 family** processor? These require a positive going reset signal. See the Disassembler/Debugger Notes, or Section 3 of Chapter 2.

WHY

STARTUP watches for the reset address on the bus, and then lets the trace buffer fill up before freezing the trace. The trace buffer will never be displayed on the screen if a proper reset never occurs, or if the lines the UniLab uses to sense the address have not been hooked up properly.

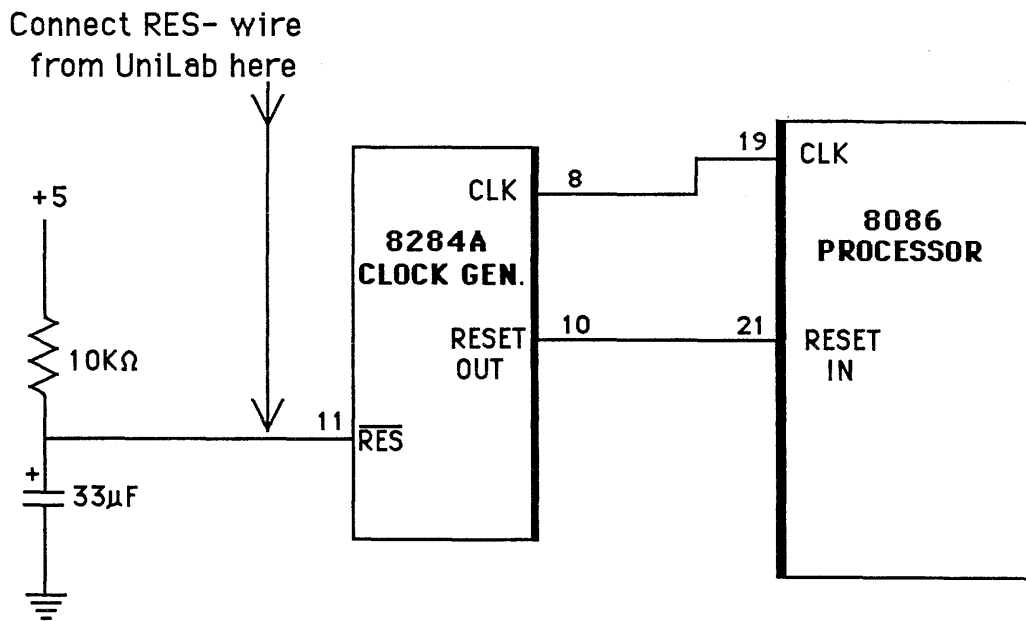
WHAT TO DO

Check the reset wire. First check it visually if you are using cables (as opposed to pods)-- make certain that it has been connected to the RC circuit that drives the logic gate which drives the reset pin. See below. Then check at the reset pin of the processor, with a logic probe or oscilloscope, to make certain that the chip gets a good reset signal when you **STARTUP**.



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

Check the address lines. Make certain that the ROM cable makes a good connection in the socket on your board, and that the extra address lines, if any, connect to the DIP clip at the proper pins.



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

-- Solutions in Depth --

Error message: "NO ANALYZER CLOCK"

Quick check:

- Power supply connected to target, and power on?
(Not necessary with pod connection).
- Bad control wire connections? See below.
- **RES-** wire not connected properly? See pages 9-8 and
Section 3 of Chapter 2.
- Target system has gone to sleep? See below.

WHY

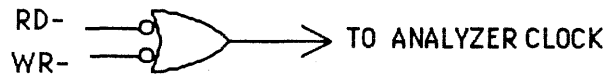
Usually a result of a bad connection-- the UniLab does not sense the control signals from the microprocessor, which tell it when the bus contains valid data and when it holds a valid address. If the power is not on, or the processor is doing nothing because it has not seen a reset, then there will be no clock.

The same symptom will result if the target system receives a command to wait or stop. This could happen if there is a bad data or address line, or if the target board does not restart because of an improperly connected reset signal.

WHAT TO DO

Bad control wires. Double-check all the control wires from the UniLab that sense the timing signals. These are **K1-**, **K2-**, **RD-**, and **WR-**.

Also check whether or not the you have the proper analyzer cable for your microprocessor.



CLOCK LOGIC FOR MOTOROLA PROCESSORS

Target system asleep. Hit any key to stop the search for trigger, and then use the command **TD** to dump the trace. Look at the bottom of the trace, and see if the processor executed a halt command.

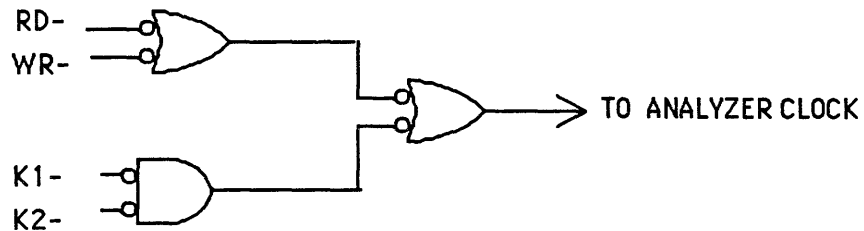
If it did, then check the address of the code that told the processor to stop. Is that an address that the

processor should reach? If you use **DM** to disassemble from memory, is the opcode commanding a halt actually there?

If the program should not get to the code address where it reads the halt, check the **RES-** wire. See two pages back.

You should also check the address lines.

If the address is correct but the data is wrong, check the data lines.



CLOCK LOGIC FOR INTEL PROCESSORS

-- Solutions in Depth --

Program runs, UniLab traces, but reads bad data from stack

Quick Check:

- Stack pointer not pointing at correct location? See below.
- RAM chip bad? See below.

WHY

The test boards that we use at Orion have RAM at some location in the memory map. The program loaded into emulation memory by the **LTARG** command was developed for our test boards.

But you might very well not have RAM at the address range where the Orion boards have RAM.

WHAT TO DO

To determine whether the stack pointer is okay, look at the first few lines of the trace, where the stack pointer gets initialized. Check the value of the stack against the addresses on your board that are occupied by RAM. Remember that the stack grows by decrementing the pointer.

If the stack pointer needs a different value, use the UniLab command `<word> <addr> MM!`. You use that command to change the program memory. It pokes a new 16 bit word into emulation ROM. Or, if your package includes it, use the optional on-line assembler, **ASM**, to change the instruction.

You will want to change the address field of the instruction that initializes the stack pointer. You should patch the program, so that the 16-bit address of the stack pointer points to RAM.

If the stack pointer points to RAM, but you still get bad values off the stack, you should suspect that you have a bad RAM chip. Try swapping in a new chip. If the bad values continue, then start checking your data and address line connections, both between the board and UniLab, and on the board.

Program runs and UniLab traces, but does not disassemble properly

Quick check:

- First address of trace is not reset address? You might have an improperly connected reset wire. See page 9-8.
- No disassembly at all? See below.
- Clock speed of target board too high? See below.
- **CONT** column correct? See below.
- Correct disassembler for microprocessor? See below.
- Correct cable for microprocessor? Double-check it.
- Cable wired to DIP CLIP correctly? See below.

WHY

The UniLab listens to the bus, and interprets each 8-bit value it sees as opcode, data, or address, depending on the control signals it reads from the microprocessor. It then tries to disassemble the commands it sees, based on the disassembly table.

If you don't have all the wires connected properly, or you have the wrong disassembler, or you don't have the disassembler turned on, then you will not see what you expect.

WHAT TO DO

If the first address of the trace is wrong, then there is no point in looking at anything else. The only thing that matters is the first wrong step. Look at page 9-8.

If you think that you need to enable disassembler, just type in the command **DASM**.

If the target system clock is too fast for the UniLab, then some bus cycles will be skipped. Check for missing addresses in the fetch stream.

If the left digit of the CONT column is different from the sample trace, then the UniLab software does not know whether each bus cycle is a read, a write, or a fetch. That digit is generated by **K1-**, **K2-**, **RD-**, and **WR-**.

If you don't have the proper disassembler for your processor, then it's a surprise that you have gotten this far. Type the UniLab command **PINOUT** to find out what processor your software thinks it supports.

To check that you connected to the DIP clip correctly, look at the connection diagram in the **TARGET NOTES** section of the manual.

-- Solutions in Depth --

Program runs, UniLab traces properly, but cannot set a breakpoint-- gives a "Debug Control not Established" message
Quick Check:

- Are you trying to set a breakpoint at address 0?
This is not allowed-- try another address.
- Does the analyzer work correctly? See below.
- Does the emulator work correctly? See below.
- Stack pointer points to real memory? See below and on page 9-12.
- Breakpoint set in code after stack pointer initialized? See below.

WHY

The UniLab sets breakpoints by preserving the byte at the address you specify, and inserting an absolute jump instruction, or a software interrupt. When the target system reaches that code, control gets passed to our idle register. The idle register asserts a jump instruction on the bus, which causes the processor to jump to the beginning of the jump instruction. The idle register thus holds the processor in an infinite loop.

Meanwhile, the UniLab software uses "an overlay area" in ROM, putting a routine there to save and read the state of the processor. (HO tells you where the overlay area is for your processor.) The idle register then changes, to assert a jump to the overlay area, which ends in a jump back to the idle register. All this swapping of control requires that the functions of the analyzer and emulator already work properly, and that the stack pointer points to functioning RAM.

WHAT TO DO

If either the analyzer or emulator does not work properly, then you must get those functioning before you try to set a breakpoint. Especially, read/write cycles must be properly identified-- which means that the disassembler must be working.

On processors with an external stack, if the stack pointer does not point to a good RAM chip, then you will not be able to set a breakpoint. Look at the trace that results from **STARTUP**, paying special attention to the POPs off the stack. Does the same data you push on the stack come back off it? If not, look at patch instructions on page 9-12.

You must not set a breakpoint until after the stack pointer has been initialized. Try setting the breakpoint at the example address shown in the Disassembler/Debugger writeup for your processor.

Program runs, UniLab traces properly, but cannot set a breakpoint-- hangs with red light next to Analyzer socket on until key pressed

Quick Check:

-- Can you set a trigger on the breakpoint address? (That is, does program actually reach that address?) See below.

WHY

The UniLab sets a breakpoint at the address that you specify. It sets the break point by swapping in a special code at that address.

When the microprocessor hits that code, it behaves like a train sent onto a siding in the train yard. But if you don't put the code at the start of an instruction, then the microprocessor gets derailed.

WHAT TO DO

Are you setting a breakpoint at the start of an opcode? Double-check this by setting a trigger on the address at which you are trying to set a breakpoint. The UniLab command is **NORMT <address> AS**. If you cannot set a trigger on the address, then that address is not the start of an opcode. Try a byte or two in either direction.

-- Solutions in Depth --

Bad input buffers on the UniLab, as if an IC has been blown

Quick check:

-- Does the UniLab mysteriously show bad data coming in? Do some inputs always show up high or always remain low, despite the reality? See below.

WHY

If you blow one of the input buffers-- by frying it with a high voltage, or through some other mishap-- then the IC will be damaged.

All ICs with external inputs are socketed, so they can easily be replaced.

WHAT TO DO

Make certain that you know what the problem is. Connect the suspicious input to ground, and then capture a trace. Connect the input to +5 voltage, and capture a trace. Inspect both traces to determine whether or not the input responds to the actual state of the circuit that it is meant to measure.

If you have a blown IC. Either send it to Orion for repair work, or replace the ICs yourself. All the Orion chips are standard pieces. All chips with external inputs are socketed for easy replacement.

Inputs:	Chip # on board:	Input Group:
A0-A7	U14	LADR
A8-A15	U15	HADR
D0-D7	U9	DATA
D8-D15	U8	HDATA
M0-M7	U7	MISC
C4-C7	U6	CONT

-- Solutions in Depth --

Screen flickers when you use PgUp key to look at line history

Quick check:

-- Issue the command **CLEAR**. Then use
SAVE-SYS to save the altered system.

LIST OF APPENDICES

- Appendix A: UniLab Command and Feature List
- Appendix B: Sources of Cross Assemblers and C Compilers
- Appendix C: Cabling Chart
- Appendix D: Custom Cables
- Appendix E: UniLab II Specifications
- Appendix F: Writing Macros
- Appendix G: EPROMs and EEPROMs Supported
- Appendix H: Microprocessors Supported
- Appendix I: System Messages
- Appendix J: **.BIN** Files and **.TRC** Files

**Appendix A:
UniLab Command and Feature List**

1AFTER	11
16BIT	13
2AFTER	14
3AFTER	14
8BIT	15
:	16
;	17
=BC	18
=EMSEG	19
=HISTORY	21
=SYMBOLS	23
?FREE	23
ADR	24
ADR?	25
AFTER	26
AHIST	28
ALSO	29
ALT-FKEY	30
ALT-FKEY?	30
AS	31
ASC	32
ASEG	33
ASM	34
ASM-FILE	36
AUX1	37
AUX2	37
B#	38
B.	38
BINLOAD	39
BINSAVE	40
BPEX	41
BYE	42
CATALOG	42
CKSUM	43
CLEAR	44
CLEAR'	44
CLRMBP	45
CLRSYM	46
COLOR	47
COM1	48
COM2	49
CONT	50
CONTROL	52
CTRL-FKEY	53
CTRL-FKEY?	53
CYCLES?	54

-- Command List --

D#	55
DASM	56
DASM'	57
DATA	58
DCYCLES	60
DEF	61
DM	62
DMBP	62
DN	63
DOS	64
EMCLR	65
EMENABLE	66
ESTAT	68
EVENTS?	69
FETCH	70
FILTER	71
FKEY	72
FKEY?	73
G	74
GB	75
GW	76
H>D	77
HADR	78
HDAT	79
HDATA	80
HDG	82
HDG'	82
HELP	83
HEXLOAD	84
HEXRCV	86
INFINITE	87
INIT	88
INT	89
INT'	90
IS	91
LADR	92
LOG	93
LOG'	93
LP	94
LTARG	95
M	96
M!	97
M?	98
MASK	99
MCOMP	100
MDUMP	101
MEMO	102
MENU	104
MESSAGE	104
MFILL	105

-- Command List --

MISC	106
MISC'	108
MLOADN	109
MM	110
MM!	111
MM?	112
MMOVE	113
MODE	114
MS	115
N	116
NDATA	117
NMI	118
NMIVEC	119
NMIVEC'	119
NORMB	120
NORMM	121
NORMT	122
NOT	123
NOW?	124
ONLY	125
ORG	127
PAGE0	128
PAGE1	128
PAGINATE	129
PAGINATE'	129
PCYCLES	130
PEVENTS	131
PINOUT	132
PRINT	133
PRINT'	133
PROMMSG	133
Q1	134
Q2	134
Q3	134
QUALIFIERS	135
RB	136
READ	138
RES	139
RESET	140
RESET'	140
RMBP	141
RSP	142
RSP'	142
RZ	143
S	144
S+	145
SAMP	146
SAVE-SYS	147
SC	148
SET	149

-- Command List --

SET-COLOR	149
SHIFT-FKEY	150
SHIFT-FKEY?	150
SHOWC	151
SHOWC'	151
SHOWM	152
SHOWM'	152
SMBP	153
SOURCE	154
SOURCE'	154
SR	156
SST	157
SSTEP	158
STANDALONE	159
STARTUP	160
STIMULUS	161
SYMB	162
SYMB'	162
SYMFILE	163
SYMFILE+	164
SYMFIX	165
SYMLOAD	166
SYMSAVE	166
SYMTYPE	167
T	168
TCOMP	169
TD	170
TEXTFILE	171
THIST	172
TMASK	173
TN	174
TNT	174
TO	175
TOFILE	176
TOFILE'	176
TOP/BOT	177
TRIG	178
TS	179
TSAVE	180
TSHOW	181
TSTAT	182
WORDS	182
WSIZE	183

**Appendix B:
Sources of Cross Assemblers and C Compilers**

The UniLab software is designed to work with any assembler or compiler. The only thing the UniLab needs is the object code in either binary format or INTEL hex format.

Even this hurdle can be overcome with one of the various conversion programs on the market. For example, Avocet has a product which converts Motorola S-records into binary format. See the Vendor listing for Avocet below.

As a service to our users we have compiled the following list of inexpensive cross assemblers and compilers. The two character abbreviations indicate the sources listed on the following pages. We would appreciate any user feedback so that we can keep this list current.

