



8001/8002A
 μ PROCESSOR LAB
REAL-TIME
PROTOTYPE ANALYZER
8001 Option 46
8002A Option 46
USER'S GUIDE

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077
070-2785-00

Serial Number _____

First Printing MAR 1979

WARRANTY

The 8001/8002A μ Processor Lab System and options, excluding customer supplied equipment, is warranted against defective materials and workmanship, under normal use, for a period of ninety (90) days from date of shipment. CRTs found to be defective after the ninety (90) day period and up to twelve (12) months from date of shipment will be exchanged at no charge for the material. Tektronix will repair or replace, at its option, those System components which Tektronix determines to be defective within the warranty period.

In addition, in those areas where Tektronix has service centers available for this system, on-site warranty repair is provided at no charge during the first ninety (90) days from date of shipment.

Tektronix shall be under no obligation to furnish warranty service if:

- a. Attempts to install, repair, or service the equipment are made by personnel other than Tektronix service representatives.
- b. Modifications are made to the hardware or software by personnel other than Tektronix service representatives.
- c. Damage results from connecting the 8001/8002A μ Processor Lab System to incompatible equipment.

There is no implied warranty for fitness of purpose. Tektronix is not liable for consequential damages.

Copyright © 1979 by Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, TELEQUIPMENT, and  are registered trademarks of Tektronix, Inc.

Printed in U.S.A. Specification and price change privileges are reserved.

TABLE OF CONTENTS

	Page		Page
SECTION 1		GENERAL INFORMATION	
		Introduction	1-1
		About This Manual	1-2
		Manual Application	1-3
		Description of RTPA Option	1-3
		Overview of RTPA	1-4
		RTPA Triggering Modes	1-24
		Emulation Modes	1-26
		Using the RTPA With A Logic Analyzer	1-28
SECTION 2		RTPA COMMANDS	
		Introduction	2-1
		Command Conventions	2-1
		Description of RTPA Commands	2-4
		EVT Command	2-6
		BIF Command	2-22
		RTT Command	2-34
		DRT Command	2-36
		CNT Command	2-38
SECTION 3		HOW TO USE THE RTPA COMMANDS	
		Introduction	3-1
		Assumptions	3-1
		Limitations of Some Emulator Processors	3-1
		Equipment Required	3-2
		Emulation Mode	3-2
		Procedures	3-2
SECTION 4		RTPA COMMAND INTERACTIONS	
		Introduction	4-1
		Interaction with 8001 System Commands	4-1
		Interaction with 8002A System Commands	4-1
		Interaction with 8001/8002A Debug Commands	4-1
		Interactions Between RTPA Commands	4-2
SECTION 5		RTPA APPLICATIONS	
		Introduction	5-1
		Equipment Required	5-1
		Emulation Mode 0	5-2
		Emulation Modes 1 and 2	5-16
		Using the Test Probe Clips	5-27
		Using the Logic Analyzer	5-32
		Summary	5-51
SECTION 6		GLOSSARY	
APPENDIX A		EMULATOR CHARACTERISTICS	
		Introduction	A-1
		RTPA Timing Considerations	A-1
		Addition of Wait States	A-1
		Program Calculations	A-5
		RTPA Measurements	A-6
		Individual Emulator Characteristics	A-10
		3870 Emulator Processor	A-11
		6800 Emulator Processor	A-13
		8080A Emulator Processor	A-15
		8085A Emulator Processor	A-17
		9900 Emulator Processor	A-19
		Z80 Emulator Processor	A-21
APPENDIX B		COMMAND DICTIONARY	
		Introduction	B-1
APPENDIX C		INSTALLATION	
		Installing the Real-Time Prototype Analyzer	C-1
		Grounding	C-5
		Jumpers/Switches	C-5
		Packaging for Shipment	C-8
APPENDIX D		SAMPLE PROGRAM	

Section 1 GENERAL INFORMATION

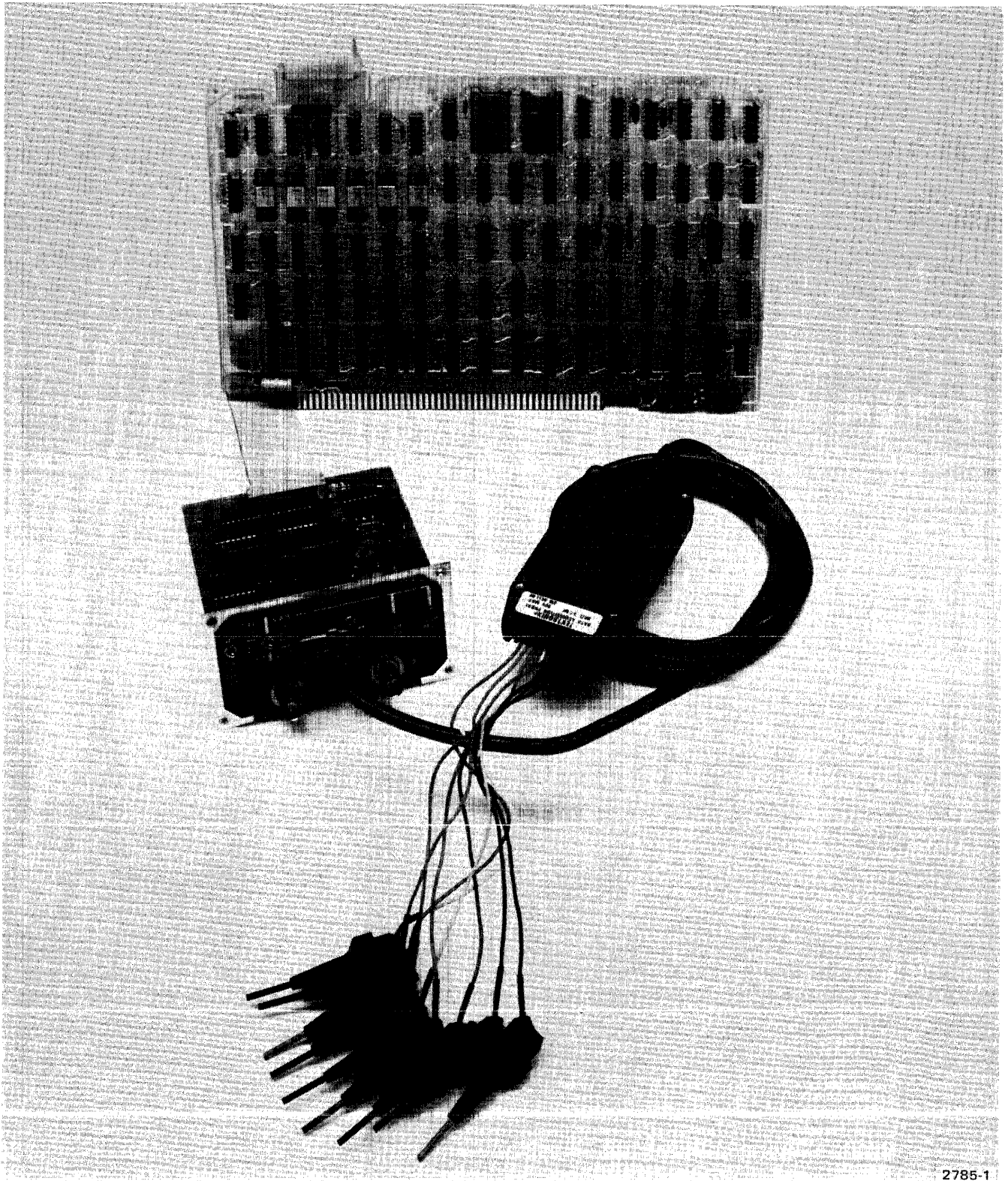
	Page
Introduction to the RTPA	1-1
About This Manual	1-2
Manual Application	1-3
Description of RTPA Option	1-3
Overview of RTPA	1-4
General Description	1-4
RTPA Functions	1-4
How the RTPA Works	1-7
Input Data	1-7
Input Data Storage (RTT Buffer)	1-10
Event Comparison (EVT1 and EVT2)	1-13
Event Pass and Delay Counters	1-19
Event Triggers	1-21
System Interrupts	1-21
Data Acquisition Interface	1-22
RTPA Triggering Modes	1-24
Pre-, Center, or Post-Triggering	1-24
Pre-Triggering	1-25
Center Triggering	1-25
Post-Triggering	1-26
RTT Storage	1-26
Emulation Modes	1-26
Emulation Mode 0	1-27
Emulation Mode 1	1-27
Emulation Mode 2	1-28
Using the RTPA With A Logic Analyzer	1-28

ILLUSTRATIONS

		Page
Fig. No.		
1-1	Real-Time Prototype Analyzer	1-ii
1-2	Relationship of data window to the designated event	1-1
1-3	Functional diagram of RTPA	1-5
1-4	RTPA block diagram	1-9
1-5	Trace control and RTT buffer timing diagram	1-11
1-6	Event comparison block diagram	1-12
1-7	Flowchart showing timing relationships with SLV OPREQ signal	1-15
1-8	Comparison of EVT1 address and bus operation options	1-16
1-9	Result register and EVT flip-flops simplified block diagram	1-18
1-10	Pass and delay counter simplified block diagram	1-20
1-11	Data Acquisition Interface simplified block diagram	1-22
1-12	Data Acquisition Interface panel controls and connectors	1-24
1-13	RTPA Pre, Center, and Post-Triggering modes	1-25
1-14	Typical interconnections showing the RTPA triggering a logic analyzer	1-29

TABLES

		Page
Table No.		
1-1	Input Data Bits to the RTPA	1-8
1-2	Result Registers	1-16



2785-1

Fig. 1-1. Real-Time Prototype Analyzer

Section 1

GENERAL INFORMATION

INTRODUCTION TO THE RTPA

The Real-Time Prototype Analyzer (RTPA) is an optional feature of the 8001/8002A μ Processor Lab. The RTPA may be factory-installed, if ordered originally with the 8001/8002A system, or obtained later as a field modification installation. See Appendix C of this manual for installation instructions.

The RTPA, when used in conjunction with the 8001/8002A system, allows you to capture and store a window of data from a microprocessor-based system's program, while the program is being executed. The window is associated with a designated event that may be moved anywhere in the program. This window represents 128 bus transactions from the system's program and may be positioned before, during, or after the designated event. Fig. 1-2 is a graphic representation of this data window showing its relationship to the designated event.

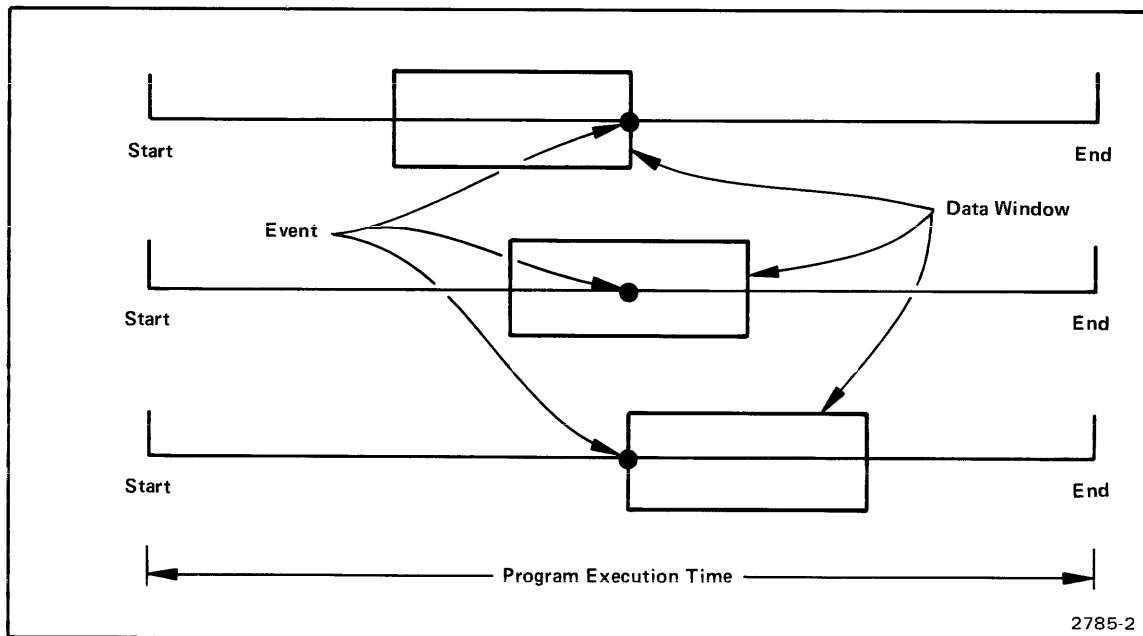


Fig. 1-2. Relationship of data window to the designated event.

If the data window is positioned to a portion of the program that is not functioning properly, the data in the window can be further analyzed and evaluated to find the error in the program.