PROCESSOR	ASSEMBLER SUPPLIERS	COMPILER SUPPLIERS
SC/MP	MI	
NS16000	25,PC	
1802/5	25,AV,MI,WE,AA,EN,RE,SD,UW	
3870/F8	25,AV,MI,AA,SD,UW	
COP 400	25,AV,AA	
6301	25,EN,RE,UW	IT
6502	25,AV,MI,EN,RE,SD	MX
6800/2/8	25,AV,MI,DM,EN,RE,SD,UW	
6801/3	25,AV,MI,DM,EN,RE,SD,UW	IT
6805	25,AV,MI,DM,EN,RE,SD,UW	IT
6809	25,AV,MI,DM,EN,RE,SD,UW	IT
68000	25,AV,QU,EN,SD,UW	MX,IT,MT,LA,UW
68HC11	RE,SD,UW	IT
TMS 7000	AA,EN,CY,SD	
NEC 7500	25,AV	
NSC800	25,EN,RE	MT
8048-50/41	25,AA,AV,MI,CY,RE,SD	
8051/31	25,AA,AV,MI,CY,RE,SD,UW	AR,MC
8054	MI	
8085	25,AV,MI,EN,CY,RE,SD,UW	MT
8086/8	25,SW,EN,SP,CY,SD,UW	MX,MS,MT,LA
8096	25,CY	
9900/5	AA,EN,RE	
Z-8	25,AV,AA,EN,CY,RE,SD,UW	
Z-80, 64180	25,AV,AA,EN,RE,SD,UW	MX,KY,LA
Z-8000	25,EN,RE	

Vendor List begins on next page.

VENDORS

NOTE: All prices are approximate. Contact the vendor directly for latest information. This listing is a service to our customers, and does not constitute a recommendation.

- 25 2500AD Software Inc.
17200 East Ohio Dr.
Aurora, CO 80017, (303) 369-5001.
Eight-bit versions are \$199.50, 16-bit are \$299. They include recursive macros, nested conditional assembly, listing control, and a linker.
- AA Allen Ashley
395 Sierra Madre Villa
Pasadena, CA 91107, (818) 793-5748.
Resident editing capability, assemble to memory. \$150.
Macro/relocatable versions also available for \$250.
- AR Archimedes, (415) 771-3303. C cross compiler for 8051. \$851.
- AV Avocet Systems Inc.
804 South State St.
Dover, DE 19901, (302) 734-0151. (800) 448-8500.
\$200 for CP/M-80 or MS-DOS versions. \$500 for the NEC 7500 and 68000.
HEXTRAN converts Motorola S-records to binary format. \$250.
- CY Cybernetic Micro Systems
P.O. Box 3000
San Gregorio, CA 94074, (415) 726-3000.
Conditionals, Macros. \$295. Written in 8088 assembler.
- DM Decision MicroSystems Co.
Box 120783
Nashville, TN 37212, (615) 320-7221. \$210.
- EN Enertec Inc.
19 Jenkins Ave.
Lansdale, PA 19446, (215) 362 0966. \$250 and up.
- FA Farbware
1329 Gregory
Wilmette, IL 60091.
Structured macro assembler \$200.
- HA Hawkeye Grafix
23914 Mobile
Canoga Park, CA 91307, (213) 348-7809. \$100.

IT Introl, (414) 276-2937.
C cross compiler for 6801, 6301, 6805, 6809, 68HC11,
68000, 68020. \$1950.

KY KYSO, (503) 389-3452.
C cross compiler for Z80.

LA Lattice, (312) 858-7950.
C cross compiler for 68000, 8088, Z80. \$500.

MC MicroComputer Control, (609) 466-1751.
C cross compiler for 8051. \$1495.

MI Midwest Micro-DelTek, Inc.
5930 Brooklyn Blvd.
Brooklyn Center, MN 55429, (612) 560-6530.
Limited macros, cross reference, conditionals, 1K of
object/minute. \$300.

MS MicroSoft
10700 Northup Way
Bellevue, WA 98004. C compiler for 8086, 8088. \$495.
As of release 4.0 of their software, MicroSoft does not make
ROMable code directly. You can purchase utilities which are
supposed to make the output of the MicroSoft compiler into
ROMable code.

MT MicroTek, (408) 733-2919.
C cross compiler for 68000, 68008, 68010, 68020. \$1750.
C cross compiler for 8085, Z80, 64180, 8088, 8086,
80188. \$1550.

MX Manx Software Systems
One Industrial Way
Eatontown, NJ 07724, (800) 221-0440.
C cross compiler for 8086, 68000, 8080, Z80, 6502.
\$750.

PC Program Concepts Inc.
P.O. Box 8164
Charlottesville, VA 22901, (804) 978-1850. \$595.

QU Quelo
843 NW 54 th St.
Seattle, WA 98107, (206) 784-8018.
Macros, conditionals, linker, cross ref, in C, \$300.

RD RD Software
1290 Monument St.
Pacific Palisades, CA 90272, (213) 459-8119.
This is based on one that appeared in Dr. Dobb's
Journal in June 1981 and April 1982. \$200.

RE Relational Memory Systems
 PO Box 6719
 San Jose, CA 95150, (408) 265-5411.
 Three different prices:
 Macro assembler, non-relocatable. \$139.
 Relocatable code for 8-bit systems. \$395.
 Relocatable code for 16-bit systems. \$495.

SD Software Development Systems
 3110 Woodcreek Dr.
 Downers Grove, IL 60515, (312) 971-8170. \$295
 Relocatable code, macros.

SE Seattle Computer Products, Inc.
 1114 Industry Dr.
 Seattle, WA 98188, (206) 575-1830. \$95.

SO Solutionware Corporation
 1283 Mt. View-Alviso Rd., Suite B
 Sunnyvale, CA 94086. Debuggers also.

SW Speedware
 118 Buck Circle, Box T
 Folsom, CA 95630 (916) 988-7426. \$99.
 With resident editor similar to Turbo-Pascal. Written
 in 8088 assembly language for speed.

SY Syscon Corp.
 3990 Sherman St.,
 San Diego, CA 92110, (619) 222-6381. \$250.

UW UniWare
 Software Development Systems
 3110 Woodcreek Dr.
 Downers Grove, IL 60515, (312) 971-8170.
 8 and 16-bit cross-assemblers, \$295.
 C cross-compiler for 68000, \$595.

WE Westico
 25 Vanzant St.
 Norwalk, CT 06885, (203) 853-6880. \$225 Macro, \$225 Linker.

WW Western Wares
 Box C,
 Norwood, CO 81423, (303) 327-4898. \$395.

All of the Intel Series III MDS software can be run on the IBM PC with the UDI package from Real-Time Computer Science Corp., P.O. Box 3000-886, Camarillo, CA 93011, (805) 482-0333 (\$500), or the ACCESS package from Genesis Microsystems, 196 Castro St., Mountain View, CA 94041, (415) 964-9001.

Appendix C: Cabling Chart

- | | | |
|----|---------------------|-----|
| 1. | Non-Piggyback chips | C-1 |
| 2. | Piggyback chips | C-7 |

PROCESSOR:	1802	16032	6301X0+	6303R	6303X	6502	6800
CABLE	g	c	b	b	b	b	b
<u>Cable wires:</u>							
A11		12	26	26	38	20	20
A12		11	25	25	37	22	22
A13		10	24	24	36	23	23
A14		9	23	23	35	24	24
A15		8	22	22	34	25	25
*RES-	3	34	6	6	6	cap	cap
*NMI-	36	45	4	4	8	6	6
GND	20	25	1	1	1	1	21
RD-	7	33	40	40	64	39	37
WR-	35						5
K1-							
K2-							
ALE	33	37					
C7	5	40	38	38	61	34	34
C6	6	41			60	7	
C5		42			63		
C4		43			62		
A19		4					
A18		5					
A17		6					
A16		7					
A0						2	

- + at the end of the processor name indicates expanded mode or max mode
- * RES- and NMI- are open collector outputs. Connect only to appropriate points. (NMI- needed only for certain debugger operations)

PROCESSOR:	68000	68008	6801+	6802	6805E2 6805E3	6809E
CABLE	p	p	b	b	b	b
Cable wires:						
A11	39	9	26	20	16	19
A12	40	10	25	22	15	20
A13	41	11	24	23	gnd	21
A14	42	12	23	24	gnd	22
A15	43	14	22	25	gnd	23
*RES-	18	37	6	cap	1	37
*NMI-	23	42	4	6	2	2
GND	53	15	1	21	20	1
RD-	14	13	40	37	4	34
WR-				5		
K1-	10	31				
K2-	6	28				
ALE						
C7	9	30	38	34	5	5
C6	26	43		7	3	32
C5	27	44				38
C4	28	45				36
A19	47	19				
A18	46	18				
A17	45	17				
A16	44	16				
A0	7	46				

- + at the end of the processor name indicates expanded mode or max mode
- * RES- and NMI- are open collector outputs. Connect only to appropriate points. (NMI- needed only for certain debugger operations)

PROCESSOR:	68HC11	80186	80188	80286	8031+
CABLE	b	a	a	i	e
Cable wires:					
A11	13	10	10	20	24
A12	12	7	7	19	25
A13	11	5	5	18	26
A14	10	3	3	17	27
A15	9	1	1	16	28
*RES-	39	24	24	cap	ckt
*NMI-	40	ckt	ckt	ckt	12
GND		26	26	9	20
RD-	27	62	62	(11)	29
WR-		63	63	(9)	16
K1-		40	40	(17)	17
K2-		39	39	(16-)	31
ALE		61	61	(5)	
C7	28	54	54	67	
C6	26	53	53	4	
C5		52	52	5	
C4				66	
A19		65	65	12	
A18		66	66	13	
A17		67	67	14	
A16		68	68	15	
A0		17		34	

+ at the end of the processor name indicates expanded mode or max mode

() indicates a bus controller pin

* RES- and NMI- are open collector outputs. Connect only to appropriate points. (NMI- needed only for certain debugger operations)

ckt connect to processor pin through an inverting circuit

PROCESSOR:	8048+	8080	8085	8086	8086+
CABLE	e	h	a	a	l
Cable wires:					
A11	24	40	24	5	5
A12	gnd	37	25	4	4
A13	gnd	38	26	3	3
A14	gnd	39	27	2	2
A15	gnd	36	28	39	39
*RES-	4		36	cap	cap
*NMI-	6		ckt		
GND	20	2	20	20	20
RD-		18	32	32	(11)
WR-	10		31	29	(9)
K1-	8	{ 1 }	11	27	(1)
K2-	gnd	{ 6 }	3	24	(4)
ALE		17	30	25	(5)
C7		4	34	28	28
C6		3			27
C5		9	29		26
C4		10	33		
A19				35	35
A18				36	36
A17				37	37
A16				38	38
A0				16	16

- + at the end of the processor name indicates expanded mode or max mode
- () indicates a bus controller pin
- { } indicates a clock controller pin
- * RES- and NMI- are open collector outputs. Connect only to appropriate points. (NMI- needed only for certain debugger operations)

PROCESSOR:	8088	8088+	8096	HD64180	NSC800
------------	------	-------	------	---------	--------

CABLE	a	l	r	e	q
Cable wires:					
A11	5	5		24	4
A12	4	4	latch	25	5
A13	3	3	latch	26	6
A14	2	2	latch	27	7
A15	39	39	latch	28	8
*RES-	cap	cap	62	7	33
*NMI-			ckt	8	21
GND	20	20	42	1	20
RD-	32	(11)	17	63	32
WR-	29	(9)	38	62	31
K1-	27	(16)	9	61	26
K2-	24	(4)		58	37
ALE	25	(5)	16		30
C7	28	28			
C6		27			
C5	34	26			29
C4			15		27
A19	35	35		59	34
A18	36	36		31	
A17	37	37		30	
A16	38	38		29	
A0	16		latch		

+ at the end of the processor name indicates expanded mode or max mode

() indicates a bus controller pin

* RES- and NMI- are open collector outputs. Connect only to appropriate points. (NMI- needed only for certain debugger operations)

ckt connect to processor pin through an inverting circuit

latch attach the UniLab wires to the outputs of the latches, not directly to the processor pin.

PROCESSOR:	(8800) SUPER 8	(8681/82) Z-8+	(8400) Z-80	Z8001	Z8002
------------	-------------------	-------------------	----------------	-------	-------

CABLE	d	d	e	c	c
Cable wires:					
A11	45	16	1	4	3
A12	44	17	2	5	4
A13	43	18	3	6	5
A14	42	19	4	10	9
A15	41	20	5	9	8
*RES-	30	6	26	16	14
*NMI-			17	15	13
GND	34	11	29	36	31
RD-	37	8	21	19	17
WR-			22		
K1-			27		
K2-			20		
ALE				34	29
C7	31	7		30	25
C6				20	18
C5				21	19
C4				23	20
A19			19		
A18					
A17					
A16					
A0				1	40

* RES- and NMI- are open collector outputs. Connect only to appropriate points. (NMI- needed only for certain debugger operations)

2. Piggyback Chips

PROCESSOR: | HD63P01 | 65/11EB | 65F11Q | 65/41EB | 68P01 |

CABLE	n	k	k	k	n
Cable wires:					
A11	rom	rom	9	rom	rom
A12	join	join	8	join	join
A13	together	together	60	together	together
A14	A12-A15	A12-A15	61	A12-A15	A12-A15
A15			7		
*RES-	6	20	6	20	6
*NMI-	4	22	23	22	4
GND	40	21	44	40	40
RD-	1	3	45	3	1
WR-					
K1-					
K2-					
ALE					
C7			40		
C6					

* RES- and NMI- are open collector outputs. Connect only to appropriate points. (NMI- needed only for certain debugger operations)

ckt connect to processor pin through an inverting circuit

rom A11 connects through the ROM plug

(8613/03)

PROCESSOR: | 68P05V07 | 80C51VS | 87P50 | Z8 |

CABLE	m	e	f	e
Cable wires:				
A11	rom	rom	rom	rom
A12	gnd	gnd	gnd	gnd
A13	gnd	gnd	gnd	gnd
A14	gnd	gnd	gnd	gnd
A15	gnd	gnd	gnd	gnd
*RES-	2	ckt	4	6
*NMI-	3	12	6	
GND	1	20		11
RD-				9
WR-			10	
K1-		8	8	
K2-		gnd	gnd	
ALE				
C7				7
C6				

* RES- and NMI- are open collector outputs. Connect only to appropriate points. (NMI- needed only for certain debugger operations)

ckt connect to processor pin through an inverting circuit

rom A11 connects through the ROM plug

gnd ground these address lines

Appendix D: Custom Cables

How Cables Work	D-1
Problems with Decoded OE- Signals	D-2
Customizing Cables	D-3
Analyzer Connector Signals	D-4
Analyzer Cable Design	D-5
The ROM Cable	D-9
ROM Connector Signals	D-10
UniLab Circuitry	D-11
Analyzer Cable Schematics	D-12

How Cables Work

The Sockets

The two 50-pin connectors on the front of the UniLab bring out extra signals so that operation of the instrument can be easily altered to meet the needs of different processors.

Since clocking logic requirements vary from one processor family to another, jumpers on the connector are used to make some interconnections.

Altering Standard Cables

Standard ribbon cables are provided that will work for most systems. In some cases, these cables must be reconfigured for proper operation with your system.

Since the connections are all made by the same insulation-displacement "U" contacts used in "Scotchflex" and "Quick-Connect" prototyping systems, they can easily be changed. A special wire-insertion tool is included with your UniLab for this purpose.

The Analyzer Cable

The analyzer is internally connected to all of the signals on the ROM cable. Any additional signals required for full monitoring of bus operations are picked up by connecting patch wires on the analyzer cable to your processor pins. This is usually done with a 40-pin Dip-Clip. The wires can also be plugged in to .025" wire wrap pins.

Your UniLab comes with an analyzer cable that is configured for the processor of your choice. You can alter your cable to support other processor families, or purchase additional cables.

Problems with Decoded OE- Signals

Most of the analyzer cable configurations (all except B, H, M, & N-- see the diagrams at the end of this appendix) assume that you have a memory enable signal connected to the OE- pin of the ROM socket into which the emulator cable is plugged. the OE- pin is pin 20 of the 24-pin ROM, pin 22 of the 28-pin ROM.

If you have address decoding in this signal, or if this pin is simply grounded, there may be problems.

The problem is that this signal is used as a low true master enable for the emulator's tri-state data bus outputs. This signal is necessary in systems that multiplex addresses over the data bus to prevent the UniLab from getting on the bus while addresses are being multiplexed.

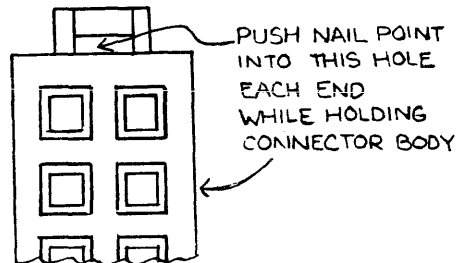
Most of the cable designs connect this input to the OE- signal on the ROM socket with a jumper between pins 42 and 39 on the analyzer connector. (The OE- signal is passed inside the UniLab from the ROM cable to pin 39 of the analyzer connector.)