The RTPA enhances the debugging features and capabilities of the 8001/8002A system. The program can be debugged within the environment of the 8001/8002A system using the program memory, input/output ports, and clock. Any part or all of the program can be transferred to the prototype (microprocessor-based) system for debugging and evaluating the hardware/software sequence discrepancies and timing errors; while executing the program.

ABOUT THIS MANUAL

This User's Guide can be used by both software- and hardware-oriented designers and engineers.

Section 1 contains an overview of the RTPA. It describes how the RTPA works, using a functional block diagram, and tells how the RTPA hardware functions under software control (the RTPA commands). Thus, providing the user with debugging and evaluation capabilities during software/hardware integration of the prototype (microprocessor-based) system.

Section 2 contains an explanation of the RTPA command names, describes each command and its associated parameters, explains when the commands should be entered, and what happens when the commands are invoked. This section contains a few examples where necessary for clarity.

Section 3 describes how to enter the RTPA commands and shows the displays that result. The procedures in this section include examples that define the capabilities and limitations for each RTPA command. The procedures and examples are constructed so that you can repeat the examples on your own 8001/8002A system.

Section 4 defines how the RTPA commands interact with the system commands (8001/8002A system) and with other RTPA commands.

Section 5 describes how the RTPA commands can be used in emulation Modes 0, 1, and 2 to debug and evaluate your software/hardware design. This section contains practical applications. Examples describe how to enter and display the RTPA commands during the design integration phases.

Section 6, the glossary, defines RTPA-associated terms used in this manual.

Appendix A describes the characteristics of the various emulators as they relate to the RTPA operation. Additional emulator specifics are contained in the 8001 or 8002A μ Processor Lab System User's Manuals.

Appendix B is an abbreviated command dictionary that alphabetically lists and defines all 8001/8002A system commands associated with RTPA operation.

Appendix C defines the installation procedures associated with RTPA operation.

Appendix D contains a reproduction of the sample program list file, Fig. 3-3. It is included for use as a ready reference when performing the examples in Sections 3 and 5.

MANUAL APPLICATION

This manual is based on the latest software versions for the 8001/8002A system. The following is a list of the latest software versions available at this writing. Earlier software versions in some instances may provide different displays than contained herein; in such cases, the examples used in this manual should be modified accordingly.

μ Processor Lab	Software Version
8001	TEKOPS Version 2.0
8002A	TEKOPS Version 3.0

DESCRIPTION OF THE RTPA OPTION

The RTPA (Fig. 1-1) consists of the following three sub-assemblies:

- Real-Time Trace module
- Data Acquisition Interface unit
- Data Acquisition Probe

The Real-Time Trace module is a circuit board that plugs into the 8001/8002A system motherboard. (See Appendix C for the installation procedures.) The Real-Time Trace module contains all the circuitry associated with the functions and features of the RTPA commands.

The Data Acquisition Interface unit consists of a circuit board and panel that is attached to the rear panel of the 8001/8002A system. (See Appendix C for the installation procedures.) The Data Acquisition Interface unit contains the buffer/storage registers and receivers that interface the nine channels of the P6451 Data Acquisition Probe to the Real-Time Trace module. Eight of the channels are used to transmit 8 bits of data information from the prototype circuitry to the RTPA input. The ninth channel is a clock channel from the prototype to the buffer/storage registers in the Data Acquisition Interface unit. The Data Acquisition Interface unit also contains drivers that interface the event triggers to the BNC connectors on the Data Acquisition Interface panel. (See Appendix C for the location of the BNC connectors.) These event triggers, generated by the RTPA, are used to trigger external equipment, such as an oscilloscope or logic analyzer.

The Data Acquisition Probe (P6451 Data Acquisition Probe) is a nine-channel active probe. Eight of the channels are used for data acquisition. The ninth channel is used for a clock signal. The ten leads that plug into the probe head, have test clips attached for connecting to an 8-bit data source as follows:

- 8 leads to the 8-bit data source.
- 1 lead to the clock source.
- 1 lead to ground (logic ground).

OVERVIEW OF THE RTPA

General Description

The RTPA, when installed in the 8001/8002A, extends the capabilities of the system to include various general features as follows:

1. Utilizes up to 43 bits of input data, consisting of:
 - 6-bit address bus
 - 8 or 16-bit data bus
 - 8-bit data from an external probe
 - 3-bit bus operations
2. Provides a trace storage and display of up to 128 selected bus transactions stored in a buffer.
3. Compares the 43 bits of input data to the comparison files in the event comparators.
4. The comparators are conditioned by two delays (the pass and delay counters) to provide triggering and breakpoint capability.
5. The trigger and breakpoint may be positioned anywhere throughout the program; the breakpoint is conditioned by the event comparison and counting options.
6. Stores the binary value of up to eight external test clips in the buffer.
7. Provides measurements of program execution time (milliseconds or microseconds), emulator clocks, event occurrences, bus instructions, or bus transactions between two designated events located anywhere in the program.
8. Is compatible with all of the emulator processors that are supported by the 8001/8002A system.

RTPA Functions

This is a general discussion of the functions of the RTPA. The individual commands are covered in Section 2 of this manual. Throughout the following discussion refer to the functional block diagram shown in Fig. 1-3.

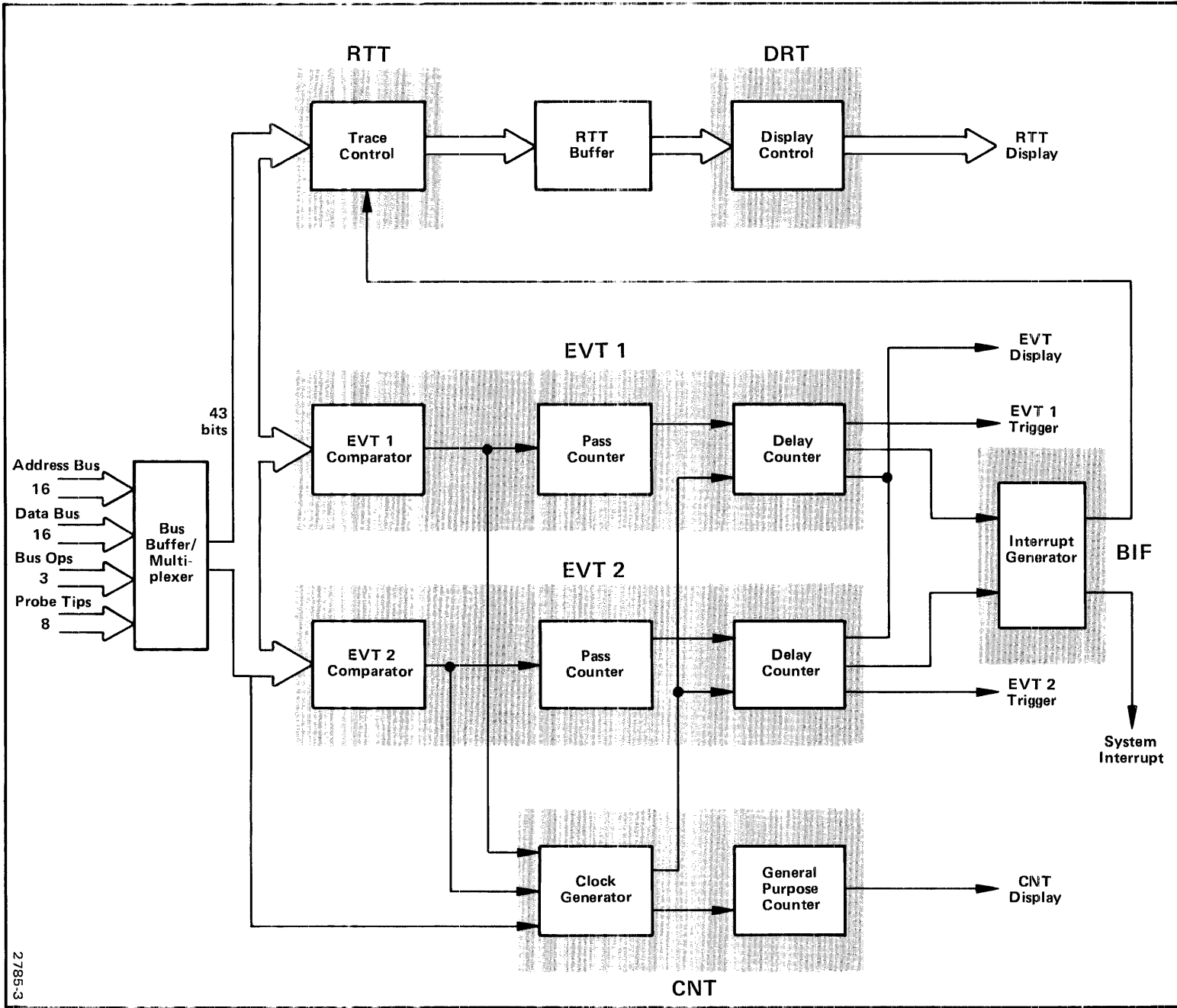


Fig. 1-3. Functional diagram of RTPA.

2785-3

The functional description that follows describes the basic functional operations within the RTPA. In some instances the functional operation does not correlate with the actual hardware circuitry. For instance, Fig. 1-3 shows the EVT1 and EVT2 comparators as two separate comparators dedicated to EVT1 and EVT2. Physically there are two comparators, an 8-bit and 16-bit comparator; however, the comparison actions for both EVT1 and EVT2 are acted upon by each of the comparators in a certain timing sequence. This will become evident when the timing relationships of the event comparators are discussed later in this section. However, for now, the functional separation of EVT1 and EVT2 actions will provide you with a better understanding of the overall operation of the RTPA.

The input data to the RTPA is temporarily stored in the buffer/multiplexer when a program is executed. The input data is simultaneously presented to the Trace Control and to both EVT1 and EVT2 comparators. The Trace Control, under software control, determines if the input data is to be stored in the RTT buffer.

The EVT comparison files, also under software control, are compared to the input data. Software control of the EVT comparison files consists of entering the EVT comparison options (address, data, test clips and bus operations) in either or both EVT1 and EVT2 comparators. The entered EVT comparison options are then compared to the input data. If they are identical, a match is made and an event trigger is generated. If they are not identical, no match is made and no trigger generated. The EVT1 and EVT2 comparators perform separate comparison functions.

When a match is made by either or both EVT1 and EVT2 comparators there are two alternatives.

1. If the pass counter and/or delay counter (under software control) is set, the EVT match is delayed by the values set into the counters.
2. If neither is set, the counters are essentially bypassed, and an event trigger occurs immediately.

When an EVT match is made, a trigger is presented to the Interrupt Generator either delayed or not delayed by the pass and delay counters. If the pass counter is set (a number entered into the pass counter), each EVT match decrements the pass counter by one. When the pass counter value is zero, the pass counter is satisfied and the delay counter is enabled if the delay counter is set.

The delay counter can be set for any of the following delays.

- real-time (milliseconds or microseconds)
- emulator clock cycles
- event occurrences (EVT1 or EVT2)
- bus instructions (fetch, bus cycles or storage cycles)

The specific delay entered into the delay counter is decremented until the delay counter reaches zero. At that time the Interrupt Generator, if set, is enabled.

The Interrupt Generator, also under software control, determines the relationships between the two EVT hardware-generated triggers and the control of program execution. When the interrupt generator receives an EVT match (either delayed or not delayed) it generates a corresponding EVT1 or EVT2 trigger. Depending on the software control settings the interrupt generator will perform one or more of the following functions:

- Does not break program execution (if EVT mode is set).
- Arms the other EVT comparator (if ARM or FRZ mode is set).
- Breaks program execution and returns control of the program to the 8001/8002A system terminal or back to the program for continued execution (if BIF mode S or C is set).
- Freezes the RTT buffer (if BIF mode is set).
- Stops the general purpose counter (if EVT or BIF mode is set).

The general purpose counter is used to count the designated units between the start of program execution and the breakpoint, or between the settings of EVT1 and EVT2. The units of count are the same as those chosen for the delay counter.

Under software control, the contents of the RTT buffer can be displayed after a program has been executed. The internal workings of the RTPA are described in more detail in the following paragraphs.

How The RTPA Works

Refer to the RTPA block diagram shown in Fig. 1-4, and to the event comparison block diagram shown in Fig. 1-6, as you read the following description.

Input Data

The input data is presented to the RTPA from the address bus, data bus, data acquisition probe, and bus operation control lines. The maximum number of data bits available at the multiplexer input is dependent on whether the active emulator processor in the 8001/8002A system is an 8-bit or a 16-bit processor. Up to a maximum of 48 bits (for a 16-bit processor) or 40 bits (for an 8-bit processor) may be present at the input of the multiplexer. Table 1-1 shows how these bits are derived.