If your target system has a ground or address decoding on the OE- pin of the ROM socket, you may have to separately connect pin 42 of the analyzer cable to an appropriate memory enable signal. On Intel bus controllers this signal is called MEMR-. Note that this signal also prevents the UniLab from getting on the bus during I/O cycles.

The problem with address decoding in the OE- signal is that the UniLab will then be unable to emulate memory for other ROM sockets.

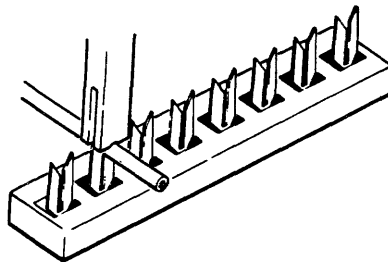
Customizing Cables

Two special tools are included with the UniLab so that you can easily modify the cables supplied. The first tool is actually a # 16 wire brad with a piece of shrink tubing on it. You can use it to open the connector by poking it into the space at the end of the connector as shown below:



Carefully release each end of the connector before you try to remove the cover. Don't try to remove the cover without the nail or you may break the plastic tabs!

Wires can be disconnected by simply pulling them out of the wedge in the connecting pin with small needle-nose pliers. A special tool is included with your UniLab for pushing wires into the connecting wedges (it looks like a tiny baseball bat). Note that these connections are identical to the ones used on "Quick Connect", "Speedwire", and "Scotchflex" prototyping boards. The drawing below shows the proper way to use the tool.



You can use #26-30 solid or stranded wire for making connections. Wire-wrap wire works nicely. If you are jumpering a probe wire across to a second pin (as on pins 21-22 in fig e), hold the wire in place with your thumb while you use a small needle-nosed plier to put the necessary "jog" in the wire before you use the insertion tool.

If you are working with several different families of processors which require different jumper options, you should probably buy additional analyzer cables so you don't have to change jumpers whenever you change processor family.

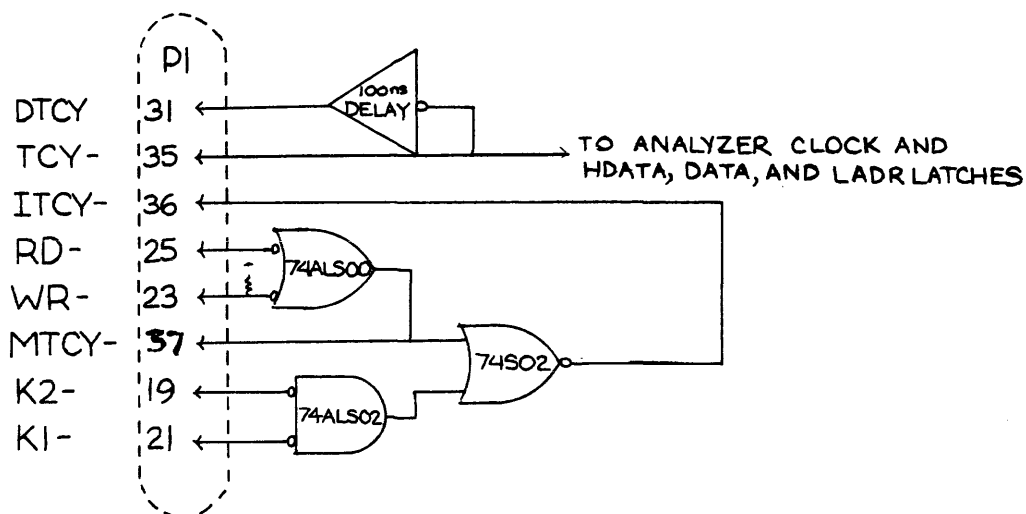
Analyzer Connector Signals

PIN#	SIGNAL	REMARKS
1	M7	MISC analyzer byte input.
2	M6	"
3	M5	"
4	M4	"
5	M3	"
6	M2	"
7	M1	"
8	M0	"
9	GND	Signal ground.
10	A19E	Msb emulator address inputs. Page select only
11	A18E	"
12	A17E	"
13	A16E	Msb emulator address input. To enable RAM.
14	+5V	Not used. Could power external circuits.
15	RDD	Emul enab. Can use to disable processor RD'.
16	RES-	Target reset. Open 7406 collector + 47 ohms.
17	NMI-	Interrupt. Open 7406 collector.
18	GND	Return for RES-.
19	K2-	Clock input. $ITCY' = MTCY + (K1' \text{ and } K2')$
20	C7	Control input. Normally used for R/W,I/O,etc
21	K1-	Clock input. $ITCY' = MTCY + (K1' \text{ and } K2')$
22	C6	Control input. Normally used for R/W,I/O,etc
23	WR-	Clock input. $MTCY = RD' + WR'$
24	C5	Control input. Normally used for R/W,I/O,etc
25	RD-	Clock input. $MTCY = RD' + WR'$
26	C4	Control input. Normally used for R/W,I/O,etc
27	A15E	Msb emulator address input.
28	ALE	Address Latch Enable for A0-A19.
29	INVI	Uncommitted inverter input.
30	INVO	Uncommitted inverter output.
31	DTCY	TCY clock delayed 100 ns.
32	CCK'	CONTROL byte input register clock (+ edge).
33	HACK'	HADRes byte input register clock.
34	MCK'	MISC byte input register clock.
35	TCY'	LADR,DATA,& HData input register clock.
36	ITCY'	Intel clock output. Jumper to TCY,CKs above.
37	MTCY'	Motorola clock output. Jumper to TCY,CKs.
38	ALE'	Inverter output from pin 28 input.
39	OE'	Output Enable' signal from ROM socket.
40	IDLE'	Not Used. Low when IDLE loop active. DMA?
41	CE'	Chip Enable' signal from ROM socket.
42	OEE'	Emulator Output Enable' (unlatched).
43	C3	CONTROL inputs. Normally A16-A19 from below.
44	C2	"
45	C1	"
46	C0	"
47	A19S	Latched emulator address signal (A19E).
48	A18S	"
49	A17S	"
50	A16S	"

Analyzer Cable Design

You can design your own cables for new processor types by copying and combining the techniques used in the cables shown in figure 7.1. The signal list for the analyzer connector immediately preceding this section should assist you further. While most of those signals are self explanatory, the analyzer input register clocking deserves some explanation. The analyzer logic and 3 of the 6 analyzer input bytes are clocked by the + edge of TCY' (pin 35). The other 3 input bytes (pins 32-34) are usually jumpered to this clock, but can be connected separately to clock their inputs at other parts of the clock cycle.

The diagram below shows the UniLab clocking logic:



The usual source of TCY' is ITCY' for Intel-type processors, or MTCY' for Motorola-types. MTCY' goes low whenever both RD- and WR- are high. By connecting these inputs to Motorola's E and VMA signals, the analyzer will be clocked on the falling edge of E if VMA is true. ITCY' goes low whenever RD- or WR- go low, or both K1- and K2- go low. Clocking in this case occurs at the end of the WR- or RD- pulse. DTCY is an inverted and 100 ns delayed version of TCY'. By using this signal to clock the CONTROL input register (CCK') the source of the clock (eg WR-) can be captured reliably as an analyzer input.

The ALE input (pin 28) controls transparent latches on the A0-A19 inputs to the emulator. The outputs of these latches are internally connected to the emulator and analyzer inputs, so clocking of the analyzer's address byte inputs can occur any time after they are stable. An inverted version of ALE is brought out on pin 38 so that inputs can be clocked by the end of ALE. An uncommitted inverter is also provided at pins 29-30 for processors, like the Z8000 & 16032, which use a low-true ALE signal.

signal.

The emulator output is enabled whenever the OE' signal on the ROM socket goes low and the 20 address bits satisfy the =EMSEG and EMENABLE statements which have been made. It is assumed that address decoding is done in the standard way using the CE' input to the ROM with OE' as a general memory enable. It is possible that some systems will make use of the OE' signal for address decoding. If this is the case, and you want to emulate several ROMs, you will have to change the jumpering to pin 42 to connect it to a more general enable' signal (such as MRDC' on Intel systems).

If there are I/O ports in the system which may have addresses which fall within the emulated address range, you must be sure that this enable excludes I/O operations. If no such signal exists, you can use an unused address input. For example, if the A19 input is connected to an IO/M' signal, and you specify 7 =EMSEG, the emulation memory will only be enabled when A19 goes low (memory cycles). Remember that =EMSEG acts as a modifier for all EMENABLE statements which follow, so if =EMSEG is changed the EMENABLEs must also be restated.

The descriptions which follow refer to the analyzer cables whose schematics appear at the end of this chapter.

(a.) Intel 8085, 8088-min mode, 80186, 80188, NSC-800: The CONTROL byte input register clock (pin 32) is isolated from the other input clocks and connected to ALE' (pin 38). This causes S0-2 to be clocked at the end of the ALE signal. All other input clocks (pins 33-36) are jumpered to ITCY' (pin 36), so that clocking will occur at the end of a low pulse on RD' or WR' or INTA'(K1-). Since K2- must be held low, it is connected to RES OUT on the 8085. C6 is internally connected to K1- to identify interrupt cycles.

To provide fewer cable variations, the 16-bit Intel processors all use the K1- & K2- inputs to derive a read and INTA clock by gating DT/R' and DEN' together. The write clock for the expanded mode can then come from either IOWC' or MWTC' at the bus controller. Note also that K1- is jumpered to C6 at the cable connector so that (DT/R') can be used both as a clock and an analyzer input without two separate connecting wires. A0 need be connected only if you have a 16-bit processor, as it is connected directly by the 8-bit ROM cable.

(b.) Motorola 6800, 6801/3, 6802/8, 6805E2, 6809: The UniLab address latch enable and CCK' pin are jumpered to the inverted DTCY signal so that control signals and addresses will be latched 100 ns after the rise of the E clock signal. This prevents trouble from the extremely short hold time of the address signals. Also note that the analyzer is clocked by the MTCY signal on the fall of E.

(c.) National 32016, Zilog Z8001, Z8002: The ALE signal is inverted, using the uncommitted inverter on pins 29 & 30. The control signals are latched at the end of the address strobe.

(d.) Motorola 68000, 68008, TI 9900, 99000, Z-8 romless, Intel 8085: A very simple configuration with all analyzer inputs clocked by ITCY.

(e.) Zilog Z-80, Z-8 piggyback, Intel 8051, 8031: WR', M1', and IORQ' are all used for clocking and captured by the analyzer to identify the cycle type. C5, C6, & C7 are therefore jumpered to WR-, K1-, & K2- so that a single probe can be used to make both connections. All analyzer inputs are clocked by DTCY so that the source of the clock will be captured. The address latches are enabled by DTCY' to prevent trouble from the short address hold time on Z-80A' B' & C' instruction fetches.

For the Z-80 only, A19 is connected to MREQ' so that OE' on the ROM socket needn't include an I/O term. Because of this you must use "7 =EMSEG" to get enable only when this signal is low.

(f.) Universal ROM-clocked: The OE' signal at the ROM is jumpered directly to the (RD-) clock input. C5 is jumpered to WR- and C6 is jumpered to K1- so that if a clock signal is connected to either of these leads, the signal will be captured by the analyzer without a separate connection. To reliably capture that input, the CONTROL byte input clock (CCK) is connected to DTCY. The address latch enable is connected to DTCY through the uncommitted inverter to prevent trouble from short address hold times. Since CE' at the ROM socket is connected to A16, you must use "E =EMSEG" to get enable when this signal goes low. You can make the UniLab ignore this signal by entering "F =EMSEG" then the EMENABLE statement, then "ALSO E =EMSEG" then repeat the EMENABLE statement.

(g.) RCA 1802: The TPB signal is used to clock the control inputs while the analyzer is clocked by MRD or MWR. Since the UniLab address latches cannot be separated, the MSB addresses must be connected to the target address latch outputs.

(h.) Intel 8080: The \emptyset 2TTL clock signal is taken from pin 6 of the 8224 clock generator. The DBIN' signal is inverted and connected to K1-. The analyzer clock function is thus \emptyset 2.DBIN + WR. The MEMWR- signal at the 8228 bus controller is used as an emulator enable. I/OW-, MEMR-, MEMW-, and INTA- are connected to C7 thru C4 so that the left digit of the analyzer control column will identify the cycle types as follows: F=I/OR, B=MEMR, D=MEMW, E=INTA, 7=I/OW.

(i.) Intel 80286: The ALE signal from the 82288 is jumpered to the CONTROL clock input so that S0 and S1 will be captured.

(k.) Rockwell 65/11 piggyback: Connects C7 to OE' which is R'/W, inverts CE and connects it to A15. A15 can be jumpered at the end of the cable to A12-14 for true address display.

(l.) Intel 8088/8086 max mode: Identical to cable A except that the uncommitted inverter is used to invert DEN signal. This inverter output is jumpered to the K1- input. Connect the DEN wire to pin 16 of the 8288 bus controller. A0 need be connected only if you have a 16-bit processor, as it is connected directly by the 8-bit ROM cable.

(m.) 6805 piggyback: Since no bus clock signal is provided by the processor, a circuit board is provided that derives clock from the signal at the crystal. This circuit includes a 74HCT74 CMOS divide-by-4 counter, which is reset whenever a transition occurs on the A0 signal. This reset ensures that the analyzer clock will be in sync with the internal processor clock.

(n.) 6801 piggyback, 6301: Identical to cable K except that the clock polarity is reversed.

(p.) 68000, 68008: Identical to cable D except that the OEE' input is grounded so that emulation will be enabled when either half of the data bus is read.

Note that some of these diagrams are untested and are provided only to help you get started. If you find any errors, please report them to us so that others can benefit from your discovery.

Also note that, since +5 volts is available at the connector, it is possible to make cables with logic gates on them if necessary. If you want to make a more conventional processor-specific emulator plug that plugs strictly into the processor socket in the target system, the RDD signal on pin 15 can be used with an OR gate to disable the RD- strobe at the processor when the emulation memory is active. This makes it unnecessary to unplug any ROMs that are being emulated, so all UniLab connections from both connectors could be made directly to a piggy-back processor with all signals except the RD' strobe directly connected. Of course this sacrifices universality and some transparency, but it might be more convenient in some situations.

The ROM Cable

There are 4 types of standard ROM cables:

C8-24. For 8-bit processors and 24-pin PROMs (2716,2732).

C8-28. For 8-bit processors and 28-pin PROMs (2764,128,256).

C16-24. For 16-bit processors and 24-pin PROMS.

C16-28. For 16-bit processors and 28-bit PROMS.

The C8-24 has an A11 pin, which can be left plugged into the A11 receptacle on the ROM cable if you are using 2732s or 32K ROMs. If you are using 16K ROMs, the receptacle must be plugged into the proper pin on the DIP clip to pick up the A11 signal at the processor. Other MSB address signals are likewise connected to the processor. Pin numbers for making these connections to the major processors are shown in the table in the previous section. Note that 24-pin cables will work fine in 28-pin ROM sockets if they are plugged in leaving the pin 1 & 2 end of the socket open. Extra address signals are simply picked up at the processor.

To minimize interconnections and signal loading, the analyzer data and address connections are taken from the ROM cable also. If your system has a unidirectional buffer between the ROM socket and the processor, these connections will not show data during write cycles. You can correct this condition by cutting the jumper ribbon cable on your ROM cable and installing a separate ribbon cable to the analyzer inputs on pins 35-42 (also 27-34 for 16-bit). You can order a cable that makes all connections at the processor by just ordering a C8-D or C16-D.

Note that all ROMs that are simulated must be removed from their socket to prevent bus contention. The ROM cable plugs into only one of the sockets-- except in the case of 16-bit systems, where there must be a second ROM plug in one of the most-significant-byte ROMs. In 8-bit systems the most-significant data bits are brought out in a separate cable, so they can be used as extra general-purpose analyzer inputs.