Table 1-1
Input Data Bits to the RTPA

Input Data	8-Bit Processor	16-Bit Processor
Address—from the 8001/8002A system address bus, A0–A15.	16	16
Data—from the 8001/8002A system data bus, D0–D7 or D0–D15.	8	16
Probe Clips—the binary value of the eight probe clips from the Data Acquisition Probe. Probe clips are attached to the user's prototype system	8	8
Bus Operations—consists of three control signal lines from the 8001/8002A system: M/IO (Memory/Input-Output), R/W (Read/Write), and FETCH.	3	3
Maximum number of input data bits	35	43
Identification Bits—consists of five identification bits that are dedicated for use (internally) in the RTPA for the control of certain storage and display functions of the RTT buffer.	5	5
Maximum number of bits at the multiplexer input.	40	48

A maximum of 35 or 43 bits of input data, as indicated in Table 1-1, are temporarily stored (with the five additional identification bits) in the multiplexer during each bus transaction as a program is being executed. Any or all of the input data bits can be used by the RTPA, depending on the conditions established in the EVT comparators and the RTT buffer by the RTPA commands.

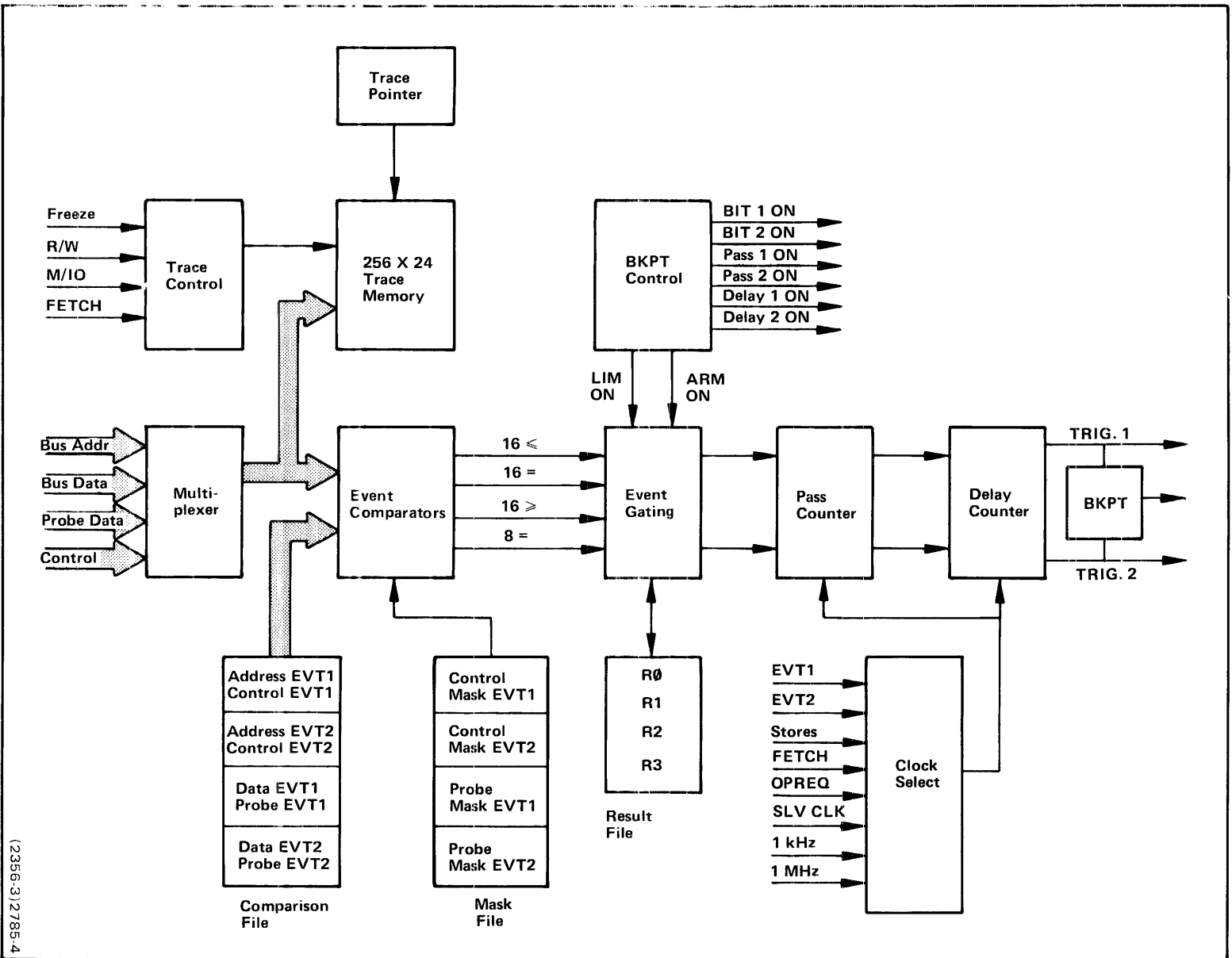


Fig. 1-4. RTPA block diagram.

(2356-3)2785-4

The multiplexer permits the selection of data bits from the 40 or 48 bits of data that are temporarily stored in the multiplexer input. The data bit selection occurs within each bus transaction as a program is being executed. During the first half of the bus transaction, the address (16 bits) and the bus operations plus identification bits (3 bits plus 5 bits) for a total of 24 bits appear at the multiplexer output. The data bits at the multiplexer output are presented to the RTT buffer and the event comparators. During the last half of the bus transaction, the data (8 or 16 bits) and probe clip bits (8 bits) for a total of 16 or 24 bits appear at the multiplexer output; these bits are also presented to the RTT buffer and event comparators. The following paragraphs describe the reactions of the RTT buffer regarding the storage of these data bits. The effects of the event comparators on the data bits are described later in this section.

Input Data Storage (RTT Buffer)

The input data (up to 43 bits) are stored in the RTT buffer during the execution of a program. The trace control and RTT buffer, under software control of the RTT command, specify which of the following bus transactions are stored; only one option can be selected at any time.

F—Instruction fetches only	IR—I/O reads only
I—I/O accesses only	IW—I/O writes only
M—Memory accesses only	MR—Memory read only
R—Read operations only	MW—Memory writes only
W—Write operations only	ALL—All bus activity

All of the storage functions of the RTT buffer are controlled by the SLV OPREQ (Slave Operation Request) signal from the 8001/8002A system. When operating in emulation Mode 0 the 8001/8002A system clock controls the timing of the SLV OPREQ signal. In emulation Modes 1 and 2 the prototype clock controls the timing of the SLV OPREQ signal. The timing diagram in Fig. 1-5 shows the storage sequence of the input data in the RTT buffer in relation to the SLV OPREQ signal. Refer to Fig. 1-5 as you read the following sequence.

1. The SEL (select) signal is low prior to the leading edge of SLV OPREQ. This provides temporary storage for the input data (16 address bits and 3 bus operation control bits) at the multiplexer output.
2. On the leading edge of SLV OPREQ, the STR (store) signal goes low, permitting the input data (at the multiplexer output) to be stored in the RTT buffer (providing the bus transaction satisfies the RTPA command requirements). Simultaneously, the address and bus operation input data are made available to the event comparators, as discussed later in this section.

3. The SEL (select) signal goes high prior to the trailing edge of SLV OPREQ. This provides temporary storage for the input data (8 or 16 bits from data bus and 8 bits from the data acquisition probe clips) at the multiplexer output.
4. On the trailing edge of SLV OPREQ, the STR (store) signal again goes low, permitting the input data (at the multiplexer output) to be stored in the RTT buffer (providing the bus transactions satisfies the RTT command requirements). Simultaneously, the data and probe input data are made available to the event comparators, as discussed later in this section.

Note in Fig. 1-4 and 1-5, and in the above sequence, that up to 24 bits of data can be stored in the RTT buffer on the first half of SLV OPREQ. Then, up to 24 bits of data can be stored in the RTT buffer on the second half of SLV OPREQ. The storage capacity of the RTT buffer is 256×24 bits; therefore, 128 bus transactions can be stored in the buffer at anytime.

The RTT buffer is a "pipeline" type of storage device. When a bus transaction is stored, the oldest transaction in the buffer is lost. The RTT buffer contents are displayed sequentially; that is, from the oldest to the most recent transaction stored in the buffer.

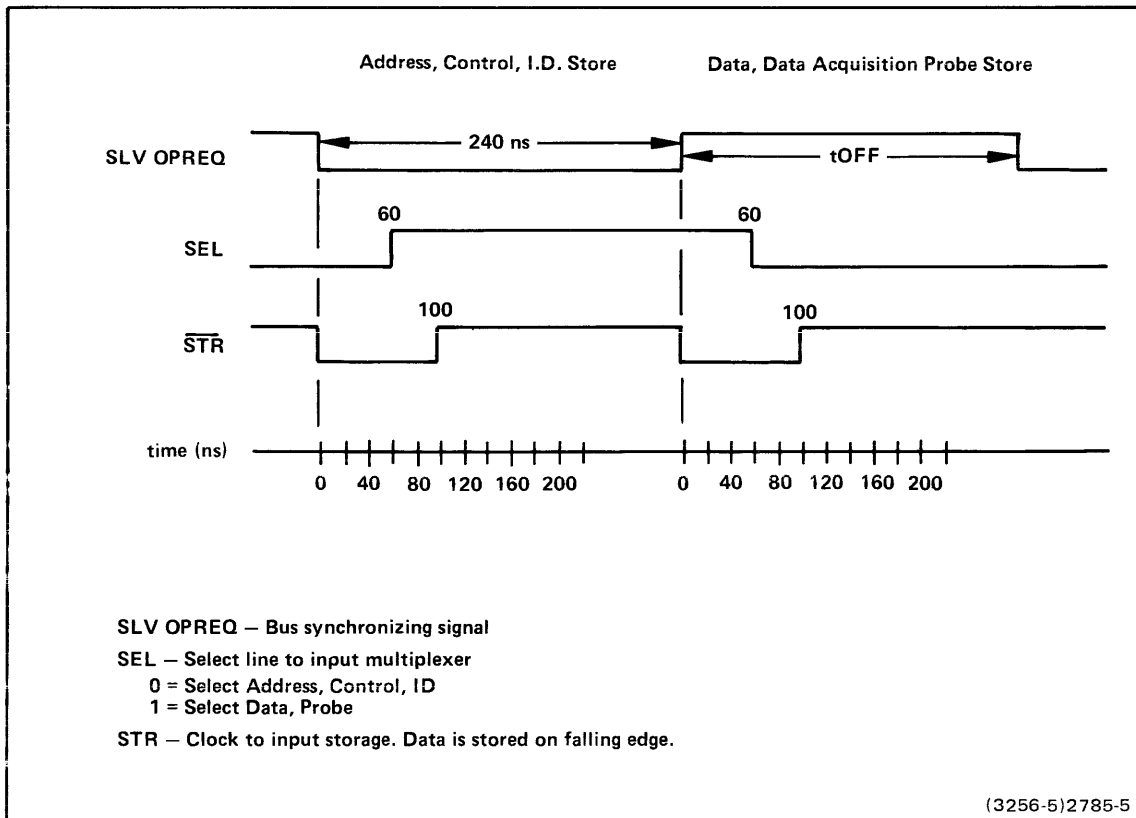
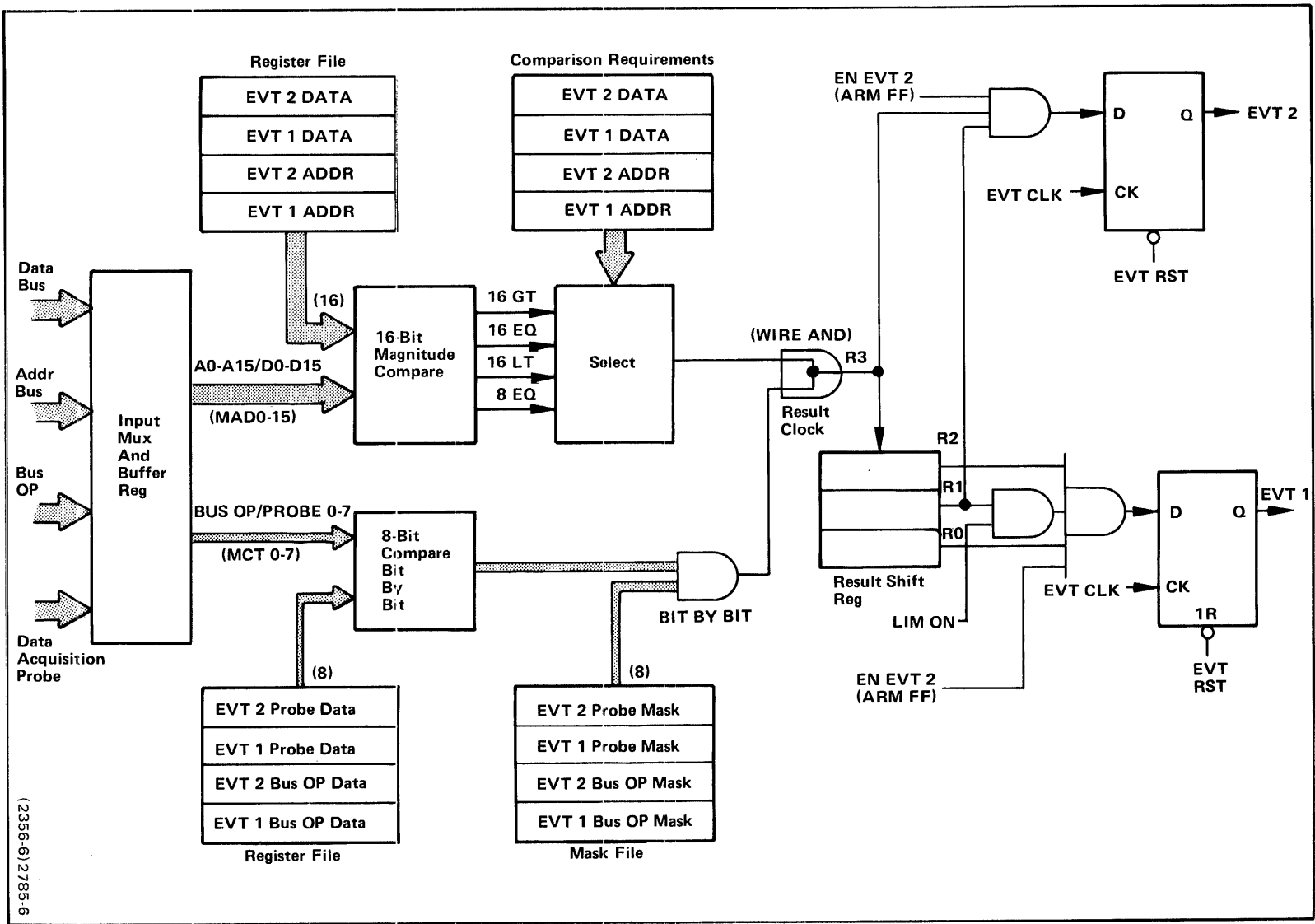


Fig. 1-5. Trace control and RTT buffer timing diagram.

Fig. 1-6. Event comparison block diagram.



Event Comparison (EVT1 and EVT2)

Recall that in the previous discussions on the RTPA functions, and in Fig. 1-3, the event comparators are referred to functionally as EVT1 comparator and EVT2 comparator. From a hardware standpoint, there are two comparators, an 8-bit and a 16-bit comparator. The two comparators perform the following operations. (Refer to Fig. 1-6).

1. The 16-bit magnitude comparator compares the input address and data bytes (at the multiplexer output) against the address and data comparison files for both EVT1 and EVT2. Note the four comparison files shown in Fig. 1-6 for the address and data. These comparison files are under software control; the address and data values are entered in the comparison files by the EVT1 and EVT2 commands. Four comparisons are made for each of the four comparison files, as follows:
 - 16-bit greater than or equal (\geq)
 - 16-bit equal (=)
 - 16-bit less than or equal (\leq)
 - 8-bit equal (:=)
2. The 8-bit comparator compares the input probe and bus operations data bits (from the multiplexer output) with the probe and bus operations comparison files for both EVT1 and EVT2. Note the four comparison files shown in Fig. 1-6 for the probe and bus operations. These comparison files are also under software control; the probe and bus operation values are entered in the comparison files by the EVT1 and EVT2 commands. A bit-by-bit comparison is performed for each of the four comparison files.

The timing relationships between the input data (at the multiplexer output) and the eight comparison files (in the event comparators) are very important in understanding how the event comparators perform their comparison functions. These timing relationships are described in the following paragraphs.

Basic Timing Relationships in the Event Comparators. There are several assumptions we must make prior to describing the sequence of events and the timing relationships in the event comparators. These assumptions are listed as follows:

- The emulator processor is active.
- The address, data, probe clip and bus operation parameters have been entered into the event comparison files with the EVT1 and EVT2 commands.
- The program is being executed.

Fig. 1-7 is a combination timing diagram and flowchart showing the comparison actions between the input data and the event comparators during one cycle of the SLV OPREQ signal. Each block represents one event in the comparison sequence. The left edge of each block shows the start of the event in a timing relationship to one cycle of the SLV OPREQ signal. For instance, the storage of input data in the RTT buffer (block #1) and the comparison of EVT1 options A and B to the input data (block #2) both start on the leading edge of the SLV OPREQ signal. The events shown in blocks #3, #4, and #5 occur approximately 60, 100, and 160 ns respectively after the leading edge of the SLV OPREQ signal. Blocks #6 through #10 have the same timing relationship to the trailing edge of the SLV OPREQ signal. During one cycle of the SLV OPREQ signal the following sequence of events takes place:

1. On the leading edge of the SLV OPREQ signal, the input data (address—16 bits and bus operations—3 control bits) are stored simultaneously in the RTT buffer and in the two comparator registers. The address byte (16 bits) is temporarily stored in the magnitude comparator register and the bus operations bits (3 bits) are temporarily stored in the bit-by-bit comparator register.
2. The "00" registers of all the comparators are accessed; the input data stored in the comparator registers is compared to the address and bus operation values entered (with the EVT1 command) in the EVT1 comparison files. Refer to Fig. 1-8, which explains blocks #2 and #3 of Fig. 1-7 in greater detail.
3. If a match occurs between the input data and the comparison files, the results of both comparators are clocked to the "RO" register of the result register. See Figures 1-7 and 1-8.
4. Approximately 100 ns following the leading edge of SLV OPREQ, the "01" registers of all comparators are accessed. The input data (address byte and bus operation bits), still temporarily stored in the comparator registers, is compared to the address and bus operation values entered (with the EVT2 command) in the EVT2 comparison files. The procedure is the same as that depicted in Fig. 1-8, except with EVT2 option A and B parameter values.
5. If a match occurs the results of both comparators are clocked to the "R1" register of the result register. See Fig. 1-7.
6. On the trailing edge of SLV OPREQ the input data (data bus—8 or 16 bits and probe data—8 bits) are stored in the RTT buffer and also temporarily stored in the two comparator registers. The data bus is temporarily stored in the magnitude comparator register, and the probe data is temporarily stored in the bit-by-bit comparator register. See blocks #6 and #7 of Fig. 1-7.
7. The "10" registers of all the comparators are accessed; the input data (data bus and probe data), temporarily stored in the comparator registers, is compared to the data and probe values entered (with the EVT1 command) into the EVT1 comparison files. Refer to Fig. 1-7.
8. If a match occurs the results of both comparators are clocked to the "R2" register of the result register.

9. Approximately 100 ns following the trailing edge of SLV OPREQ, the "11" registers of all the comparators are accessed. The input data (data bus and probe data), still temporarily stored in the comparator registers, is compared to the data and probe values entered (with the EVT2 command) into the EVT2 comparison files.
10. If a match occurs the results of both comparators are clocked to the "R3" register of the result register.

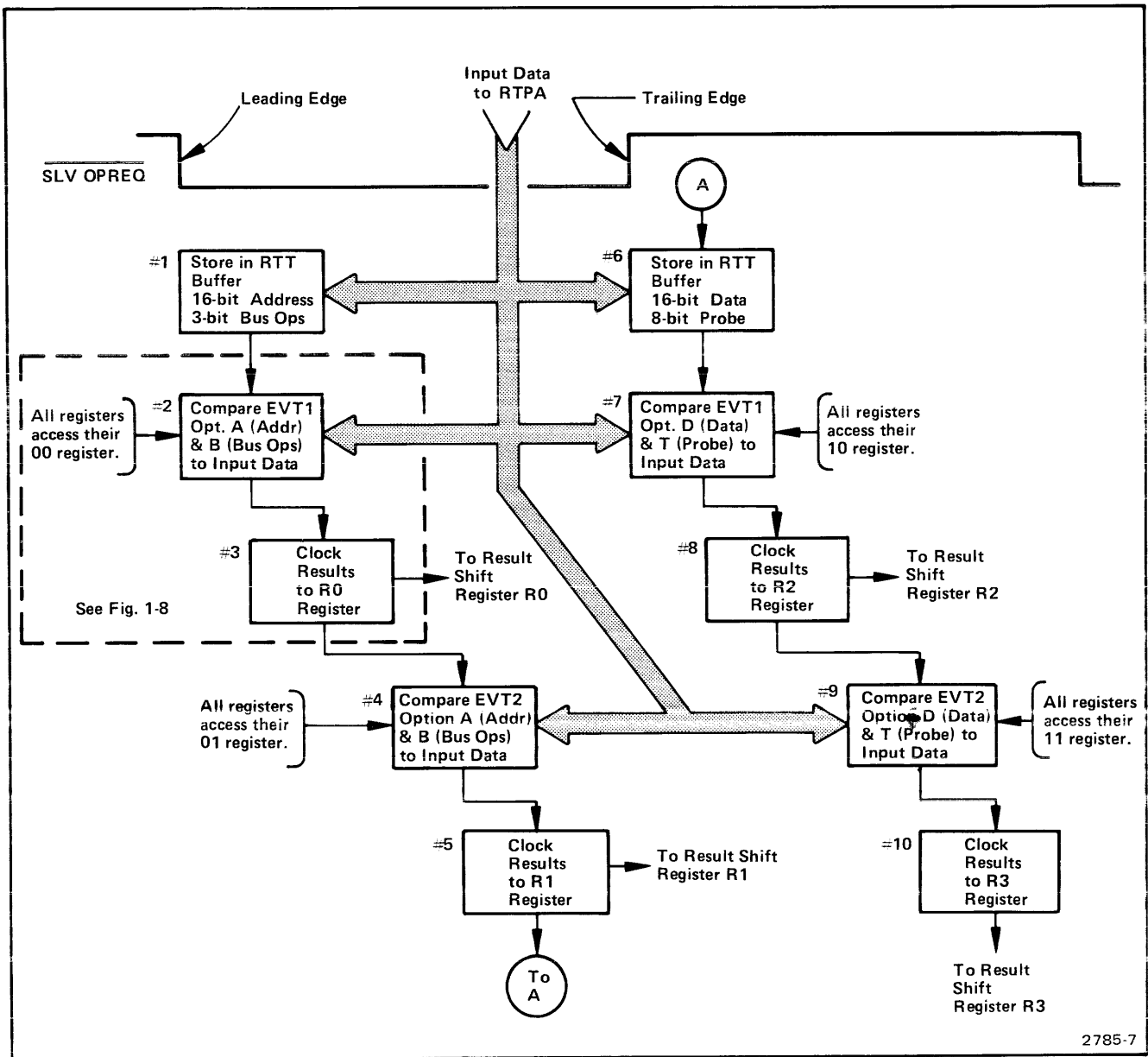


Fig. 1-7. Flow chart showing timing relationships with SLV OPREQ signal.