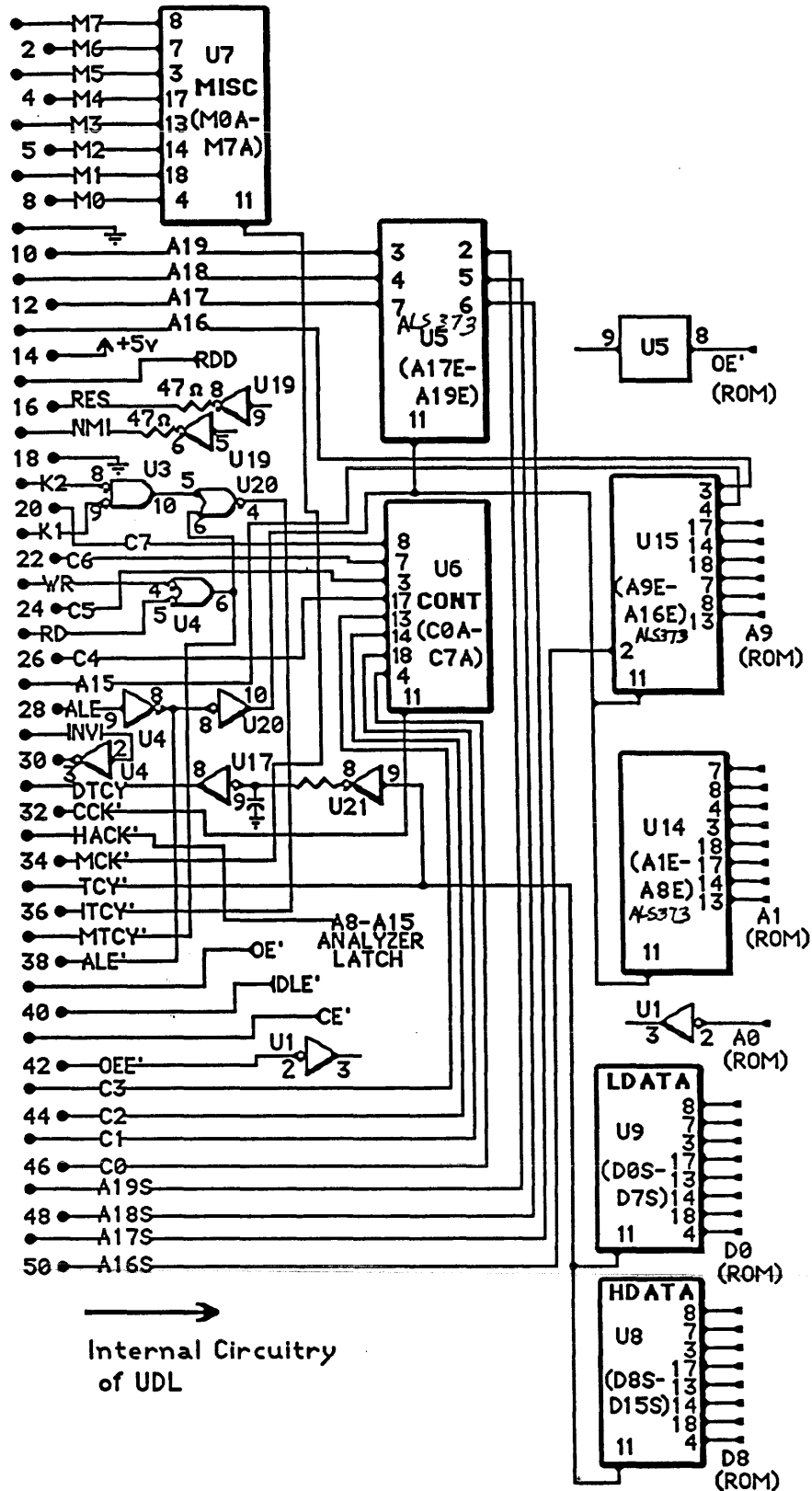
ROM Connector Signals

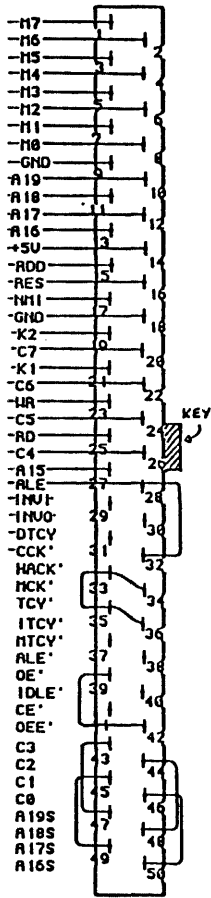
PIN#	SIGNAL	REMARKS
1	A14E	Direct connect on 256K ROMs. (lower left pin)
2	A12E	Direct on 64K or larger ROMs.
3	A13E	Direct on 128K or larger ROMs.
4	A7E	Emulator address inputs.
5	A8E	"
6	A6E	"
7	A9E	"
8	A5E	"
9	A11E	Direct on 32K or larger ROMs
10	A4E	Emulator address input.
11	OE'	To pin 39 on analyzer connector (for jumpering)
12	A3E	Emulator address input.
13	A10E	"
14	A2E	"
15	CE'	To pin 41 on analyzer connector (for jumpering)
16	A1E	Emulator address inputs.
17	A0E	"
18	GND	Signal ground. Shields adr inputs from data out.
19	D7E	Emulator data output. (odd addresses)
20	D6E	"
21	D0E	"
22	D5E	"
23	D1E	"
24	D4E	"
25	D2E	"
26	D3E	"
27	D11E	(even adr. LSB & MSB byte paralleled for 8-bit)
28	D10E	"
29	D12E	"
30	D9E	"
31	D13E	"
32	D8E	"
33	D14E	"
34	D15E	"
35	D7A	Analyzer data inputs. Usually jumpered to DnE.
36	D6A	"
37	D0A	"
38	D5A	"
39	D1A	"
40	D4A	"
41	D2A	"
42	D3A	"
43	D15A	(LSB & MSB byte paralleled for 8-bit)
44	D14A	"
45	D8A	"
46	D13A	"
47	D9A	"
48	D12A	"
49	D10A	"
50	D11A	"

Note: MSB and LSB are swapped for Intel convention. e.g. D8 above is really D0, etc.

UniLab Circuitry

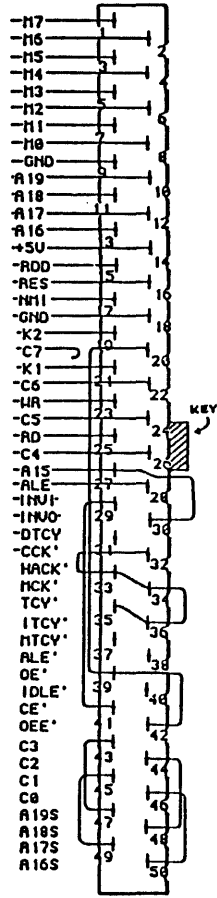
This page shows part of the internal schematics of the UniLab-- the input circuitry. The combination of this schematic and the diagrams that follow, showing the internal jumpers of all our standard cables, should give you enough information to customize a cable.





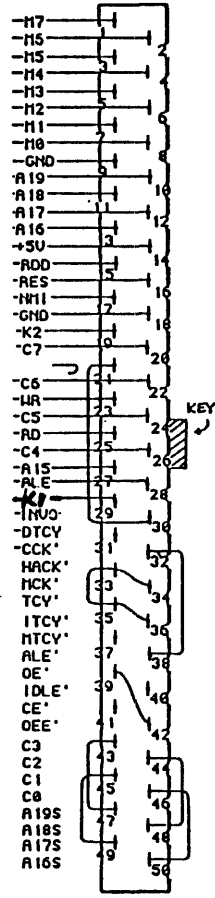
CABLE I

FOR 80286
(w/82189)



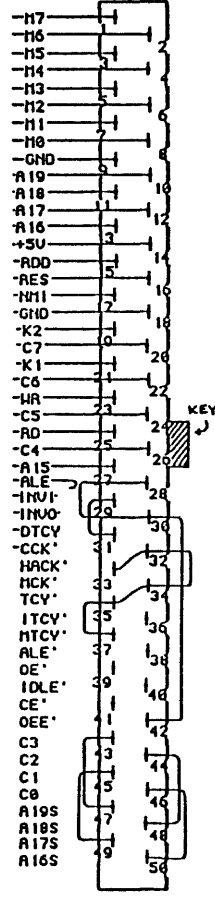
CABLE K

FOR 6511 P-Back



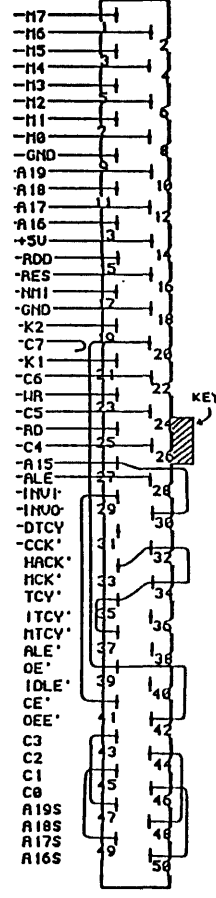
CABLE L

FOR 8086 max mode
8088 max mode



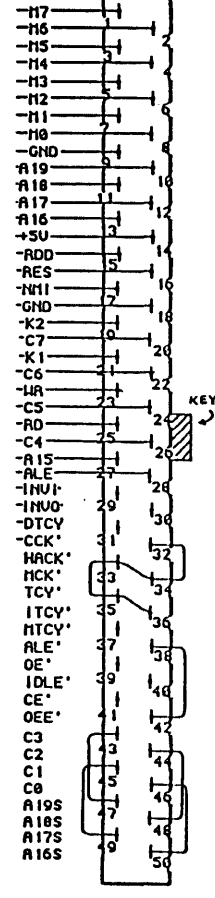
CABLE M

FOR 6805 P-Back
(w/clk.ckt.)



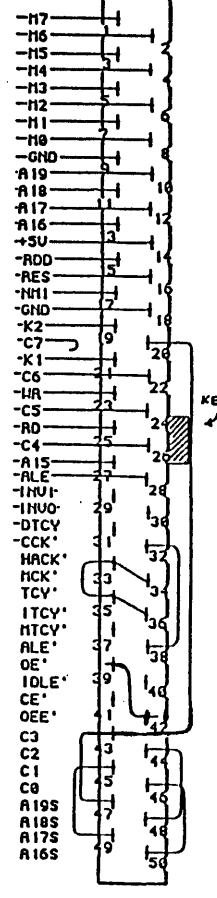
CABLE N

FOR 6801 P-Back
C901, C805



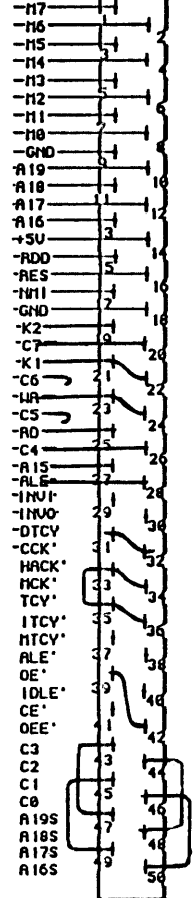
CABLE P

FOR 68000,
68009



CABLE Q

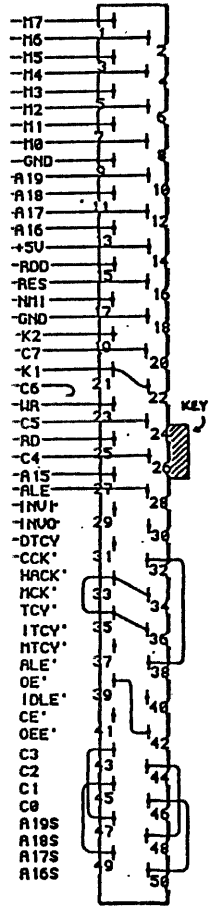
FOR NSC-800



CABLE R

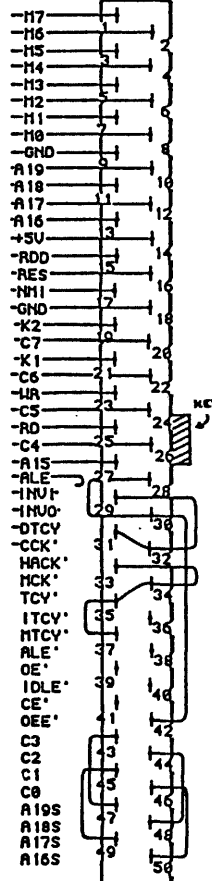
FOR 8096

(this page intentionally blank)



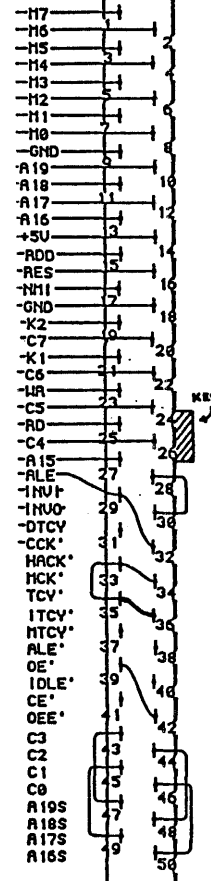
CABLE A

8085, 80186,
80188, 8086 min,
FOR 8098 min



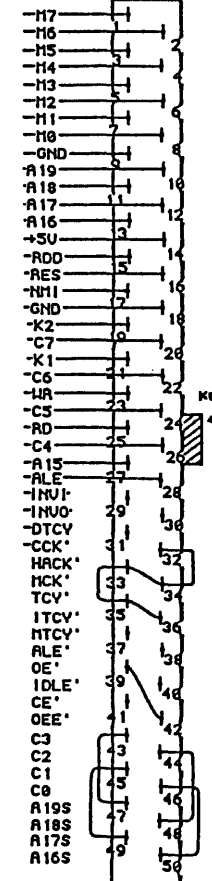
CABLE B

805/11A
6800, 6802,
FOR 6809, 6502



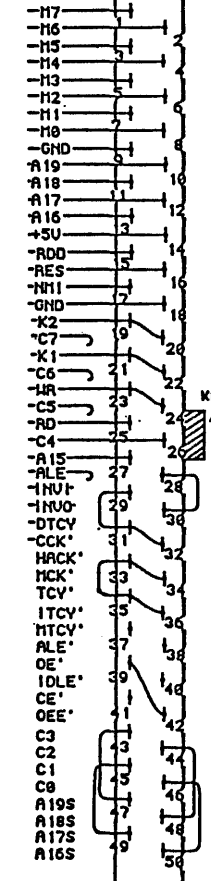
CABLE C

Z-8001/2,
FOR 16032



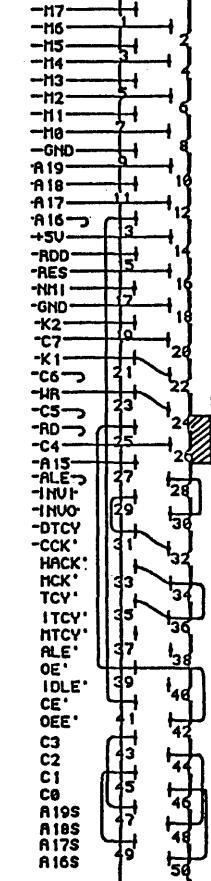
CABLE D

9900, 99000,
FOR Z-8+



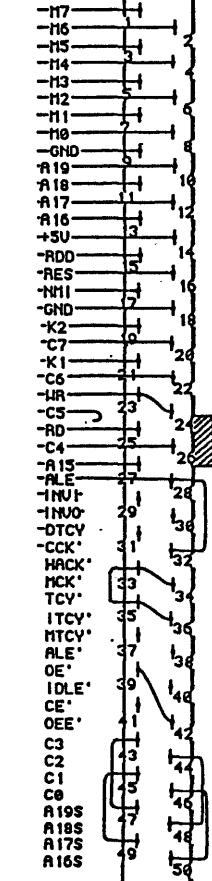
CABLE E

Z80, 8051,
8031, 8039,
FOR Z-8 P-PACK



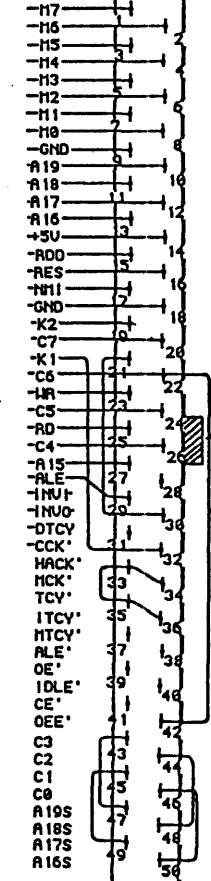
CABLE F

8048/35/39/
40/49/50
FOR (ROM Clocked)



CABLE G

FOR 1802



CABLE H

FOR 8080

Appendix E: UniLab II Specifications

Host Computer Interface

RS-232C connector, 19,200 or 9,600 baud,
switch selectable.

Diskette Formats

IBM PC 5 1/4", MS-DOS

Emulator

Download time: 1 second for 2K bytes, including 16-bit block error check.
195 ns max access time ROM emulation. (145 ns optional)
32K x 8-bit or 16K x 16-bit standard. Programmable by cable, program option.
Expandable to 128K bytes with optional plug-in board.
20-bit enable address decoding.
Individual 2K segments can be selected in any combination within 17-bit field.
Stand-alone operation possible as a ROM emulator.
16-bit Idle register loops target CPU allowing loading of emulation RAM and resumption of program execution.
Optional, target-processor-specific, software gives full debug capability including register and target memory display and change, breakpoints, and single-stepping.
Program loading software: from hex or binary disk files, hex serial download, memory image, ROM read.

Bus-State Analyzer

48-bit wide Trace Display and Memory.
48 data inputs. Two groups of 8 can be separately clocked.
6 clock signal inputs. Gated to form one bus clock:
Clock edge filter prevents re-trigger before 100 ns.
395 ns minimum bus cycle (10 MHz 68000).
297 ns with optional high-speed option.
Address demultiplexing latches included-- also used by emulator.

Analyzer Trigger

4-step sequential trigger.

RAM truth tables allow search for any function of 8 bits at each 8-bit group, for each step.

8 truth tables per step x 4 steps = 32 tables, each 256-bit.

16-bit inside/outside range detection on address lines.

4-bit segment enable gives 20-bit address capability.

Pass Counter: wait up to 65,382 events or cycles before 4th step.

Before/After/At Pass count trigger enable.

Delay Counter: wait up to 65,382 events or cycles to stop trace.

Filter feature: Records only cycles that satisfy trigger.

Oscilloscope sync output. (Sync on trigger.)

Interrupt output: Interrupt target on trigger (if enabled).

LED indicates searching for trigger. Stand-alone operation possible while waiting for trigger.

Software Features

Menu or command driven with single context for all four instruments:

- 48-Channel Bus State Analyzer

- In-Circuit Emulator

- PROM Programmer

- Stimulus Generator

Extensible macro capability.

Cursor key control of text and trace display.

Pop-up mode switch panel.

Split screen displays, user-definable.

On-line glossary.

Menu-driven shell displays equivalent command lines.

40 user-definable soft-keys.

On-line assembler.

Bonus features: Calculator, ASCII table, IC pinout library, memo message feature, direct DOS access, EGA/ECD support.

Software Options

Graphical Software Performance Measurement.

EPROM/EEPROM Programmer

Smart programming algorithm for high speed.

28-pin Textool zero insertion force socket handles 24 and 28 pin devices.

Programs single supply EPROMs and EEPROMs.

See Appendix G. Programs 2716, TMS2516, 2532, 48016, 2732A, 2764/128, 27256/64A/128A, 27512.

Signal Inputs

TTL logic levels (74ALS inputs).

0.1 ma maximum loading includes emulator & analyzer.

Signal Outputs

TTL logic levels (74LS244 outputs).

100 ohms forward terminating resistors on Emulator data lines.

Reset output (RES-): open collector, 7406 thru 47 ohms.

Interrupt output (NMI-): open collector, 7406, low true.

9 Stimulus outputs (at EPROM socket): 8255 NMOS outputs.

Physical

Size: 2.1" hi x 13" wide x 7.8" deep.

Weight: 4 lbs. (1.8 kg.)

Shipping Weight: 11 lbs. (5 kg.)

Fits easily in a slim-line brief case.

Power

100 kHz switching supply built in.

110V \pm 10% 50/60 Hz 15 watts (standard)

220V \pm 10% 50/60 Hz 15 watts (optional)

Accessories Included

User's Guide.
Reference Manual.
40-pin IC clip.
16-pin IC clip.
Input stimulus cable.
Component clip adaptor probes (2).
Jumper wiring tool.

Accessory Options

Personality Paks for most popular microprocessors include:
ROM Emulator cable. 8-bit, 24-pin version unless otherwise specified (C8-24).
Analyzer cable pre-configured for your target processor.
Disassembler/Debugger Software (DDB-xxx).

ROM Cable options:

8-bit, 28-pin ROM emulator cable	C8-28
8-bit, direct connect emulator cable	C8-D
16-bit, 2 x 24-pin ROM emulator cable	C16-24
16-bit, 2 x 28-pin ROM emulator cable	C16-28
16-bit, direct connect emulator cable	C16-D

RAM Expansion options:

32K emulation RAM expansion board (64K total)	EB-32
96K emulation RAM expansion board (128K total)	EB-96

Appendix F: Writing Macros

Introduction

You can combine several UniLab commands and give that combination of commands a new name. This is the simplest sort of macro you can make-- really more like a convenient abbreviation than like a real program.

The macro language included with the UniLab does have control structures that allow you to write more complicated macros. These structures are fully explained in several books (see page F-5). Fortunately, you don't need the control structures to write useful macros.

Contents

- How to Write a Macro
- Writing Macros on FORTH Screens
- Writing Test Programs
- Control Structures
- For the Experienced

How to Write a Macro

A macro definition begins with a colon (:) and ends with a semi-colon (;). The first word after the colon is the name of the macro-- the new abbreviation. All the other words are the commands that the new abbreviation stands for.

For example,

```
      : D10      DUP 10 + MDUMP ;
```

creates a macro called **D10**, as in **Dump 10** memory locations. This new word takes one argument, the starting address of the range that you want to dump. That address gets copied by **DUP** and then has **10** (hexadecimal) added to it. The macro then calls **MDUMP** on the address range.

Thus, if you were to type in:

```
      342 D10
```

then you would get a dump of addresses 342 through 352.

Writing Macros on FORTH Screens

The easiest way to test and alter macros is by writing them in files using the screen editor, and then loading the macro from a screen. Type **MEMO** to get a screen of the UniLab.SCR file.

After you type **MEMO**, press **CTRL-Z** to get the on-line prompts for the screen editor.

You exit from the screen editor by pressing the ESCape key twice in a row, or press ESCape followed by F to save any changes you made to the screen.

Loading macros from a screen

You can enter your macro onto the screen, and then use ESCape followed by L to load the contents of the screen. If you keep your macros on screens, you can easily alter or update them as the need arises.

Orion has set aside three screens of the UniLab.SCR file for your use-- the one you get with **MEMO**, and the two following. But don't use any screens besides these three, or you may overwrite help screens and error messages.

Writing Test Programs

You can use the macro capability of your UniLab system to write automatic test programs. In this section we will present some specific examples, which you can easily adapt to your specific needs.

A good starting test for a new system is to just let it execute no-op instructions. If the address bus has any shorts in it, will show as a departure from the normal address count sequence.

For example, let's assume a Z-80 or 8080 system with memory locations 0 to 7FF enabled. You can load the 00 no-op opcode into all enabled memory by just entering 0 7FF 0 MFILL. Now if you enter **STARTUP**, you should get a trace showing the first A6 addresses. By saving this trace with **TSAVE** and comparing it to the trace of an untested system, you can automate system checkout.

STARTUP and **S** are not suitable for use in automated test macros, since they cause the trace to be displayed till a key is pressed, another command must be used to start the analyzer without displaying the trace. That command is

<n> SC <file name>

will start the analyzer, wait n milliseconds, then compare the trace to a trace previously saved by **TSAVE**. It is very useful for automatic test sequences.

For instance, you enter

```
: TEST1 0 7FF 0 MFILL NORM A6 DCYCLES RESET 0 SC A:\Test.TRC ;
```

to define a startup test. This macro will fill the first 7FF memory locations with zeros-- no-op instructions-- then start the analyzer and compare the result to a trace that was saved as a file on drive A: using **TSAVE**.

A technician can then test the system by typing in **TEST1**. The UniLab system will reply with an OK message if the trace agrees with the one stored on the disk.

To define a test that will examine later addresses, you can enter

```
: TEST2 6FF ADR S 0 SC A:\TEST2.TRC ;
```

Of course, you will have to use **TSAVE** to save a reference trace.

If both tests work properly, we can combine them into a single test by entering:

```
: TEST ." Test 1:" TEST1 ." Test 2:" TEST2 ;
```

This defines a new word **TEST** that will execute the 2 tests in sequence, identify the tests, and display an OK message if they pass. (Note that `."` message" in a macro definition will print "message"). If either test fails, the program will automatically abort with the normal **TCOMP** fault display showing the faulty cycle and what it should have been.

Including messages in macros

If you want the operator to press a key, just put in a message to that effect by starting with `."` and ending the message with `"` as we did above.

You can then use the command **KEY** to wait for a keystroke. This word also leaves the ASCII code for the key on the stack. You can either get rid of it with **DROP** or use it as you wish. For example, you could use it to determine the next step to be taken:

```
: SIMPLE-TEST
  ." This is a simple test " CR
  ." Do you wish to continue?(y/n) " KEY
  ASCII y = IF REAL-TEST THEN
  ." Bye " ;
```

This new word, **SIMPLE-TEST**, will execute the word **REAL-TEST** if the user enters in a "y." Otherwise, it will fall through and print out the closing " Bye."

Removing faulty definitions

If you define a test that doesn't work, you can erase the definition from memory by entering:

```
FORGET <macro name>
```

FORGET will also forget any words that have been defined since the macro that you want to forget. For instance, **FORGET TEST2** will forget **TEST2** and any other words you have defined since **TEST2**. (You thus forget a whole sequence of words. Enter **VLIST** for a list of words defined-- last word first.)

Control Structures

Your system's macro capability as described so far is really just the tip of the iceberg.

A complete FORTH system is resident within the UniLab software. This language includes constructs such as DO ... LOOP, IF .. THEN ...ELSE, and BEGIN ... UNTIL, so you can define macro words that are much more complex than the simple examples we have covered.

If you want to learn more about FORTH, the best book by far is Starting FORTH, by Leo Brodie.

If you want to contact other FORTH users, try going through:

The FORTH Interest Group
P.O. Box 1105
San Carlos, CA 94070
(415) 962-8653

They have monthly meetings in many locations and publish an excellent journal called FORTH Dimensions.

The UniLab software was all developed using the MVP-FORTH PADS (Professional Application Development System). This package, and many other FORTH books and programs, is available from

Mountain View Press Inc.
P.O. Box 4656
Mountain View, CA 94040
(415) 961-4103

The public-domain portion of that system (which is a modified 1979-Standard FORTH system) is included with your UniLab. Excellent documentation for that system is included in a book by Glen Hayden called All About FORTH, which is a complete glossary of the FORTH words.

If you plan to use the FORTH capabilities, you should buy the manual for the PADS FORTH system. It is available from

FORTHKIT
240 Prince Ave.
Los Gatos, CA 95030

The manual includes source screens and documentation for many nice utilities included with the system.

You should also request from Orion the Orion Programmer's Guide (available in July 1986).

For the Experienced

If you already know FORTH, the rest of this section will point out a few useful details for you. If not, you can skip the rest of this section, because it probably won't make any sense to you. It is not necessary to learn FORTH to use the UniLab.

Redefinitions

Three standard FORTH words have been redefined in the UniLab system: **NOT** is used in trigger definitions so the synonym **O=** must be used: also **OUT** is redefined to **TOUT**. **CFA** has been redefined as **CFADR** to prevent conflicts with the hex number. Another difference to bear in mind is that the default number base in the UniLab system is Hex, while decimal is usually used in book examples.

Editor

There is a complete FORTH editor resident in your system, which can be used for writing FORTH programs, then compiling them from the screens by using the **n LOAD** command (or escape L). See **MEMO** in the **Command Reference** chapter for a little more information on the use of the editor.

The editor is a very fast full-screen editor designed for use with FORTH screens. To use it you enter

<n> EDIT

(where n is a numbered 1K block) at any time when the UniLab program is running. A summary of all the editor commands will be displayed if you enter control Z.

Normally, UniLab help screens file is open so that is what you will see, but if you use

OPEN <file name>

you can edit or examine any file in 1K pieces.

Assembler

There is also a complete, reverse polish, assembler for your host processor. We include it because it doesn't take up much space and it is useful if you want to use your system to write FORTH programs. Refer to the books listed above for more details.

The assembler is a complete (FORTH) 8086 assembler. It is loaded automatically whenever you enter **CODE**. This version was adapted for MVP FORTH by Tom Wempe from Ray Duncan's version published in Dr. Dobb's Journal, Number 64, Feb. 1982.

Decompiler

Your UniLab software also includes a simple decompiler. This is useful if you want to understand the functioning of any of the UniLab or FORTH words or recall one of your own word definitions. To use it enter

' <word> **XX**

where <word> is any word in the system. The decompiler will print the address, contents, and name of each word in the definition in sequence each time you enter **XX**.

For numbers it will print "LIT" and then the number in the next line with garbage in the 3rd column. Messages defined with "." will give garbage in the name column. Also headerless words will show junk in the name column. It's crude but amazingly effective for a definition that only occupies 12 bytes of object code!

Virtual files (.VIR)

The UniLab diskette includes three **.VIR** files: **EDIT.VIR**, **ASM.VIR**, and **UTIL.VIR**. These files are not needed to run the UniLab but you may find them useful for other work. The **EDIT.VIR** file is required to use the MEMO pad.

Some other virtual files are also used by the UniLab software. **ULxxx.VIR** contains trace routines and other material essential for the UniLab system to run.

MENxxx.VIR contains the menu overlay, the mode panels, and the symbol conversion menu. **L1Bxxx.VIR** and **L2Bxxx.VIR** contain the pinout libraries.

The names of these files will usually be coded with the version number, for example: **UL32.VIR**, **MEN32.VIR**, and **L1B32.VIR** for version 3.20. These version numbers **must** match the version number of the UniLab software or the system will crash.

The utilities are loaded from **UTILxxx.VIR** when needed by words like **start end TRIADS** or **from to copyto COPYSCRNS**. All of these commands are described in the documentation for the Mountain View Press PADS FORTH system.

**Appendix G:
EPROMs and EEPROMs Supported**

Part Number	PM	Vpp	Command	
			To Read	To Program
EPROMs				
2716	16		RPROGRAM	P2716
27C16	16		RPROGRAM	PD2716
2532	16		RPROGRAM	P2532 *
TMS2532	16		RPROGRAM	P2532 *
2732	32	21	R2732	P2732A
2732A ^b	32	21	R2732	P2732A
27C32	32	21	R2732	P27C32
27C32	32	25	R2732	P27C32 **
2764	64	21	RPROGRAM	PD2764
2764A	56	12.5	RPROGRAM	P2764
TMS2764A	64	21	RPROGRAM	P2764
27C64	64	21	RPROGRAM	P2764
27C64	56	12.5	RPROGRAM	P2764
27128	64	21	RPROGRAM	PD2764
27128A	56	12.5	RPROGRAM	P2764
27256	56	12.5	R27256	P27256 ***
27256	56-21 ^a	21	R27256	P27256 ***
27C256 ^c	56	12.5	R27256	P27256 ***
27512	512 ^a	12.5	R27512	P27512 (8BIT mode needs 64K UniLab 16BIT mode nees 128K)
EEPROMs				
4816 ^b	16		RPROGRAM	P48016
48016 ^b	16		RPROGRAM	P48016

NOTES:

- a Personality modules 56-21 and 512 are optional equipment.
- b Limited support only for 4816, 48016, and 2732A.
- c We do not support the Fujitsu MBM 27C256.

* The 2532 EPROM does not support the 16-bit mode of programming.

** Must cut pin 8 off of PM-32 for the 25-volt part.

*** You need a 64K UniLab to use 16BIT mode with the 27256 PROM.

Appendix H: Microprocessors Supported

The Orion UDL and UniLab can both be used with almost every microprocessor on the market.

Orion supports the more popular microprocessors with pre-configured cables and with processor specific software packages.

The information in this appendix is a guide to the level of support that we offer, as of November 26, 1986. It is not, however, exhaustive. As the Orion product line grows, we will continue to support more processors with cabling and software. Contact Orion Applications Engineering for the latest information.

3.20 Software Update:

All packages, except for Z8000, come with a line-by-line assembler.

All overlay and reserved areas are movable.

All software packages (except 1802) support the **NMI** features, some through the use of IRQ pin of the processor.

All software packages which support multiple processors provide you with a "patch menu." The is menu is presented to you when you call up the UniLab software, and is also available through the command **PATCH**.