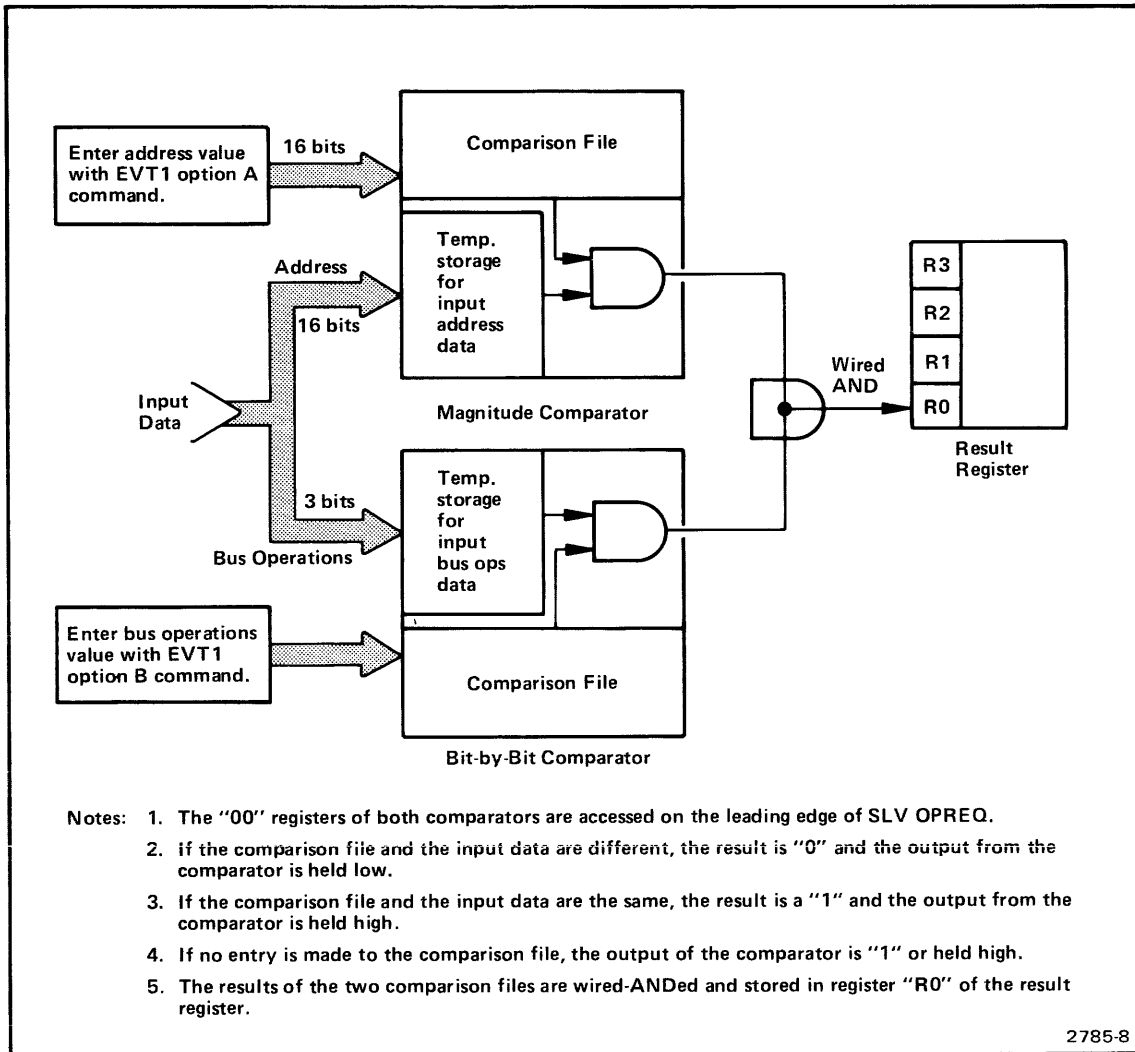


Fig. 1-8. Comparison of EVT1 address and bus operation options.

Result Register and EVT Flip-Flops. The results of the magnitude and bit-by-bit comparators are stored in the result registers. Fig. 1-9 is a simplified block diagram depicting the relationship between the stored information in the result registers and the EVT1 and EVT2 flip-flops.

The information stored in the result registers is defined in Table 1-2.

**Table 1-2
Result Registers**

Register	If a Match is Present or if No Entry is Made To Comparison Files	If NO Match Is Made	EVT (Event) Options Compared
R0	1 or Logic HIGH	0 or logic LOW	EVT1 address and bus operations (EVT options A and B)
R1	1 or logic HIGH	0 or logic LOW	EVT2 address and bus operations (EVT options A and B)
R2	1 or logic HIGH	0 or logic LOW	EVT1 data bus and probe (EVT options D and T)
R3	1 or logic HIGH	0 or logic LOW	EVT2 data bus and probe (EVT options D and T)

From Table 1-2 you will note that the registers R0 and R2 contain information on the status of the EVT1 comparison options; likewise, registers R1 and R3 contain information on the status of the EVT2 comparison options.

If the comparison options of EVT1 (R0 and R2) are satisfied (a match is made between the EVT1 comparison files and the input data), the EVT1 flip-flop provides a trigger to the EVT1 pass and delay counters. In the same manner, if the comparison options of EVT2 (R1 and R3) are satisfied, the EVT2 flip-flop provides a trigger to the EVT2 pass and delay counters.

In Fig. 1-9, the state of the signal line "LIM ON" is set by either the EVT LIM or BIF LIM command. These two operational modes will be described in detail later in Section 2. For the present, the LIM mode command sets the LIM ON signal line to a "1" or logic HIGH. This places an additional requirement on the triggering of the EVT1 flip-flop. When the LIM ON line is set HIGH, the information in result register R1 (EVT2 address and bus operations) must also be a logic HIGH, in addition to registers R0 and R2, before the EVT1 flip-flop can be triggered. This means that EVT1 options A, B, D, and T plus EVT2 options A and B, must all match the data input during one cycle of the SLV OPREQ signal before the EVT1 flip-flop can provide a trigger to the pass and delay counters.

NOTE

The operation of the EVT LIM and BIF LIM commands, described later in Sections 2 and 3, are misused and misunderstood more than any of the other RTPA commands. If you have a good grasp on the above description of the effects of the LIM ON signal line in the triggering of the EVT1 flip-flop, the descriptions on the LIM commands contained in Sections 2 and 3 should be easier to understand.

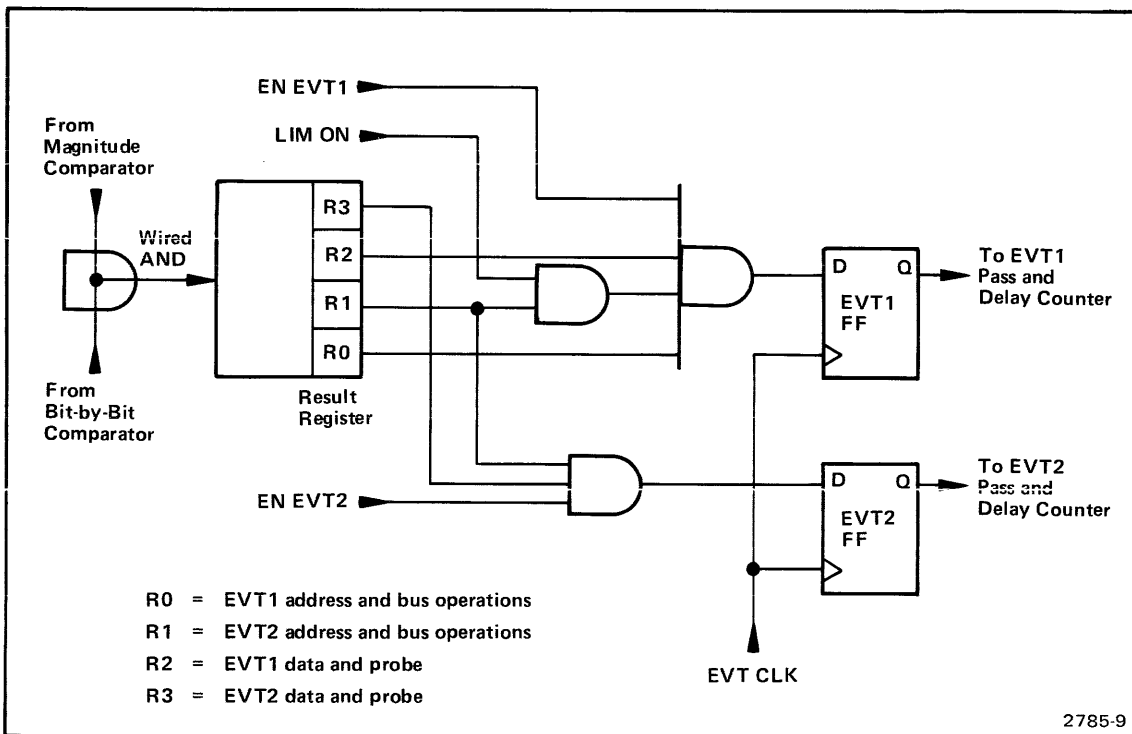


Fig. 1-9. Result register and EVT flip-flops simplified block diagram.

Event Pass and Delay Counters

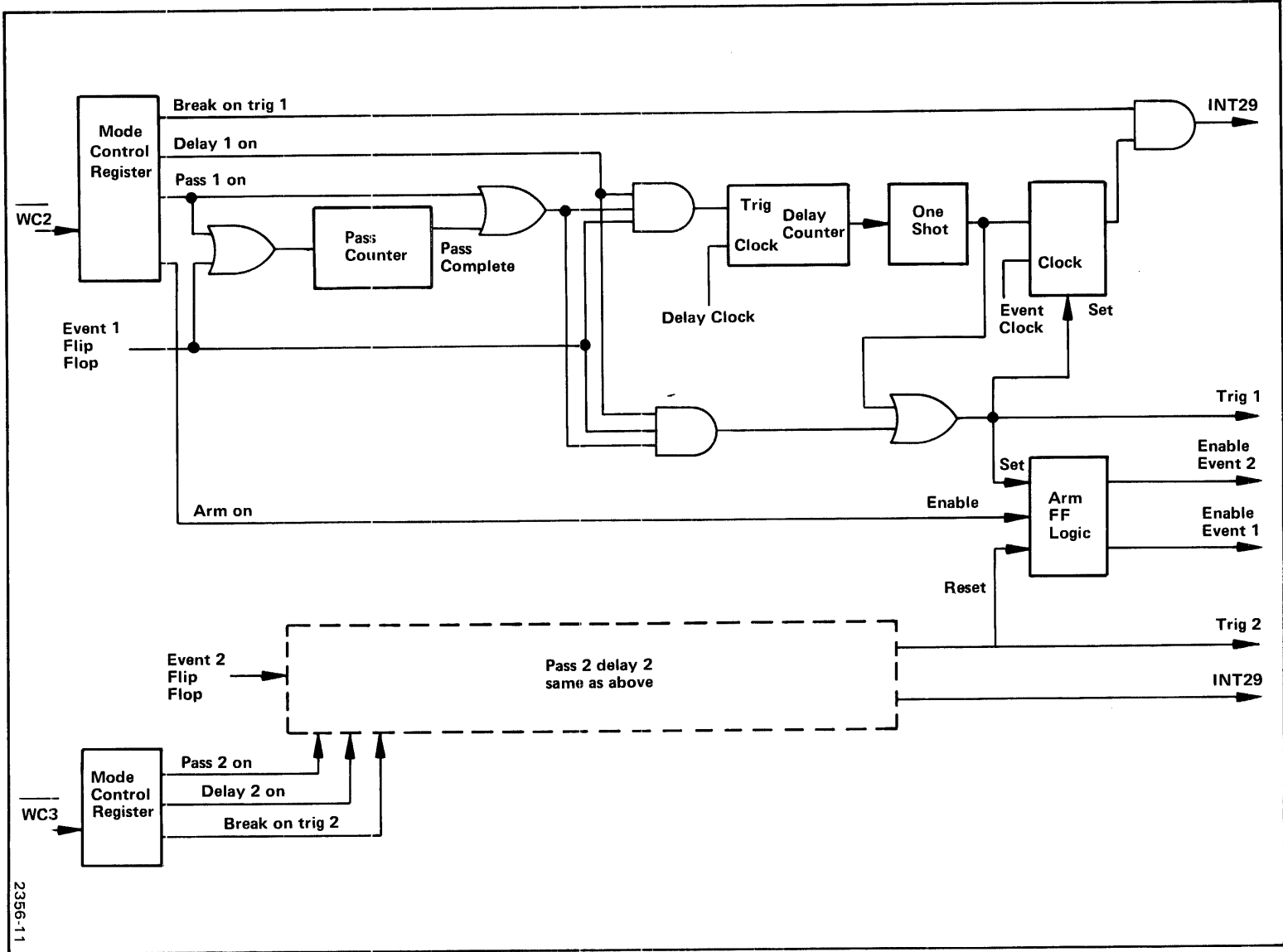
The EVT1 pass and delay counters are identical to the EVT2 pass and delay counters; therefore, the following comments and circuitry shown in Fig. 1-10 will be directed to the EVT1 pass and delay counters only. The pass and delay counters present a delay to the trigger from the EVT1 flip-flop. (Refer to Fig. 1-10.) Either counter may be set; a value is entered with the EVT1 option P and C command. If a counter is not set it will be bypassed. The following delay combinations may be specified:

1. The pass and delay counters are not set. (EVT1 options P and C command are not specified.) If the pass and delay counters are not set, the trigger from the EVT1 flip-flop generates Trigger 1. If the breakpoint is set on EVT1, an interrupt is generated to the system to stop the program execution.
2. The pass counter is set and the delay counter is not set. (EVT1 option P command is entered; EVT1 option C command is not specified.) The trigger from the EVT1 flip-flop enables the pass counter circuitry. Each time an EVT1 trigger is generated (from the EVT1 flip-flop) the pass counter is decremented by one. When the pass counter reaches zero, its output generates Trigger 1, since the delay counter is not set. If the breakpoint is set on EVT1, an interrupt is generated to the system to stop program execution.
3. The pass and delay counters are both set. (Both EVT1 options P and C commands are entered.) The pass counter operation is the same as in step 2, above. When the pass counter reaches zero, the delay counter circuitry is enabled and the delay count is started. The units of delay are specified with the CNT (Count) command. This command is described in detail in Section 2. The following units of delay may be specified:
 - real-time (milliseconds or microseconds)
 - emulator clock cycles
 - event occurrences (either EVT1 or EVT2)
 - bus instructions (fetch, bus cycles, or storage cycles)

When the delay counter reaches zero, its output generates Trigger 1. If the breakpoint is set on EVT1, an interrupt is generated to the system to stop program execution.

4. The pass counter is not set and the delay counter is set. (EVT1 option C command is entered; EVT1 option P command is not specified.) The trigger from the EVT1 flip-flop bypasses the pass counter (since it is not set) and enables the delay counter, starting the delay count. When the delay counter reaches zero, its output generates Trigger 1. If the breakpoint is set on EVT1, an interrupt is generated to the system to stop program execution.

Fig. 1-10. Pass and delay counter simplified block diagram.



2356-11

Event Triggers

Event triggers 1 and 2 perform the following functions:

1. When a trigger is generated and a breakpoint is set for that trigger, the trigger causes an interrupt to be generated to the system, and stops program execution.
2. When either trigger is generated, it is available at the two BNC connectors on the Data Acquisition Interface panel, installed in the rear panel of the 8001/8002A system. These connectors, labeled EVENT TRIG OUT 1 and EVENT TRIG OUT 2, are used to provide trigger pulses to external equipment, such as an oscilloscope or logic analyzer. See Fig. 1-11 for electrical characteristics of the trigger pulse.
3. The triggers are also used within the RTPA to: enable the other event comparator; enable, disable, or reset the general purpose counter; and freeze the RTT buffer.

System Interrupts

The preceding discussion stated that the system interrupts are generated by the event triggers. The BIF command (described in detail in Section 2) establishes conditions between the two triggers that must be met before a system interrupt is generated. Each of the BIF command modes imposes conditions upon a system interrupt, as shown in the following table.

BIF Command	Conditions for Interrupt
BIF 1	Only Trigger 1 generates the interrupt.
BIF 2	Only Trigger 2 generates the interrupt.
BIF ARM	Trigger 1 enables EVT2 comparators. Trigger 2 generates the interrupt.
BIF IND	Either Trigger 1 or Trigger 2 can generate the interrupt.
BIF LIM	Only Trigger 1 generates the interrupt.
BIF FRZ	The same conditions as BIF ARM, above.

Data Acquisition Interface

The Data Acquisition Interface unit provides an interconnection between the prototype (via the Data Acquisition Probe leads) and the RTPA. Fig. 1-11 is a simplified block diagram of the Data Acquisition Interface. Any or all eight of the Data Acquisition Probe clips can be connected to any data source in the user's prototype or to any other data source. The Data Acquisition Probe clock channel should be connected to a clock source that is related to the data source of the eight probe clips. The probe clock channel is used to clock the storage register in the Data Acquisition Interface, transferring the data from the probe channels to the input of the Real-Time Trace module.

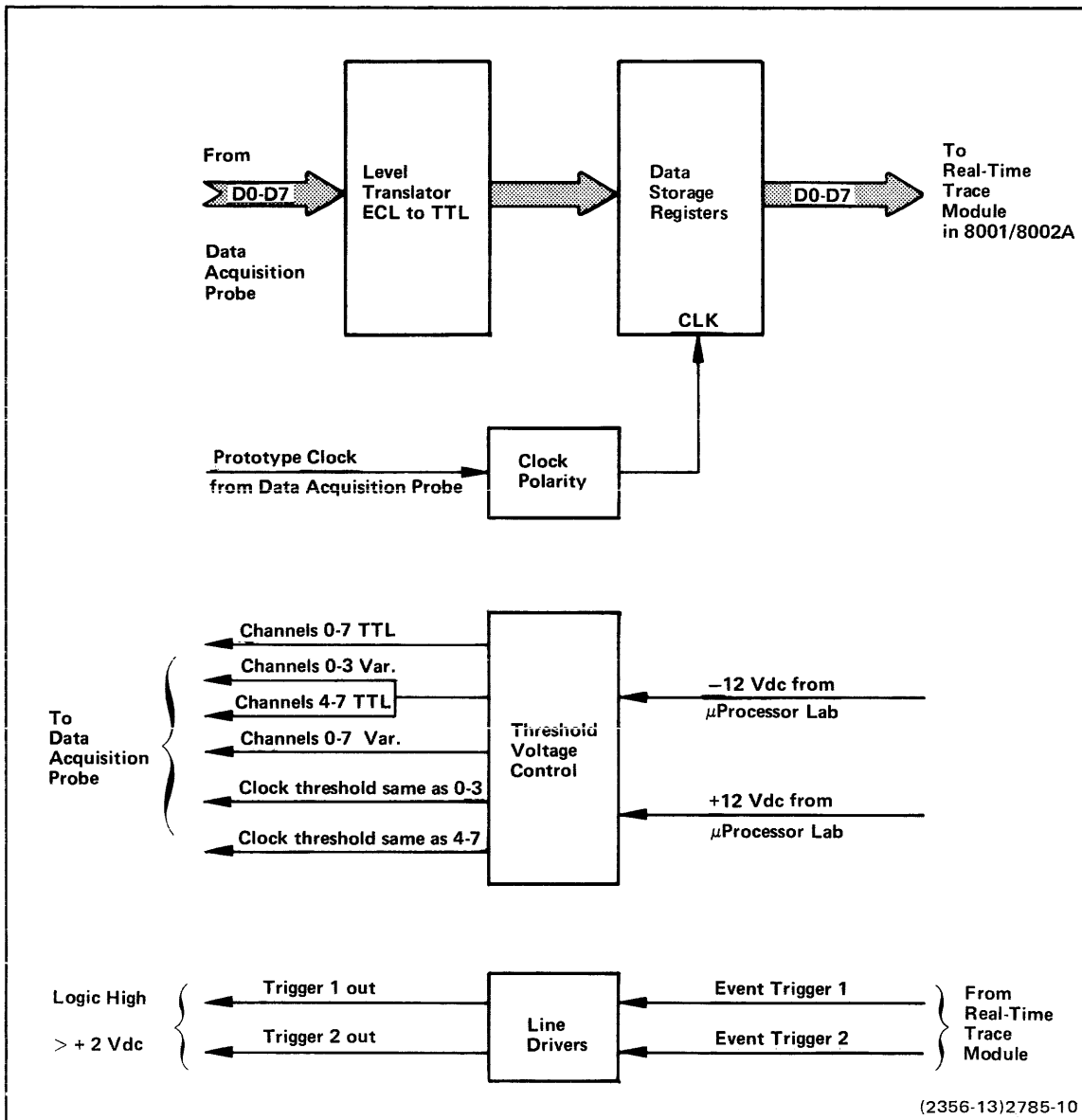


Fig. 1-11. Data Acquisition Interface simplified block diagram.

The controls and connectors on the Data Acquisition Interface panel are shown in Fig. 1-12. The function of these controls and connectors are defined in the following table. (Refer to Figures 1-11 and 1-12.)

Control/Connector	Function												
EVENT TRIG OUT 1 and EVENT TRIG OUT 2	The two BNC connectors provide connections for triggering external equipment, such as an oscilloscope or logic analyzer. The logic HIGH output of the trigger is $>+2$ Vdc.												
VAR ADJ and VAR MON	The Variable Threshold Adjust is a screwdriver adjustment of the threshold voltage. A voltmeter connected to the Monitor Jack permits monitoring the adjustable voltage of +10 volts to - 10 volts.												
Threshold Level Switch	This three-position switch provides 8 variable channels, 8 fixed TTL (+1.4 volts) channels, or a combination of 4 fixed TTL and 4 variables channels.												
	<table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: left;">Switch Position</th> <th style="text-align: center;">TTL Channels</th> <th style="text-align: center;">Variable Channels</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">Up</td> <td style="text-align: center;">-----</td> <td style="text-align: center;">0-7</td> </tr> <tr> <td style="text-align: center;">Center</td> <td style="text-align: center;">4-7</td> <td style="text-align: center;">0-3</td> </tr> <tr> <td style="text-align: center;">Down</td> <td style="text-align: center;">0-7</td> <td style="text-align: center;">-----</td> </tr> </tbody> </table>	Switch Position	TTL Channels	Variable Channels	Up	-----	0-7	Center	4-7	0-3	Down	0-7	-----
Switch Position	TTL Channels	Variable Channels											
Up	-----	0-7											
Center	4-7	0-3											
Down	0-7	-----											
EXT CLK PLRT	The External Clock Polarity switch permits selecting either the rising or falling edge of the clock from the Data Acquisition Probe.												
PROBE	A 25-pin connector for connecting the Data Acquisition Probe to the Data Acquisition Interface panel.												

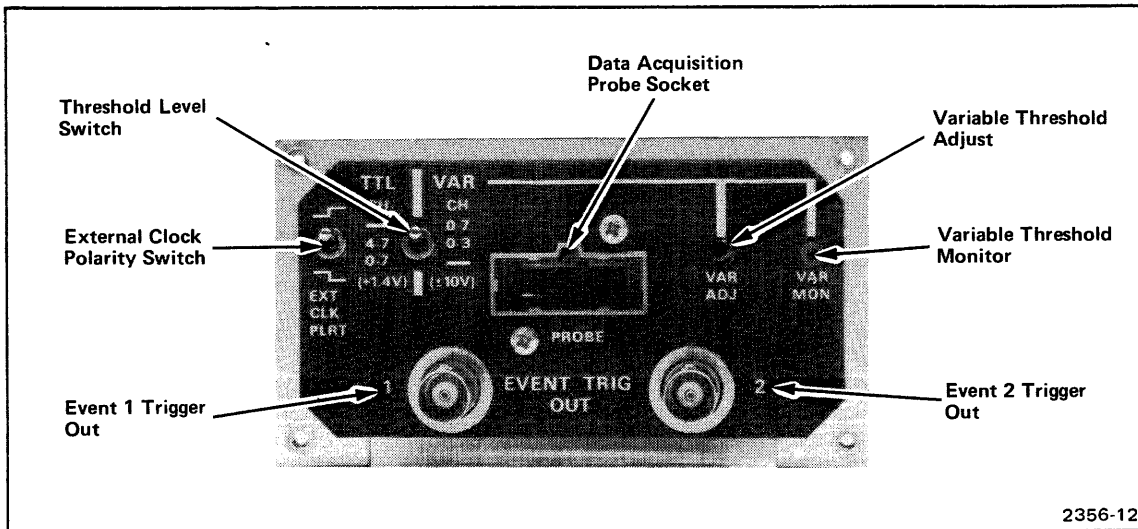


Fig. 1-12. Data Acquisition Interface panel controls and connectors.

RTPA TRIGGERING MODES

Pre-, Center, or Post-Triggering

The RTT buffer has the capacity to store 128 bus transactions at any given time. During the execution of a program, data is continually being stored in the buffer. With the proper RTPA commands specified, the RTT buffer can be directed to store data around a certain event, thereby simulating pre-, center, or post-triggering conditions similar to those obtained with a logic analyzer.

Recall the data window we talked about at the beginning of this section. It is shown again in Fig. 1-13. The "event" shown in Fig. 1-13 refers to a specific bus transaction or probe data information that you are interested in. Any one or all of the EVT comparison options (A, B, D, and T) for either EVT1 or EVT2 can be used to identify this event.

Several combinations of RTPA commands may be used to direct the RTT buffer to store data around a specified event, as shown in Fig. 1-13. The following paragraphs describe briefly how this may be accomplished. The various triggering modes for pre-, center, or post-triggering are described in detail, with examples of each mode, in Section 3, Procedure 7.

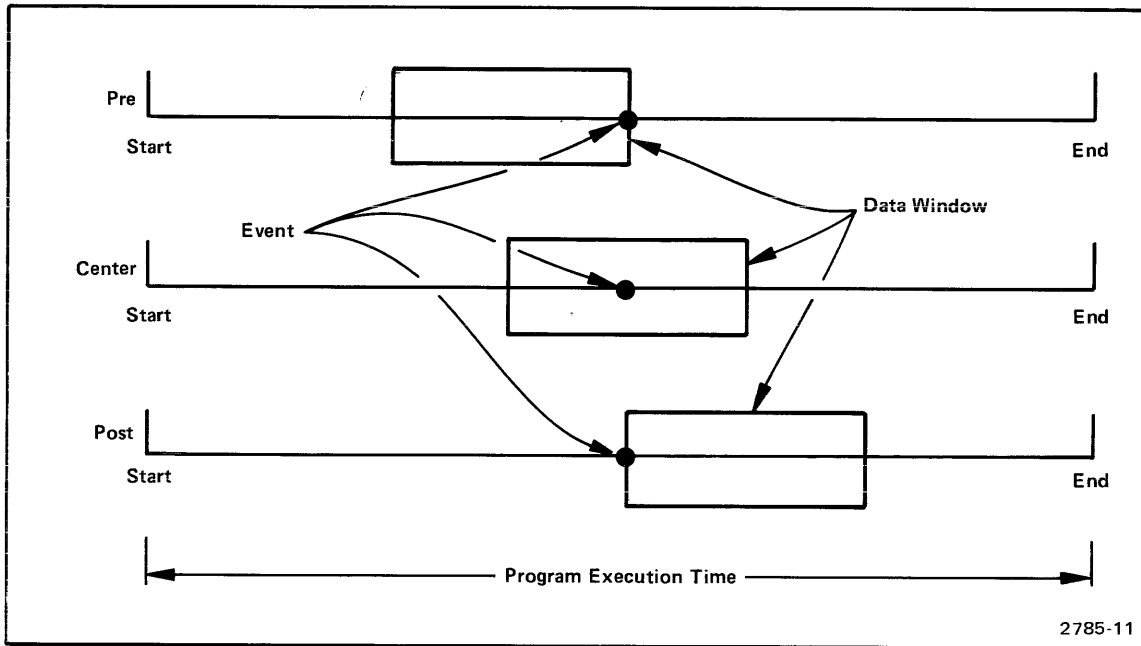


Fig. 1-13. RTPA Pre-, Center, and Post-triggering modes.