1. FULLY SUPPORTED PROCESSORS:

1.A. NON-PIGGYBACK CHIPS

PROCESSOR	Analyzer Cable	Software Package	Reserved	Overlay
1802	G	DIS-18	N/A	N/A
1805	G	DIS-18	N/A	N/A
1806	G	DIS-18	N/A	N/A
2800 *	Q	DDB-Z80	38-3D	3D-
6301X0+	B	DDB-63	FFB9-BA	FFBC-EF
6303X	B	DDB-63	FFB9-BA	FFBC-EF
6303R	B	DDB-63	FFB9-BA	FFBC-EF
6305X2	B	DDB-685	1FB9-BA	1FBA-E3
6305Y2	B	DDB-685	1FB9-BA	1FBA-E3
6500/1E	B	DDB-65	FFAC-BC and 40	FFBD-FB
6502	B	DDB-65	FFAC-BC and 40	FFBD-FB
65C02	B	DDB-65	FFAC-BC and 40	FFBD-FB
65C102	B	DDB-65	FFAC-BC and 40	FFBD-FB
65C112	B	DDB-65	FFAC-BC and 40	FFBD-FB
6512	B	DDB-65	FFAC-BC and 40	FFBD-FB
R6511Q	B	DDB-65P	FFAC-BC and 53	FFBD-FB
6800	B	DDB-68	FFB9-BA	FFBC-EF
6801+	B	DDB-681	FFB9-BA	FFBC-EF
6802	B	DDB-62	FFB9-BA and 50-53	FFBC-EF
6803	B	DDB-681	FFB9-BA	FFBC-EF
6805E2	B	DDB-685	1FB9-BA	1FBA-E3
6805E3	B	DDB-685	FFB9-BA	FFBA-E3
6808	B	DDB-682	FFB9-BA and 50-53	FFBA-F7
6809E	B	DDB-689	FFB9-BA	FFBB-
68HC11	B	DDB-611	FF71-73	FF74-BF

+ designates "expanded mode."

* Also called an NSC-800+.

1.A. FULLY SUPPORTED NON-PIGGYBACK CHIPS (continued)

PROCESSOR	Analyzer Cable	Software Package	Reserved	Overlay
68000	P	DDB-68K	07A8-AB	07AC-FC
			and a trap vector	
68HC000	P	DDB-68K	07A8-AB	07AC-FC
			and a trap vector	
68008	P	DDB-688	07A8-AB	07AC-FC
			and a trap vector	
68010 **	P	DDB-68K	07A8-AB	07AC-FC
			and a trap vector	
68020 **	P	DDB-68K	07A8-AB	07AC-FC
			and a trap vector	
8031+	E	DDB-51	FFC0-C6	FFC7-???
80C31+	E	DDB-51	FFC0-C6	FFC7-???
8032+	E	DDB-51	FFC0-C6	FFC7-???
8035+	E	DDB-48	07FA-FF	07DE-FA
8039+	E	DDB-48	07FA-FF	07DE-FA
8040+	E	DDB-48	07FA-FF	07DE-FA
8044+	E	DDB-48	07FA-FF	07DE-FA
8048+	E	DDB-48	07FA-FF	07DE-FA
8049+	E	DDB-48	07FA-FF	07DE-FA
8051+	E	DDB-51	FFC0-C6	FFC7-???
8052+	E	DDB-51	FFC0-C6	FFC7-???
8080	H	DDB-85	008-011	012-
8085	A	DDB-85	008-011	012-
8086	A	DDB-86	FFFB1-B2	FFFB3-
			and 00004-0F	
80C86	A	DDB-86	FFFB1-B2	FFFB3-
			and 00004-0F	
8088	A	DDB-88	FFFB1-B2	FFFB3-
			and 00004-0F	
80C88	A	DDB-88	FFFB1-B2	FFFB3-
			and 00004-0F	

- + designates "expanded mode."
- * Uses a register as well-- 50H.
- ** In 68000 mode only.

1.A. FULLY SUPPORTED NON-PIGGYBACK CHIPS (continued)

PROCESSOR	Analyzer Cable	Software Package	Reserved	Overlay
8086+	L	DDB-86	FFFB1-B2	FFFB3- and 00004-0F
80C86+	L	DDB-86	FFFB1-B2	FFFB3- and 00004-0F
8088+	L	DDB-88	FFFB1-B2	FFFB3- and 00004-0F
80C88+	L	DDB-88	FFFB1-B2	FFFB3- and 00004-0F
8094	R	DDB-96	* 2016-1C	201D-4F
8095	R	DDB-96	* 2016-1C	201D-4F
8096	R	DDB-96	* 2016-1C	201D-4F
8097	R	DDB-96	* 2016-1C	201D-4F
80186	A	DDB-86	FFFB1-B2	FFFB3- and 00004-0F
80188	A	DDB-88	FFFB1-B2	FFFB3- and 00004-0F
80286	I	DDB-86	FFFB1-B2	FFFB3- and 00004-0F
8344	E	DDB-51	FFC0-C6	FFC7-???
HD64180	E	DDB-Z80	38-3D	3D-
NSC800	Q	DDB-Z80	38-3D	3D-
Z8800	D	DDB-S8	0781-85	0786-FF
Z8681	D	DDB-Z8	and 60 07AF-B3	07B4-
Z8682	D	DDB-Z8	and 60 07AF-B3	07B4-
Z-80	E	DDB-Z80	and 60 38-3D	3D-
Z8001	C	DDB-Z8K		700-
Z8002	C	DDB-Z8K		700-
Z8003	C	DDB-Z8K		700-
Z8004	C	DDB-Z8K		700-

*** Must use =RAM.SEGMENT and =ROM.SEGMENT with these two chips.

1.B. FULLY SUPPORTED PIGGYBACK CHIPS

PROCESSOR	Analyzer Cable	Software Package	Reserved	Overlay
6301	N	DDB-63	FFB9-BA	FFBC-EF
6305Y1	M	DDB-685	0FB9-BA	0FBA-E3
6500	K	DDB-65P	FFAC-BC and 40	FFBD-F8
6511EB	K	DDB-65P	FFAC-BC and 40	FFBD-FB
R6541	K	DDB-65P	FFAC-BC	FFBD-FB
6801	N	DDB-681	FFB9-BA	FFBC-FD
68P05V07	M	DDB-685	0FB9-BA	0FBA-E3
68P05W0 *	M	DDB-685	0FB9-BA	0FBA-E3
8040	F	DDB-48	07FA-FF	07DE-FA
8048 **	F	DDB-48	07FA-FF	07DE-FA
8049 **	F	DDB-48	07FA-FF	07DE-FA
8050 **	F	DDB-48	07FA-FF	07DE-FA
80C51	E	DDB-51P	FFC0-C6	FFC7-???
Z8822	D	DDB-S8	0781-85 and 60	0786-FF
Z8603 ***	E	DDB-Z8	071F-B3 and 60	07B4-

* 6805 with A/D converter.

** Also called the 87P50, by National Semiconductor.

*** Same configuration also supports the Z8612 and Z8613.

2. OTHER PROCESSORS FOR WHICH ORION PROVIDES ANALYZER SUPPORT

2.A. NON-PIGGYBACK

Processor	Cable
16032	c
9900	d
99000	d
TMS7000	m

2.B. PIGGYBACK

Processor	Cable
COP420	f
3870	m
3873	m

3. PROCESSORS THAT ORION SUPPORTS INDIRECTLY

There are some processors that you cannot hook up to the UniLab or UDL. However, Orion supports these processors indirectly, by supporting processors that are opcode compatible with them.

Thus you can use a processor supported by Orion for the development work.

<u>Processor</u>	<u>Use for development work</u>
1804AC	1805 or 1806
6301U0	6305X2
6301V0	6301 piggyback
6301V1	6301 piggyback
63L05F1	6305E0
6305V0	6305X2
6305X0	63P05Y0
6305X1	6305X2
6305Y0	63P05Y0
6305Y1	6305Y2
6500/1	6500/1E
6500/11	65/11EB
6500/12	65/11EB
6500/15	65/11EB
6500/16	65/11EB
6500/41	6541
6500/42	6541
6503	6502
6504	6502
6505	6502
6506	6502
6507	6502
6513	6512
6514	6512
6515	6512
CPD68HC04P2	6805 piggyback
MC68HC04P2	6805 28 pin
CPD68HC04P3	6805 piggyback
MC68HC04P3	6805 28 pin
6805F2	6805 piggyback
6805G2	6805 piggyback
6805S1	6805 piggyback
6805S6	6805 piggyback
6805U1	6805 piggyback
6805V1	6805 piggyback
6805W1	6805 piggyback with A/D converter
8048	8048 piggyback
8051	8050 piggyback

3. PROCESSORS THAT ORION SUPPORTS INDIRECTLY (Continued)

Processor	Use for development work
8393	8096
8394	8096
8395	8096
8396	8096
8748	8048 piggyback
8793	8096
8794	8096
8795	8096
8796	8096
9761H	80C51 piggyback
Z8010	Z8012 piggyback
Z8011	Z8013 piggyback
Z8020	Z8022 piggyback
Z8021	Z8023 piggyback
Z8030	Z8032 piggyback
Z8031	Z8033 piggyback
Z8601	Z8603 piggyback
Z8611	Z8613 piggyback
Z86C11	Z8613 piggyback

Appendix I: System Messages

Contents:

1. ERROR AND STATUS MESSAGES
2. PARAMETER ENTRY MESSAGES
3. MENU MESSAGES

1. ERROR AND STATUS MESSAGES

<number> ? - PROM programmer error. Usually this means an RS-232 error.

...enter INIT - The UniLab needs to be initialized with the host computer. You should enter **INIT**, then proceed.

Address entry error. Needs 'address-start address-end' command.
- Insufficient parameters given to a command.

bad table file - Your assembler table file has been corrupted (the file called xxx.TBL). Try copying it from the distribution diskette again.

beyond blkmax - DOS tried to read a forth screen file beyond the limit set in the variable BLKMAX. This is mainly protection against accessing files on a hard disk as forth blocks.

beyond eof - DOS tried to read a file beyond its end of file.

boundaries for bins overlap - An error message from the histogram producer (**AHIST** or **THIST**). The program will not produce a histogram until this error is fixed. It occurs if any two ranges of addresses or times share a region. For example,
1000 - 2000 and 1500 - 2500
or even just
1000 - 2000 and 2000 - 3000.

bytes truncated, beyond 64K memory boundary. - A load was attempted that tried to load data beyond FFFF.

Can't Call DOS - Attempt to call DOS failed, probably because you are in the menu mode. If COMMAND.COM is not on your root directory, you will get this message.

Can also result from having the setting of `files=` in your CONFIG.SYS file too small. See the Installation chapter.

Can't find GLOSS.ORI - The file GLOSS.ORI has to be in the directory pointed to by the DOS environment string `GLOSSARY` in order to use LOOKUP and WORDS. See the Installation chapter.

Can't find GLOSS.TXT - The file GLOSS.TXT has to be in the directory pointed to by the DOS environment string `GLOSSARY` in order to use LOOKUP and WORDS. See the Installation chapter.

Can't find GNames.ORI - The file GNames.ORI has to be in the directory pointed to by the DOS environment string `GLOSSARY` in order to use LOOKUP and WORDS. See the Installation chapter.

Can't find xxxx - File could not be located on the disk or directory. Often caused by not having the proper values assigned to `ORION` and `GLOSSARY`. See the Installation chapter, Software Installation.

Can't open overlay. - Your `.COM` file and your `.OVL` file do not match up. Probably you have the command file from one version of the UniLab software and the overlay file left over on your disk from an older version.

Can't R/W non-emulated address without working Debugger control!
- An attempt to use MDUMP, M, MM!, MCOMP, MFILL etc., on memory that is not in emulation memory. Debug control has not been established via RB or NMI. Once you have established debug control, you can read and alter target system RAM as well as emulation ROM.

Can't use ALSO to add another ADR range. - The range trigger specs for the UniLab can get very complicated. The low- and high- order addresses are not intrinsically linked in the truth tables. Multiple ranges would probably yield a lot of triggers that would be combinations of wrong high and low address. For this reason, multiple ranges are not allowed.

compile only - See All About FORTH (see page F-5).

conditionals not paired - See All About FORTH (see page F-5).

Data is <data> at addr <adr> ..but is <data> at addr <comp adr>-
Displayed when MCOMP is used and the memory does not match up.

definition not finished - See All About FORTH.

Disk full - The disk cannot hold any more files-- use a fresh diskette, or remove some files that are no longer needed.

diskerr # n - There is something wrong with a DOS disk operation. Consult your DOS manual for the error number's meaning.

divide overflow - Arithmetic error caused by dividing a number by zero.

eadr ng - End Address Error. An error in an internal command sent to the UniLab. Usually caused by an RS-232 error or a general system failure.

empty stack - The operating system is trying to use a number on the FORTH stack, and the FORTH stack is empty.

Emulator Memory Enable Status - A message displayed before the status of emulation ROM is displayed.

End of DOS file reached before formatted end-of-file. - Your Intel hex format file being loaded by HEXLOAD contains bad data, or lacks the checksum at the end.

end of text file - The end of the file being displayed by TEXTFILE has been reached.

End of Trace Buffer - The display has come to the end of the trace buffer. Use **TT** to start from the top, or <n> **TNT** to start from cycle number n.

Enter <parameter description> - A menu prompt, describing to you the number(s) required by that menu item. See Section 3 of this appendix.

Eprom programmed and verified, finished... - A status message from the EPROM programmer. Everything is fine.

Eprom VERIFY Error !!! - A status message indicating that there has been a problem while programming your EPROM.

file access error - The on-line assembler encountered a problem while trying to read or write or close a file. The most likely cause: you do not have the .TBL file in the correct directory. It should be in the directory pointed to by the DOS environment string **ORION** (SET ORION=????). See the **Installation** chapter.

File Name? --- Prompt requesting the name of the file to be opened or saved. This only appears if you do not include the file name in the same line as the command to open or save the file.

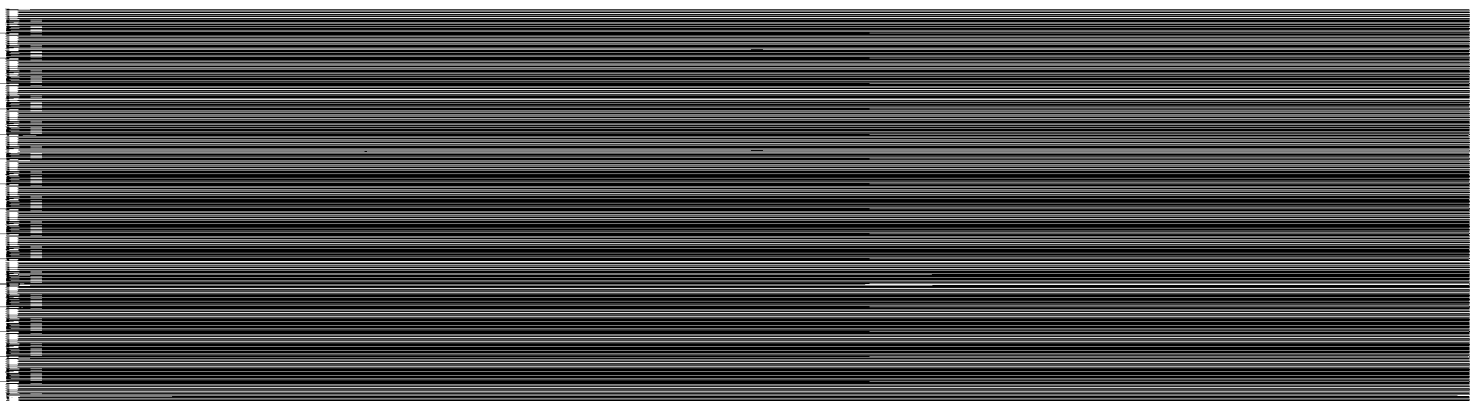
full-stack - Internal stack is overflowing. If this occurs, it might be that there is not enough room in the system. Possible cause: many user macros.

Hardware errors in Emulation Memory - While attempting to determine the memory size of your UniLab's emulation memory, the software has detected a hardware fault in the UniLab.

Hit any key to return to menu - If a menu operation uses the entire screen, the menu selections will be overwritten. This message will be displayed, letting the user return to the menu display.

in protected dictionary - You cannot FORGET words that are in the protected dictionary. See All About FORTH.

Initialization request refused - a check of the UniLab's PROM has found that it is not an Orion Instruments product.



Initializing UniLab Hardware errors in Emulation Memory - while attempting to determine the memory size of your UniLab's emulation memory, the software has detected a hardware fault in the UniLab.

input > 255 - see All About FORTH.

input stream exhausted - see All About FORTH.

Invalid number - an error message from the histogram producer (AHIST or THIST). You produce this error if you try to enter a value that is not a number in the base you are using. For example, FF is not a number in decimal. You will not be able to produce a histogram until you correct the mistake.