Pre-Triggering

In pre-triggering, the breakpoint is set at the desired event, using any one or all of the EVT comparison options (options A, B, D, or T). Either EVT1 or EVT2 can be used to identify the event. When the breakpoint is reached, the trigger (1 or 2) will generate a program interrupt and freeze the RTT buffer. When the RTT buffer is frozen, it contains the 128 bus transactions immediately preceding the breakpoint (desired event). This establishes a pre-trigger condition, as shown in Fig. 1-13.

The BIF FRZ command may also be used to establish a pre-trigger condition. The EVT1 comparison options should be set for the desired event. The BIF FRZ command freezes the buffer when the EVT1 trigger is generated, thus producing the same buffer contents as the preceding paragraph.

Center Triggering

In center triggering, the EVT1 or EVT2 comparison options (A, B, D, or T) are again set to the desired event. The delay counter (EVT option C) should be set to 64 bus cycles. This means that when the desired event occurs, the breakpoint will be delayed 64 bus cycles. The breakpoint will generate a program interrupt and freeze the RTT buffer. Since the buffer holds 128 cycles, the desired event will appear in the center of the RTT buffer.

Post-Triggering

In post-triggering, the EVT1 or EVT2 comparison options are again set for the desired event. The delay counter (EVT option C) should be set to 128 bus cycles. When the desired event occurs, the breakpoint will be delayed 128 bus cycles, and the complete buffer will be filled after the desired event has passed.

RTT Storage

The preceding discussion of the RTPA triggering modes assumes, that the software control of the RTT buffer (RTT command) is set to "ALL", with all bus transactions being stored in the RTT buffer. It also assumes, that the software control for the units of delay (CNT command) is set to "C", where all bus transactions are counted.

If the RTT command is set to any other bus transaction option, only the specified transactions will be stored in the RTT buffer. (See earlier in this section, "Input Data Storage—RTT Buffer".) If the RTT command is set to a specific bus transaction the CNT command option should be changed to "CNT T". The CNT T command counts only those transactions stored in the RTT buffer, rather than counting each bus transaction, as does the CNT C command.

EMULATION MODES

The primary application for the RTPA when used in conjunction with the 8001/8002 system is in the integration of the software/hardware design and development phases of a microprocessor-based system. In a typical design situation, both the software and hardware design teams agree from the beginning on the division of the design responsibilities. Both of the teams then develop, test, and evaluate their part of the phototype system. Then, both of the teams must bring their efforts together for the final integration that will result in a completed microprocessor-based prototype system.

The RTPA is a valuable aid to the software/hardware designers throughout the prototype design process:

- To the software designer during all the phases of the software design.
- To the hardware designer in the final phases of the hardware design.
- To both the software and hardware designers in the final integration phases of the software/hardware in the prototype system.

Three emulation modes are used throughout the design to accomplish the above objectives: emulation Modes 0, 1, and 2. A description of these modes is contained in the following paragraphs.

Emulation Mode 0

Emulation Mode 0 is used primarily by the software designers and evaluators in debugging the user's prototype program with the RTPA commands. In Mode 0, the prototype program is loaded into the 8001/8002A program memory. The program is executed under the full control of the active emulator processor within the 8001/8002A system. As such, all of the features of the DEBUG and RTPA commands may be fully utilized.

The Data Acquisition Probe (EVT option T) is not normally used when operating in emulation Mode 0, since the program is executed wholly within the 8001/8002A program memory. However, the Data Acquisition Probe feature remains enabled during emulation Mode 0 and can be used if an 8-bit data source with a related clock signal is connected to the Data Acquisition Probe clips.

In emulation Mode 0, the prototype program can be debugged, with the exception of the interrupt-driven routines that can only be exercised in emulation Modes 1 and 2.

Emulation Mode 1

When operating in emulation Modes 1 or 2, the microprocessor chip in the prototype is removed and the active emulator processor's Prototype Control Probe is inserted in the microprocessor socket of the prototype system. This gives the active emulator processor full debug capabilities and control over the prototype system.

In emulation Mode 1, the first steps are encountered in the integration of the prototype program (software) with the prototype system (hardware). Any portion or all of the prototype program located in the 8001/8002A program memory can be mapped and moved to the prototype system program memory. The prototype system clock is used to control the execution of the prototype program in either the 8001/8002A program memory or the prototype system program memory.

When emulation Mode 1 is invoked, the prototype program can be further debugged by executing all the interrupt routines of the program. At this time when the prototype program is being executed in the 8001/8002A program memory, the main differences between Modes 0 and 1 are in the source of the clock and in the execution of the interrupt driven routines.

In emulation Mode 1, the memory mapping capabilities of the 8001/8002A system can be used to map (transfer) the various portions of the prototype program to the prototype's program memory for integration with the prototype hardware. When all of the prototype program is mapped and moved to the prototype's program memory, the program runs entirely within the prototype. This condition simulates emulation Mode 2.

When operating in emulation Mode 1 the RTPA commands can be used to:

1. Set the event breakpoints throughout the program.
2. Make various counting and timing measurements.
3. Write-protect designated portions of the program.
4. Establish additional event comparison parameters, up to 8 data bits from the test probe clips can be connected to the prototype hardware.
5. Display the contents of the RTT buffer showing the actual execution of the program in the environment of the 8001/8002A system program memory and/or the prototype system program memory.

Emulation Mode 2

In this mode, the prototype program is contained entirely within the prototype's memory. The prototype clock controls program execution with the active emulator in the 8001/8002A system retaining overall control of the prototype, since the Prototype Control Probe is still connected to the microprocessor socket in the prototype. The RTPA commands can be fully utilized in this mode.

Emulation Mode 2 is generally used in the final software/hardware integration phases of the prototype system design, or in troubleshooting hardware systems when the software program is known to be operational.

USING THE RTPA WITH A LOGIC ANALYZER

The debugging capabilities of the RTPA are further increased during the design and integration phases of emulation Modes 1 and 2 by using a logic analyzer. The Tektronix 7D01 Logic Analyzer, when connected to the user's prototype system, will provide timing and state table displays of up to 16 binary states.

The logic analyzer operates in either synchronous or asynchronous mode. It has a storage buffer similar to the RTT buffer in the RTPA. The logic analyzer can be triggered by the RTPA to establish a desired data window. This is especially important when you are trying to create a data window (data obtained from the logic analyzer probe clips) that is time-related to the executing program, but does not have unique features to trigger on.

The two BNC connectors on the Data Acquisition Interface unit installed in the back panel of the 8001/8002A (labeled EVENT TRIG OUT 1 and EVENT TRIG OUT 2) are used to obtain a pulse from the RTPA to trigger the logic analyzer. A BNC cable is connected to either TRIG OUT 1 or 2, depending on which EVT trigger you desire.

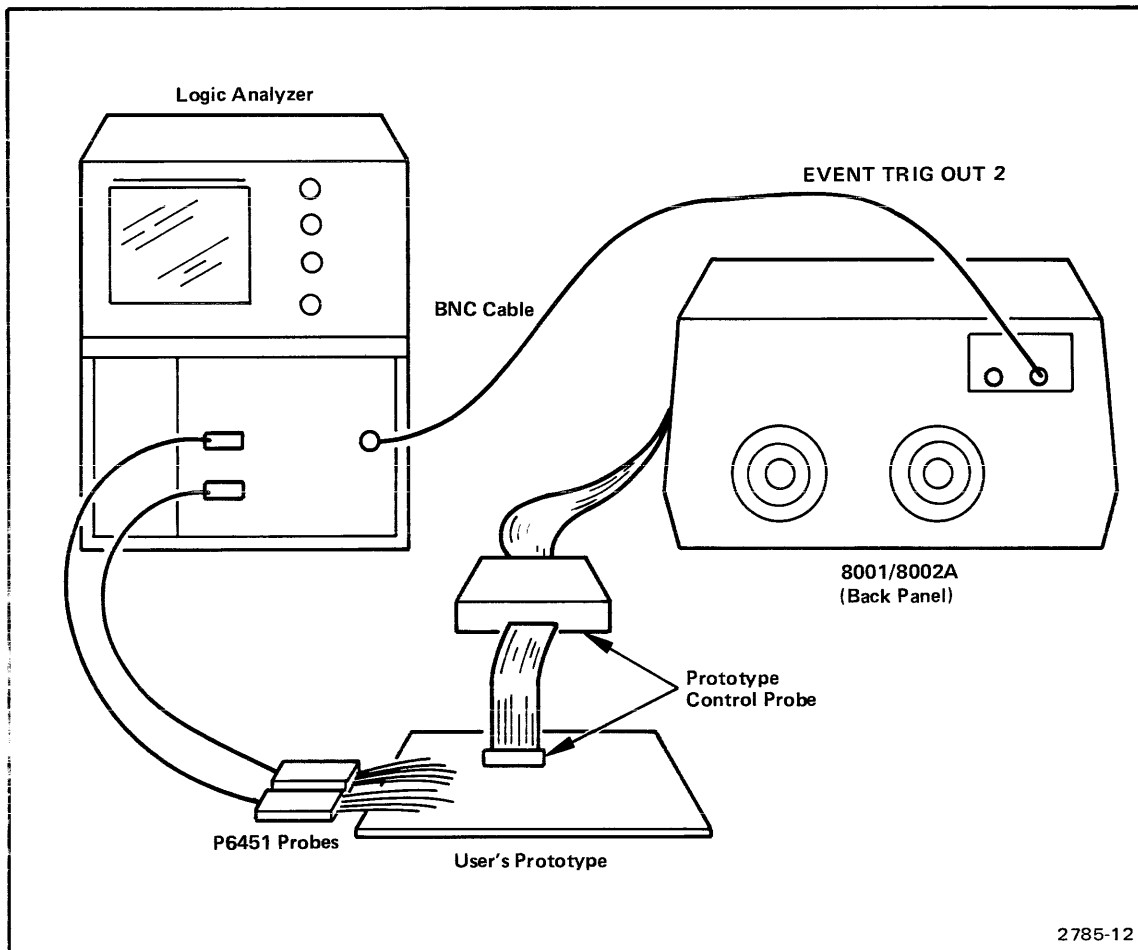


Fig. 1-14. Typical interconnection showing the RTPA triggering a logic analyzer.

Fig. 1-14 is a typical interconnection showing the RTPA triggering a logic analyzer.

CAUTION

When using the logic analyzer with the RTPA, proper grounding of all units must be made to eliminate ground loops and reduce the susceptibility of static discharges. See Appendix C for proper grounding instructions.

Practical examples and applications using the RTPA to debug the user's software and hardware design in emulation Modes 0, 1, and 2 are contained in Section 5 of this manual.