Invalid start and stop address for THIST - an error message from the histogram producer THIST. This error tells you that one of the two addresses that you gave to THIST is missing or is not a number in the base you are using.

isn't unique - the word used as a macro name is already used. This won't hurt anything except you can no longer use the previous word.

ladr ng - Load Address Error. An error in an internal command sent to the UniLab. Usually caused by an RS-232 error or a general system failure.

len ng - Length of data transmitted is bad. An error in an internal command sent to the UniLab. Usually caused by an RS-232 error or a general system failure.

loading only - see All About FORTH.

Lowbound is larger than highbound - an error message from the histogram producer (AHIST or THIST). This error occurs if a bin has a starting value that is higher than the ending value. You cannot make a histogram until you fix this error.

max 3 qualifiers - You cannot enter more than three qualifiers with the AFTER command.

MISC inputs cannot be used in qualifier or filter specs. - The MISC lines are excluded from being used in an AFTER or an ONLY trigger specification.

needs <number> parameters - The command needs more parameters than were given. Consult the glossary to see what commands the word needs.

No Analyzer Clock - The UniLab is not receiving a clock signal. All clocking is through the RD-, WR-, K1-, and K2- lines to the UniLab (except for the F cable, which gets clock from the piggyback rom socket). The processor might be stopped, or these four lines might not be correctly connected.

no drive - The disk drive you tried to access does not exist. Usually received if you enter a command such as **BINLOAD B:myprog.bin** when you have no drive B.

No Good! (Above is correct.) Was: - Message produced by TCOMP when it detects a difference between the trace on the diskette (which it assumes is correct) and the trace that has just been made, which is sitting in host memory.

No Memory Enabled - No emulation memory is set up to be selected as ROM for the target board. This is not an error.

no room - Dictionary is full and won't hold any more macros. Use FORGET to forget unused macros before adding new ones.

no trigger! - Only shows on a compare trace with the SC command. This means that the trigger spec was not reached within the time you specified.

Not a DOS text file - TEXTFILE is trying to read in a file that is not a DOS text file.

Not available in menu mode - Some commands are restricted when using menus, such as calling up the HELP displays. These would overwrite the menu if they were permitted.

Not done till delay count = - The UniLab is waiting for enough events to fill up its trace buffer. If too few bus cycles occur, or if filtering is being used, this message will be generated. You can wait, or you can hit any key to break out of this state. You will then need to type **TD** (Trace buffer Dump) to see the filtered cycles or the cycles before the clock was stopped in the buffer. Note that the cycles will appear at the end of the trace buffer, while there will still be left-over garbage toward the beginning of the trace.

not enb. - Emulation memory is not enabled for this data transfer-- not an error message, but a reminder that the address you are referencing is not enabled (for example, target system RAM).

Not enough bins available - An error message from the histogram producer (**AHIST** or **THIST**). This error occurs if you try to allot more than 15 bins using the Subdivide key (**F3**). This can occur if you already have several bins allotted and then try to subdivide one of them among all the bins.

not enough memory - The on-line assembler (**ASM** or **ASM.F**) can't allocate enough RAM to read the table file in. Use **?FREE** to check on the amount of free RAM available, and then use either **=HISTORY** or **=SYMBOLS** to reduce the amount of RAM dedicated to those two space hogs. You will have to **SAVE-SYS** and then restart the UniLab software before the new settings will take effect. Look up the appropriate entries in the **Command Reference** chapter.

not found - Word used in macro definition does not exist. Check spelling. Note that macros may use other macros, but each word in a macro must be already defined. Also used to let you know that the file was not found by **SYMLOAD**, or that the file was not the right type.

not in dictionary - **HELP** cannot find the word in the UniLab glossary.

not recognized - Command or word not available. This is usually a misspelled word. If it is a known word that should exist, type **BYE** to exit the system. Do not **SAVE-SYS**, since the operating system may be damaged.

ODD or EVEN ? - A prompt to tell the user that either the address range must be given (with an implicit odd or even start address)

ODD or EVEN ? - A prompt to tell the user that either the address range must be given (with an implicit odd or even start address) or the command ODD or EVEN must be used when reading or programming a 27512 in the 16-bit mode.

ok - Word returned by the operating system every time it accepts a command.

out of bounds - Stack underflow-- a catastrophic error. If it occurs, exit system and re-boot. Do not **SAVE-SYS**.

parse error - The on-line assembler does not recognize your assembly language command.

rcv sum ng!! - An error occurred when a HEXFILE was being received. This is usually an indication of an RS-232 problem.

Reading.... - Message displayed when reading a 27512 prom, since it takes a few minutes.

reading text...please wait - TEXTFILE reads in a file to the UniLab program and analyzes it for the number of lines. While it is creating a line index, this message is printed out.

Requires <parameter description> - A parameter entry error message. Indicates that the wrong number of parameters was given to a command. See Section 2 of this appendix.

resetting - The analyzer has been started, and the reset line toggled low then high again, to reset the target processor. Look at, chapter 6, Section 4.5.

RS232 err # n ...enter INIT - The RS-232 communication between the host and the UniLab is not functioning properly. Check the cable hookup. If you have changed baud rate, make sure that the software setting corresponds to the switch setting in the UniLab.

STANDALONE mode.... Use PROMMSG later to communicate with UniLab after EPROM is programmed. - Status message after issuing a prom programming command in **STANDALONE** mode.

target adr (not EMENABLED) - This is usually not an error message. It is a notice that you are addressing a memory location that you have not enabled. Either you have made a mistake, or you are purposely addressing RAM on your target board rather than emulation ROM. A read or write to target memory is only possible after you have established debug control. This message is always printed out as a reminder that the debugger is performing the memory read or write, and that this is not a simple transfer of data between the host and UniLab.

The command is: <command> - This is a menu display to show the user what the associated command would be if entered as a command rather than as a menu choice.

This Eprom does not seem to be erased!!! - An error message when you try to program an EPROM that is not blank.

TO address is smaller than FROM address. - A command was given which has mismatched parameters-- the second number is too small.

Too many files - Not enough files have been allocated, so you cannot open another. This can be changed by changing the **CONFIG.SYS** file in your root directory to contain the line **FILES=16**. See the **Installation** chapter.

unloadable - See All About FORTH.

Unsupported record type. Types 0, 2, & 3 only are supported. - The Intel hex format file that you are trying to load with **HEXLOAD** is of the wrong record type. Bytes 7 and 8 of each line of the file tell what record type the line uses.

Wrong UNILAB.SCR file - The file named UNILAB.SCR on the disk is not the correct version for the ULxxx.COM file. Make sure that you copied everything from the master diskette.

2. PARAMETER ENTRY ERRORS

These messages result when you call a command without giving it the proper number of parameters. You can enter any command name without the parameters, and get a prompt that tells you what the command requires.

Requires a Goto-Address, and the next Breakpoint-Address

Requires the From-Address, and the To-Address

Requires a Number-of-Lines

Requires the From-Address, the To-Address, and a Value

Requires Source-From, Source-To, and Destination addresses(1)

Requires the Start-Address and the #-of-lines

Requires an Address

Requires a Word (16 bit) Value

Requires a Cycle#

Requires a Byte (8 bit) Value

Requires an Address or Range (Address1 TO Address2)

Requires the Cycle#

Requires a Block-Number

Requires a Value and a Destination-Address

Requires an Address and a Block-Number

Requires a Nibble (4 bit) Value

Requires 'host-address target-address #bytes'

3. MENU MESSAGES

These messages are generated by entries in the menu, to let you know what values that menu item needs. When you get one of these messages, enter the value requested.

Enter the Starting target address:
Enter the Ending target address:

Enter the Target address:
Enter the First value:
Enter the Last value:
Enter the Value:

Enter the Bit# (0-7):
Enter the desired hex output code (0-FF)
Enter the Trigger address:
Enter the Source address:
Enter the Destination address:
Enter the number of lines to disassemble (default=5):
Enter the starting address of the first block to compare:
Enter the starting address of the second block:

Enter the breakpoint address in emulation memory:
Enter address to continue execution:
Enter next breakpoint address:
Hit key for next debug function (from above):

Appendix J: .BIN Files and .TRC Files

Overview

Your distribution diskette includes one or more **.BIN** files, the binary image of the simple target program for your microprocessor. That file is described in Chapter Three: **The Guided Demo**.

The chart in this appendix tells you which file to load into memory, and where to load it.

With some packages, we distribute a **.TRC** file to test your cable connection to your processor, as described in section 4 of the **Installation** instructions of Chapter Two. This appendix includes the name of the **.TRC** file, for those processor packages that include demonstration traces

Patches and 3.20 Software Update:

All software packages which support multiple processors now provide you with a "patch menu." The is menu is presented to you when you call up the UniLab software, and is also available through the command **PATCH**. Thus, the "patch word" is always **PATCH**.

.BIN files

You can load the sample program into memory with the command **LTARG**, which also takes care of enabling memory and any other needed details.

However, loading the **.BIN** file from disk gives you familiarity with the procedure you will have to follow when you load your own program from disk.

Before using BINLOAD

With most processors, the only preparation you need is enabling the 2K segment into which you will load the binary file. For example, to load the binary file **test65** into emulation memory for the 6502:

```
FF00 EMENABLE  
FF00 FFFF BINLOAD test65
```

With some processors, you might have to enable more than 2K

of memory, or set the value of some variables. These exceptions are noted in the following chart.

The <to address> argument of BINLOAD

BINLOAD needs two arguments in addition to the file name: the <from address> and the <to address>. These addresses tell **BINLOAD** where to start loading the file, and when to stop.

Of course, if **BINLOAD** reaches the end of the binary file before it reaches the <to address>, then it will stop loading.

You should always give as the <to address> the highest address in emulation memory. Though the sample programs are small, the reset vector is often at the top of memory, pointing at an address closer to the bottom of memory.

.TRC files

In general, you use the trace file to verify that you have the proper connections between your UniLab and target board.

Comparing your trace to the one on diskette

Load the sample program, either from the **.BIN** file or with **LTARG**, and use **STARTUP** to generate a trace. Then use:

AA TCOMP <trace file name>

to compare your entire trace to the trace on diskette.

You can also use:

TSHOW <trace file name>

to look at the trace that is stored on diskette.

Comparing partial traces

Sometimes the difference between the standard trace file and your trace will be trivial. For example, some simple target programs contain instructions that read RAM locations which have not been initialized-- and so the value stored in that location will vary.

The UniLab software stops checking the traces after it finds the first difference. If you want to compare your trace starting after the trivial difference, then you will use the **TCOMP** command with a different parameter:

<number of cycles to compare> **TCOMP** <trace file name>

Note that the number of cycles is a count starting from the end of the trace buffer.

You can also use **TMASK** to specify that only certain columns of the trace should be compared. Consult the TMASK entry in the **Command Reference** chapter.

N/A

Trace files are not available (n/a) for all processors.

Chart of .BIN and .TRC files

	Processor	.BIN file	Load in starting at address	.TRC file
DDB-48	8048 family members in expanded mode	TEST48	00	DEMO48
DDB-51	8051 family members in expanded mode NOTE: You must also enable F800 to FFFF for the overlay area.	TEST51	00	DEMO51
DDB-51P	8051P, 80C51P NOTE: You must also enable 800 to FFF for the overlay area.	TEST51P	00	DEMO51P
DDB-611	68HC11	TEST611	FF00	DEMO611
DDB-63	6303R 6303X (obsolete) 63P01	TEST63 TEST63 TEST63	FF00 FF00 FF00	DEMO63R DEMO63X DEMO63P
DDB-65	6502 or 65C02	TEST65	FF00	DEMO65
DDB-65P	R65/11EB R65/41 R6511Q	TEST65P TEST65P TEST65P	FF00 FF00 FF00	DEMO65EB DEMO6541 DEMOR65Q
DDB-68	6800 or 6808	TEST68	F800	DEMO68
DDB-681	6801 6803 piggyback	TEST681 TEST681 TEST681	F800 F800 F800	DEMO681 DEMO681 n/a
DDB-682	6802	TEST682	F800	DEMO682
DDB-685	6805E2 HD6305 piggyback chips 6805E3	TEST685 TEST685 TEST685P TES685E3	1F00 1F00 F00 FF00	n/a n/a DEMO685P n/a

	<u>Processor</u>	<u>.BIN file</u>	<u>Load in starting at address</u>	<u>.TRC file</u>
DDB-688	68008	TEST688	00	DEMO688
DDB-689	6809E	TEST689	FF00	DEMO689
DDB-68K	68000	TEST68K	00	DEMO68K
DDB-85	8085 8080	TEST85 TEST85	00 00	DEMO85 n/a
DDB-86	8086 min 8086+ 80186 80286	TEST86MI TEST86MA TEST186 TEST286	F800 F800 F800 F800	DEMO86MI n/a n/a n/a
NOTE:	With all members of the 8086 family, you should use SEG' before loading into emulation memory with BINLOAD . After loading, you can turn back on the interpretation of addresses as offsets from segments, with SEG .			
DDB-88	8088 min 8088+ 80188	TEST88MI TEST88MA TEST188	F800 F800 F800	DEMO88MI n/a DEMO188
NOTE:	With all members of the 8088 family, you should use SEG' before loading into emulation memory with BINLOAD . After loading, you can again turn on interpretation of addresses as offsets from segments, with SEG .			
DDB-96	8094, 8095, 8096, 8097	TEST96	2080	DEMO96
NOTE:	You must enable the entire area 0 to 2FFF, for the overlay area.			
DDB-HD64	HD64180	TESTHD64	00	DEMOHD64
DDB-SC8	NSC-800	TESTSC8	00	DEMOSC8
DDB-S8	ROMless members of the super 8 family	TESTS8	20	DEMOS8

	Processor	.BIN file	Load in starting at address	.TRC file
DDB-Z8	Z8	TESTZ8	0C	n/a
	piggyback	TESTZ8P	0C	DEMOZ8P
	NOTE: You must enable 00 through FFF.			
	NOTE #2: Assumes you have pulldown resistors on the upper address lines. If you have pullup resistors, then ALSO FFOC EMENABLE and then 0C FF FFOC MMOVE.			
DDB-Z80	Z80	TESTZ80	00	DEMOZ80
	NSC-800	TESTZ80	00	DEMONSC8
	HD64180	TESTZ80	00	DEMOHD64
DDB-Z8K	Z8001, Z8003			
	These two processors need two values initialized: 0 =ROM.SEGMENT 0 =RAM.SEGMENT			
	Z8002, Z8004	TESTZ81	00	n/a
		TESTZ82	00	n/a
DIS-18	1802 family	TEST18	00	DEMO18

FULL INDEX

This index covers all chapters in both volumes. The index at the end of Volume I covers only that first volume.

Neither index covers the appendices.

.BIN file	3-6
.TRC file	2-42
:	7-16
;	7-17
?FREE	6-24, 6-93, 7-23
=BC	7-18
=EMSEG	4-7, 6-39, 7-19
use	6-13
=HISTORY	6-93, 7-21
=SYMBOLS	6-24, 7-23
>FILE	7-176
>FILE'	7-176
\ORION	2-15
16-bit Systems	
ROM cable	2-5
16BIT	7-13
1802 family	8-5
1AFTER	4-30, 7-11
2AFTER	7-14
32K boundaries	
and MMOVE	6-59
32K UniLab	
limitations	6-37
3AFTER	7-14
48 CHANNEL BUS STATE ANALYZER	2-26
6301	8-7
6303R	8-7
6303X	8-7

6305	8-25
65/11EB	8-14
65/41	8-14
6500 family	8-10
6502	8-10
6511Q	8-14
6800	8-18
68000	8-32
68008	8-36
6801	8-21
6802	8-24
6803	8-21
6805E2	8-25
6805E3	8-25
6808	8-18
6809E	8-30
68HC11	8-38
8/16 BIT IN-CIRCUIT EMULATOR .	2-26
80186	8-53
80188	8-53
80286	8-53
8048 family	8-40
8051	
reset	2-34
8051 family	8-45
8080	8-50
8085	8-50
8086	8-53
8086/88 family	
NMI	2-38
8088	8-53
8096 family	8-61
8800	8-65
8822	8-65
8BIT	7-15
A11	8-4
ADR	4-24, 6-68, 6-75, 7-24
ADR?	4-10, 7-25
AFTER	4-11, 7-26
AHIST	6-153, 7-28
ALIGN	8-56
ALLCY	8-10
ALSO	4-29, 7-29
and EMENABLE	6-40
ALT-FKEY	4-34, 7-30
ALT-FKEY?	7-30
Analyzer	2-8
cable	2-25
menu	4-10

AS	3-12, 7-31, 9-15
ASC	7-32
ASCII	
codes	7-32
ASEG	7-33
ASM	2-43, 6-61, 7-34
ASM-FILE	6-61, 7-36
Assembler	6-61
AT	
serial port	2-7, 2-12
AUTOEXEC.BAT	2-15, 2-16
AUX1	7-37
AUX2	7-37, 9-3
B.	7-38
B#	7-38
Batch files	4-3
writing	4-20
Baud rate	
19,200	2-7
BINLOAD	3-6, 4-8, 7-39
BINSAVE	4-8, 6-46, 7-40
BPEX	7-41
Breakpoint	
address zero	9-14
setting	3-13
Breakpoint display	
example	6-109
Bus activity	6-2
Bus state analyzer	1-1
BYE	2-9, 2-45, 7-42
Cable connection	
verify	2-42
Cable diagrams	
on-line	7-42
CATALOG	7-42
Chips	
in UniLab	9-16
CHKSUM	7-43
Circuit	
open collector	2-27, 2-33, 2-36
CLEAR	2-47, 7-44
CLEAR'	7-44
Clock inputs	2-8
CLRMBP	7-45
CLRSYM	7-46
COLOR	2-47, 7-47
COM1	2-12, 7-48, 9-3, 9-7
COM2	7-49

Command file	
.COM	2-20
ULxx.COM	2-3
Command tail	4-3, 4-20
TOFILE	6-94
Compare	
traces	7-173
CONFIG.SYS	2-9, 2-15, 2-17, 2-19
Connect	
UniLab to host	2-11
Connection	
diagram	2-26
DIP CLIP	2-31
NMI-	2-36
RES-	2-33
ROM cable	2-29
verify	2-5, 2-42
Connections	
UniLab to host	2-9, 2-11
UniLab to target	2-9, 2-25
CONT	4-24, 4-27, 6-75, 7-50
CONT column	9-13
CONTROL	7-52
Controls	2-3
CTL-FKEY	7-53
CTL-FKEY?	7-53
CTRL-BREAK	2-13
Cursor keys	4-36, 6-144
chart	4-52
screen history	4-38
Cy#	6-8
Cycle numbers	6-71
"f"	4-30
Cycles	
show all	8-10
CYCLES?	4-10, 7-54
D#	7-55
DASM	7-56, 9-13
DASM'	7-57
DATA	4-24, 6-75, 7-58
Data bus	
16-bit	2-5
DB-25	2-12
DCE	9-3, 9-4
DCYCLES	7-60

DDB		
	48	8-40
	51	8-45
	51P	8-45
	611	8-38
	63	8-7
	65	8-10
	65P	8-14
	68	8-18
	681	8-21
	682	8-24
	685	8-25
	688	8-36
	689	8-30
	68K	8-32
	85	8-50
	86	8-53
	88	8-53
	96	8-61
	S8	8-65
	Z8	8-68
	Z80	8-72
	Z8K	8-76
Debug control	3-13, 8-3
and RAM	6-49
establish	6-104
example	6-108
menu	4-13
Debug Control not established		
and RAM	6-50
Debugger		
and emulation ROM	8-2
and stack	8-2
disable	6-38, 6-124
exit from	6-122
menu	4-13
stack	6-11
DEF	7-61
Demo program		
.BIN	3-6
trace (.TRC)	2-42
Demo session	8-4
Development system	1-1
DIP clip	2-31
Connection	2-31
DIS-18	8-5
Disassemble	3-8, 6-53
DISASSEMBLER	6-29
in sync?	8-18
out of sync	6-53
DM	3-8, 7-62
watch out	6-53

DMBP	7-62
DN	4-44, 7-63
DOHIST	7-28, 7-172
DOS	7-64
DOS command	
EXE2BIN	6-42, 7-39
TYPE	9-5
VER	2-2
Dos command files	
.COM	2-3
Down Arrow	4-36, 4-37, 4-52
DTE	9-4
Dumb terminal	
PC	7-48
EMCLR	2-28, 4-7, 7-65
EMENABLE	3-5, 4-7, 6-40, 7-66
and ALSO	6-40
Emulation memory	2-7
Emulation ROM	1-1, 6-34
128K	6-37
access	6-49
clear	7-65
compare	7-100
crash-free access	6-49
disassemble	7-62, 7-63
enable	6-38, 7-66
explanation	6-36
overlap	6-37
read	7-98, 7-101, 7-112
status	7-68
warning	6-49
write	7-96, 7-97, 7-105, 7-110, 7-111
Emulation settings	
save	4-23
Emulator	2-8
cable	2-25
Enable memory	3-2, 3-4
menu	4-7
End key	4-42, 4-46, 4-52
EPROM programmer	1-4, 6-125
Error messages	2-22
NG!	9-14
NO ANALYZER CLOCK	9-10
RS-232 error #xx	2-22, 9-6
Establish	
debug control	6-104
ESTAT	4-7, 6-40, 7-68
EVENTS?	7-69
EXE2BIN	6-42, 7-39

Exit 2-9, 2-45, 7-42
Exit from
 debug control 6-122

f

 cycle numbers 4-30, 6-8
F8 5-3
FETCH 4-11, 4-28, 7-70
Filter 7-71
 cycle numbers 4-30
Filters
 trigger specs 4-30
FKEY 7-72
FKEY? 7-73
Flicker 2-47
Footer
 trace display 7-82
FORTH 7-16, 7-41
 file 6-63
Function keys 4-34, 5-5, 6-142
 set 7-30, 7-53, 7-73, 7-150

G 7-74
GB 4-13, 7-75
GLOSSARY 2-15, 2-17
Graphical Performance
 Measurement 6-152
Guided demo 3-2
GW 7-76

H>D 7-77
HADR 7-78
HD64180 8-72
HDAT 7-79
HDATA 4-24, 6-75, 7-80
HDG 7-82
HDG' 7-82

Header

 trace display 7-82
HELP 5-2, 5-3, 7-83
HEXLOAD 4-8, 7-84
HEXRCV 7-86
High level language
 support 7-154
Histogram
 address 7-28
 DOHIST 7-28, 7-172
 time 7-172
Histograms 6-152

History	6-93
space allotted	7-21
History mechanism	6-19
HLL	
high level language	7-154
HLOAD	6-153, 7-28, 7-172
Home	4-40, 4-52
Host	2-7
HSAVE	6-153, 7-28, 7-172
INFINITE	7-87
INIT	2-13, 7-88, 9-3
Initialize	7-37
stack pointer	9-12
Initializing UniLab	2-13, 2-22, 9-3
Input	2-8
INSTALL	2-16
INSTALL.BAT	2-14
Installation	2-5, 2-9
INT	2-37, 7-89
INT'	7-90
INTEL	
HEX format	7-84
reset	2-33
Internal state	1-4
IRQ	2-37
IS	6-23, 7-91
LADR	7-92
Language	
FORTH	7-16, 7-41
high level	7-154
Leave	2-45, 7-42
debug control	6-122
Limitations	
32K UniLab	6-37
Line history	
size	6-93
Load	
binary	7-39
hexfile	7-84
histogram	7-172
program	6-42
symbols	6-22, 7-166
trace	7-181
Load program	
into memory	3-6
menu	4-8
LOG	7-93
LOG'	7-93

LP	7-94
LTARG	3-5, 4-37, 6-38, 6-45, 7-95, 9-1
flow chart	6-68
from menu	2-41
M	7-96
M!	4-9, 7-97
M?	7-98
Macro	7-16, 7-41
example	4-22, 6-134
Main menu	2-44, 3-4, 4-6
MASK	7-99
MCOMP	4-9, 7-100
MCS-96	8-61
MDUMP	3-7, 4-9, 7-101
MEMO	7-102
Memory (See also Emulation, ROM, and RAM)	
emulation	2-7
Memory access	
menu	4-9
MENU	2-22, 4-4, 7-104
map	4-5
Menu system	
guided demo	3-4
MESSAGE	7-104
MFILL	4-9, 7-105
MISC	4-24, 6-75, 7-106
MISC'	7-108
MLOADN	7-109
MM	7-110
MM!	2-43, 4-9, 7-111, 9-12
MM?	7-112
MMOVE	4-9, 7-113
limitations	6-59
Mode	7-114
CONT COLUMN	5-8, 6-31, 6-149
DISASSEMBLER	5-7, 6-29, 6-148
FIXED HEADER	5-8, 6-33, 6-149
LOG TO FILE	5-9, 6-150
LOG TO PRINT	5-9, 6-150
MISC # BASE	5-8, 6-32, 6-149
MISC COLUMN	5-8, 6-31, 6-149
NMI VECTOR	5-9, 6-150
PAGINATE	5-8, 6-33, 6-149
PRINTER	5-9, 6-150
RESET	5-7, 6-148
SWI VECTOR	5-9, 6-150
SYMBOLS	5-7, 6-30, 6-148

Mode panels . . . 5-3, 6-28, 6-146
 help 5-7
 MS 7-115
 MS-DOS 2-2

N 7-116
 single step 3-15
 NDATA 7-117
 NG! 9-14
 NMI 7-118, 8-3
 8086/88 family 2-38
 disable/enable 7-119
 NMI-
 circuit 2-26
 connection 2-36
 NMIVC 2-37, 7-119
 NMIVC' 7-119
 NO ANALYZER CLOCK 9-10
 NORMB 4-24, 7-120
 NORMM 4-24, 7-121
 NORMT 4-24, 7-122
 NORMx 4-25, 6-72
 NOT 4-11, 7-123
 scope 6-76
 not enabled 6-50
 Not recognized 4-26
 NOW? 4-10, 7-124
 NSC-800 8-72
 Numeric key pad 4-52

Object file
 binary format 7-39
 INTEL hex format 7-84
 On-Line Help 5-1, 7-83, 7-104, 7-182
 ONLY 4-11, 4-30, 7-125
 Open collector 2-27, 2-33, 2-36
 ORG 7-127
 ORION 2-15, 2-17
 Overlay area 8-2
 disable 7-142
 enable 7-142

PAGE0 7-128
 PAGE1 7-128
 PAGINATE 7-129
 PAGINATE' 7-129
 Parallel interface 2-8
 Patch
 stack pointer 9-12

Patch word	2-23
PC	
dumb terminal	7-48
PC compatible	2-2
PCYCLES	7-130
Personality modules	6-127
PEVENTS	7-131
PgDn	4-36, 4-37, 4-52
PgUp	4-36, 4-38, 4-52
flicker	2-47
history	6-19
PINOUT	2-9, 2-26, 2-32, 4-15, 7-132, 8-3, 9-13
catalog	7-42
PRINT	7-133
PRINT'	7-133
PROM	4-16
commands	6-35
PROM programmer	6-125
PROM programming	
menu	4-17, 6-126
PROM reading	
menu	4-16
PROMMSG	7-133
Q1	7-134
Q2	7-134
Q3	7-134
Qualifiers	7-135
trigger specs	4-31
Quit	7-42
RAM	
access	6-50
access to	6-49
and Debug control	6-49
compare	7-100
disassemble	7-62, 7-63
read	7-98, 7-101, 7-112
warning	6-50
write	7-96, 7-97, 7-105, 7-110, 7-111
RB	3-13, 3-14, 7-136
and RESET	6-112
Read	7-138
Emulation ROM	6-49
RAM	6-50
Reboot	2-9, 2-19
RES	7-139

RES-

- circuit 2-26
- connection 2-33
- Reserved area 8-2
- location 6-124
- Reserved resources 8-2
- Reset . . . 2-8, 2-33, 4-12, 4-14,
6-74, 7-140
- 8051 2-34
- and RB 6-112
- INTEL 2-33
- use 6-74
- Z80 2-35
- RESET' 6-74, 7-140
- resetting
- status message 6-74
- RI 2-36
- RMBP 7-141
- ROM
- emulation 2-7
- enable 4-23
- reading 6-44
- ROM cable
- 16-bit 2-5
- connection 2-29
- ROM chip
- analyze 2-28
- RS-232 2-7
- RS-232 error #xx 2-22, 9-6
- RSP 7-142
- RSP' 2-28, 7-142
- RZ 7-143

- S 7-144
- S+ 7-145
- SAMP 3-11, 4-10, 7-146
- Sample program 2-41
- Sample session 3-2, 8-4
- Save
- binary 7-40
- histogram 7-172
- history 6-93
- symbols 6-98, 7-166
- system . . . 6-41, 6-99, 7-147
- to printer . 6-95, 7-93, 7-133
- trace 6-96, 7-180
- transcript 6-94, 7-176
- SAVE-SYS 2-23, 4-23, 7-147
- SC 7-148
- Scope of
- NOT and TO 6-76

Screen Flicker	
fix	2-47, 7-44
Screen history	4-38, 6-93
size	6-93
Serial interface	2-7
Serial port	2-5, 2-7, 2-12
9 pin	2-7, 2-12
AT	2-7, 2-12
choose	7-37
SET	4-14, 7-149
Set GLOSSARY	2-15, 2-17
Set ORION	2-15, 2-17
SET-COLOR	2-47, 7-149
SHIFT-FKEY	7-150
SHIFT-FKEY?	7-150
SHOWC	7-151
SHOWC'	7-151
SHOWM	7-152
SHOWM'	7-152
SI	2-36
Single step	3-15
Size	
line history	6-93
symbol table	6-24
SMBP	7-153
Soft-keys	5-5
Software	
installation	2-9, 2-14
SOURCE	7-154
Source file	
view	7-171
SOURCE'	7-154
SPLIT	7-155
SR	7-156
SST	7-157
SSTEP	2-36, 3-15, 4-13, 7-158
Stack	
and debugger	8-2
debugger	6-11
Stack pointer	2-43, 9-12
patch	9-12
STANDALONE	7-159
STARTUP	3-10, 4-10, 7-160
STIMULUS	4-14, 7-161
Stimulus generator	1-4, 6-137
menu	4-14
SUPER 8	8-65
SWI VECTOR	2-28
SYMB	7-162
SYMB'	7-162

Symbol files	
fixed format	6-27
variable format	6-27
Symbols	6-30
breakpoints	6-26
clear	7-46
define	6-23, 7-91
enable/disable	7-162
example	6-25
files	6-22, 7-163, 7-166
in trace	6-21
space allotted	7-23
trace display	6-25
SYMFILE	6-22, 7-163
SYMFILE+	6-22, 7-164
SYMFIX	7-165
SYMLOAD	7-166
SYMSAVE	7-166
SYMTYPE	6-22, 7-167
T	7-168
Tail	
command	4-20
Target	2-7
Target board	
.	2-7
Target memory-- see Emulation, ROM	
and RAM.	
TCOMP	2-5, 2-42, 6-96, 7-169
mask columns	7-173
TD	7-170
Terminal	
emulation	7-48
TEXTFILE	4-49, 7-171
THIST	6-153, 7-172
TMASK	2-42, 7-173
TN	7-174
TNT	7-174
TO	4-11, 7-175
scope	6-76
TOP/BOTTOM	7-177
Trace buffer	6-2
dump	7-170
Trace compare	
TMASK and TCOMP	2-42

Trace display	6-5
16 bit data	6-16
ADR	6-8, 6-13
compare	7-169
CONT	6-8, 6-12, 6-13
cy#	6-8, 6-12
DATA	6-9, 6-14
disassemble	6-9
f	6-8
file	7-180, 7-181
HDATA	6-9
header	7-82
MISC	6-9
modes	6-28
move through	6-17, 7-168, 7-174
save	7-180
Transcript	
save	7-176
TRIG	7-178
Trigger	1-3
menu	4-11
Trigger event	6-3
Trigger specs	4-24
examples	4-27, 6-78
filtered	4-30, 6-82, 7-11, 7-125
qualifiers	6-85, 7-26, 7-135
refinement	6-89
simple	6-70
status	7-182
TROUBLESHOOTING	9-1
debugger	6-107
TS	7-179
TSAVE	6-96, 7-180
TSHOW	7-181
TSTAT	7-182
TYPE	9-5

UniLab	
internal ICs	9-16
Up Arrow	4-36, 4-38, 4-52

VER	2-2
Verify	
cable connection	2-42
Version	2-2

Watch program	
STARTUP	3-10
Windows . 4-41, 6-151, 7-155, 7-177	
change size	4-46, 7-183
size	7-61
WORDS	5-3, 7-182
Write	
Emulation ROM	6-49
RAM	6-50
WSIZE	7-183
Z8	8-68
pull-down resistors	8-70
Z80	8-72
reset	2-35
Z8000 family	8-76