Section 2 RTPA COMMANDS

	Page		Page
Introduction	2-1	RTT Command	2-34
Command Conventions	2-1	Purpose	2-34
Command Line	2-1	RTT With no Parameters	2-34
Command Name	2-1	RTT With Parameters	2-34
Parameters	2-2	RTT Options and the RTT Buffer	2-35
Delimiters	2-2	RTT Storage	2-35
Braces and Brackets	2-2	DRT Command	2-36
Stacked Parameters	2-2	Purpose	2-36
Trailing Dots	2-3	DRT With no Parameters	2-36
Description of RTPA Commands	2-4	DRT With Parameters	2-36
RTPA Command Name Summary	2-4	DRT Display	2-36
Relationship Between Command Names and Functions	2-4	CNT Command	2-38
EVT Command	2-6	Purpose	2-38
Purpose	2-6	CNT With no Parameter	2-38
EVT With no Parameters	2-7	CNT With Parameter	2-39
EVT With Parameters	2-7	Resetting the Counter	2-39
EVT Option Parameters	2-7		
EVT CLR (Clear) Parameter	2-11		
EVT Mode Parameters	2-12		
EVT Options	2-12		
EVT Option List	2-12		
EVT Comparison Options	2-14		
EVT Counting Options	2-16		
EVT Modes	2-17		
EVT ARM	2-17		
EVT LIM (Limit)	2-20		
EVT IND (Independent)	2-20		
BIF Command	2-22		
Purpose	2-22		
BIF With no Parameters	2-23		
BIF With Parameters	2-23		
BIF Mode Parameters	2-23		
Effects of BIF Mode on the General Purpose Counter	2-26		
BIF CLR (Clear) Parameters	2-26		
BIF Return-Option Parameters	2-26		
BIF Modes	2-26		
BIF 1 and BIF 2	2-27		
BIF ARM	2-27		
BIF LIM (Limit)	2-29		
BIF IND (Independent)	2-32		
BIF FRZ (Freeze)	2-33		

ILLUSTRATIONS

Fig. No.			
2-1	Relationship between RTPA command names (software control) and functional blocks (hardware) within the RTPA		2-5
2-2	Relationship between EVT1 comparison and counting options		2-9
2-3	Comparison and counting options flowchart for EVT1		2-10
2-4	Flowchart for EVT ARM mode		2-19
2-5	Flowchart for EVT LIM mode		2-21
2-6	Flowchart for BIF ARM mode		2-28
2-7	Flowchart for BIF LIM mode		2-30

TABLES

Table No.			
2-1	EVT Option List		2-13
2-2	EVT Option Operator Signs		2-14
2-3	EVT Mode Parameters		2-18
2-4	BIF Mode Parameters		2-25

Section 2

RTPA COMMANDS

INTRODUCTION

This section contains the following:

- a description of the RTPA command conventions.
- a list of all RTPA commands.
- a detailed description of each RTPA command.

Examples are included in this section only where necessary for additional clarity. You should thoroughly study the detailed description of each command contained in this section before attempting to enter the RTPA commands. The examples in Section 3, "How to use the RTPA Commands", demonstrate the capabilities of each command; that section also reviews the effect of combining commands.

Appendix B lists the RTPA commands with an abbreviated description of each command. Once you are an experienced RTPA user, this appendix will provide you with a ready reference when using the RTPA commands on an intermittent basis.

COMMAND CONVENTIONS

Command Line

An RTPA command line contains a command name, and in most instances, one or more parameters; in some cases, parameters are separated by an operator sign.

Command Name

A minimum set of characters is required for each command. This minimum set (called the short form) and the maximum set (called the long form) are shown in the syntactical description for each command name; the minimum set of characters is underlined. Any number of characters in the command name, ranging from the short form spelling to the long form spelling, may be used, as long as the exact spelling is followed. Throughout this manual the command short form is used.

Parameters

The parameters (controlling conditions) of each command line are shown in the syntactical description for each command name. When the parameter is shown capitalized it must be entered exactly as shown. A parameter shown in lowercase letters is a descriptive term that signifies a specific type of entry or list of options that may be entered.

Delimiters

The command name and parameters in the command line must be separated by delimiters. A space or comma is used as the main delimiter between the command name and the parameters, or between two parameters. Parameters that contain an option and option value are separated by one of the following operator signs:

- = equal to
- = 16-bit equality
- : 8-bit equality
- > 16-bit greater than or equal
- < 16-bit less than or equal

Braces and Brackets

Parameters enclosed in braces, { }, must be present in the command line. Parameters enclosed in brackets [] are optional. Braces and brackets are used only for syntactical representation and should not be entered as part of the command line.

Braces and brackets may be nested; the following is an example of braces nested inside brackets.

<p><i>Syntax</i></p> <p>EVT $\left. \begin{matrix} \{1\} \\ \{2\} \end{matrix} \right\} [\text{CLR}] \{ \text{option} \} \{ \text{operator} \} \{ \text{value} \} [\{ \text{option} \} \{ \text{operator} \} \{ \text{value} \}] \dots$</p>
--

Stacked Parameters

Parameters stacked within a set of braces or brackets indicate that you may select only one of the items. In the following example, either EVT1 or EVT2 parameter may be entered, but not both.

```
Syntax
EVT [1] [2] {CLR}
```

When two or more sets of braces or brackets are located on the same line, the parameters may be entered sequentially. In the following example the mode and return-option parameters (S or C) may be entered in one command line separated by a space or comma.

```
Syntax
BIF {mode} [S] [C]
```

Trailing Dots

A line of dots following a parameter indicates that the parameter may be repeated any number of times up to the maximum line length on the terminal. In the following example any one or all of the option parameters may be entered sequentially into the command line with each option separated by a space or comma. Each option consists of an option-operator-value sequence.

```
Syntax
EVT {1} [CLR] {option}{operator}{value} [ {option}{operator}{value} ] ...
```

|←— OPTION —→| |←— OPTION —→|

DESCRIPTION OF RTPA COMMANDS

RTPA Command Name Summary

The detailed descriptions of the RTPA commands are presented in the following sequence.

COMMAND	FUNCTION SUMMARY
EVT	Set or display the event comparison and counting options.
BIF	Set or clear break options.
RTT	Select type of bus transactions to be stored in RTT buffer.
DRT	Display RTT buffer contents.
CNT	Set or display count units and values.

When using the 8002A μ Processor Lab, the emulator processor must be under DEBUG control before invoking the BIF, RTT and DRT commands. Enter DEBUG prior to entering any of these RTPA commands. Make sure that TRACE OFF has been specified.

There is considerable interaction between the various RTPA commands, and between the RTPA commands and the 8001/8002A system commands (TEKOPS and TEKDOS respectively). Section 4 of this manual describes these command interactions in detail.

Relationship Between Command Names and Functions

Fig. 2-1 is a simplified block diagram representing the relationship between the RTPA command names (software control) and the functional blocks (hardware) within the RTPA. The shaded areas around the functional blocks depict basic areas of control for each RTPA command. A further breakdown of the commands will be presented in the detailed description for each command.

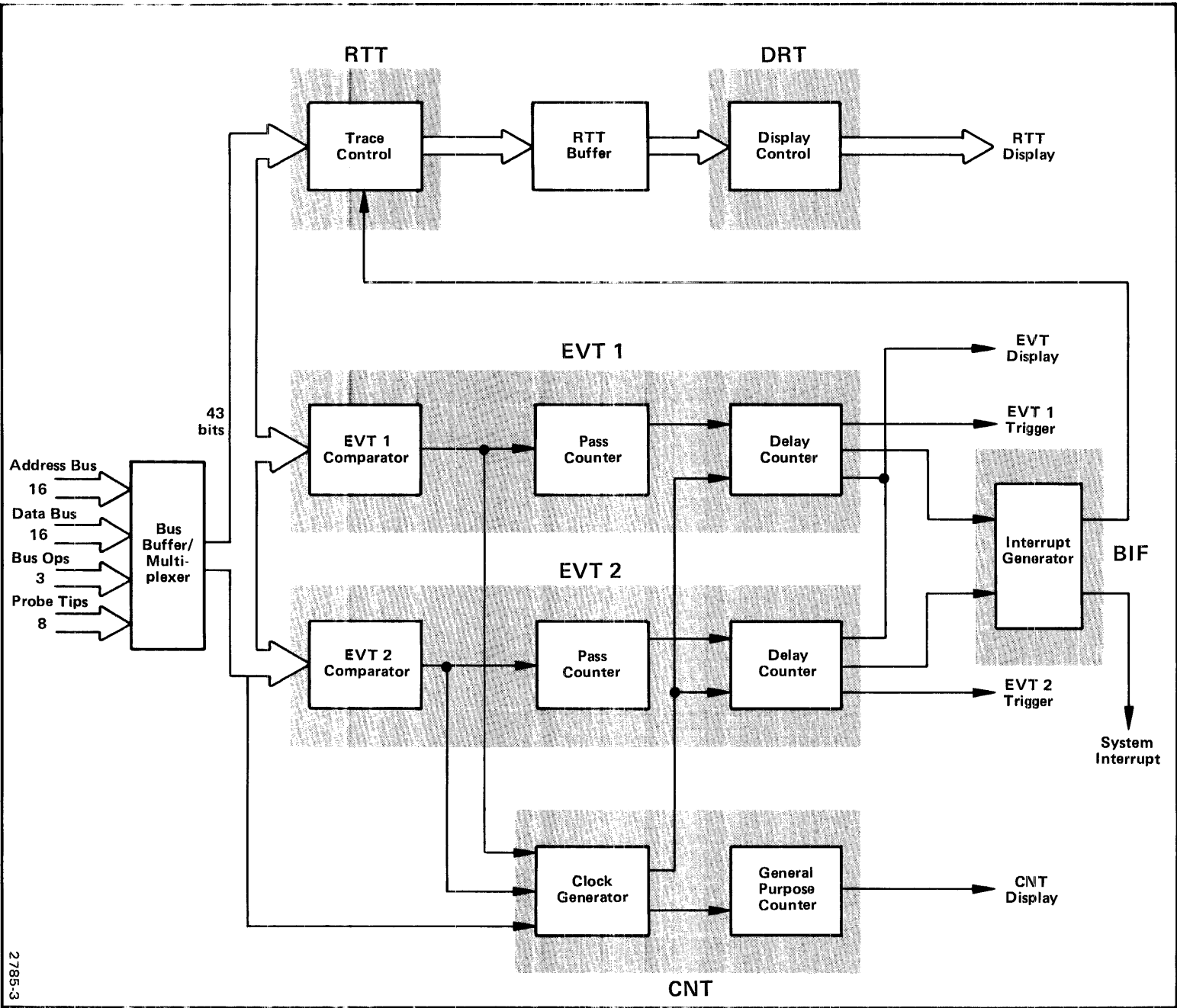


Fig. 2-1. Relationship between RTPA command names (software control) and functional blocks (hardware) within the RTPA.

EVT COMMAND

Syntax

EVT

or

EVT {mode}

or

EVT $\left[\begin{array}{l} 1 \\ 2 \end{array} \right]$ {CLR}

or

EVT $\left\{ \begin{array}{l} 1 \\ 2 \end{array} \right\}$ [CLR] {option}{operator}{value} $\left[\left[\text{option} \right] \left[\text{operator} \right] \left[\text{value} \right] \right] \dots$

Purpose

The EVT (Event) command is used to:

- enter option parameters in the EVT1 and EVT2 command lines for the purpose of:
 - creating a comparison file within the event comparators that is compared to the input data to the RTPA.
 - providing delays (pass and count delays) to the event comparator outputs.
 - generating internal and/or external event triggers.
- establish an EVT mode.
- clear the EVT1 and EVT2 command lines and EVT or BIF mode.
- display the current EVT1 and EVT2 command lines and current EVT or BIF mode.

EVT With No Parameters

Entering the EVT command with no parameters:

- displays the current status of both EVT1 and EVT2 on separate display lines.
- displays the assigned EVT or BIF mode, whichever is assigned, on the third line.

Example:

```
ENTER:      > EV
DISPLAY:    EVT1  A=0200 B=ALL
            EVT2  A=0300 B=ALL
            ARM
            >
```

EVT With Parameters

A brief explanation of the EVT option, clear, and mode parameters appears first; then, each EVT option and mode parameter is described in detail.

EVT Option Parameters

The EVT option parameters in each EVT command line may be classified into comparison options and counting options, as follows:

Comparison Options		Counting Options	
Option	Enter and Display Values	Option	Enter and Display Values
A	Address bus (8-bit or 16-bit hexadecimal).	P	Decimal value of pass counter setting.
D	Data bus (8-bit or 16-bit hexadecimal).	C	Decimal value of delay counter setting.
T	8-bit binary value of the test probe clips.		
B	Bus operations: consisting of various combinations of F (fetch), M/IO (memory input-output), R/W (read-write) or ALL (every bus transaction).		

The EVT comparison options set the address values, data values, binary state of the test clips, or type of bus operations in the EVT comparators. The EVT comparators' input data values must match the values entered in the EVT command lines before the EVT counting options are enabled. The EVT counting options are sequential: that is, option P (the pass counter setting) must be satisfied before option C (the delay counter setting) is enabled. When the comparator and counter values match, hardware triggers are generated for EVT1 or EVT2 to the external EVENT 1 & 2 TRIG OUT connectors on the Data Acquisition Interface panel (rear panel of the 8001/ 8002A). The external triggers are typically used to trigger an external logic analyzer or oscilloscope.

Fig. 2-2 is a simplified clock block diagram, and Fig. 2-3 is a flowchart of the EVT1 command. Both diagrams show the relationship between the comparison and counting options for EVT1. The relationship for EVT2 is similar.

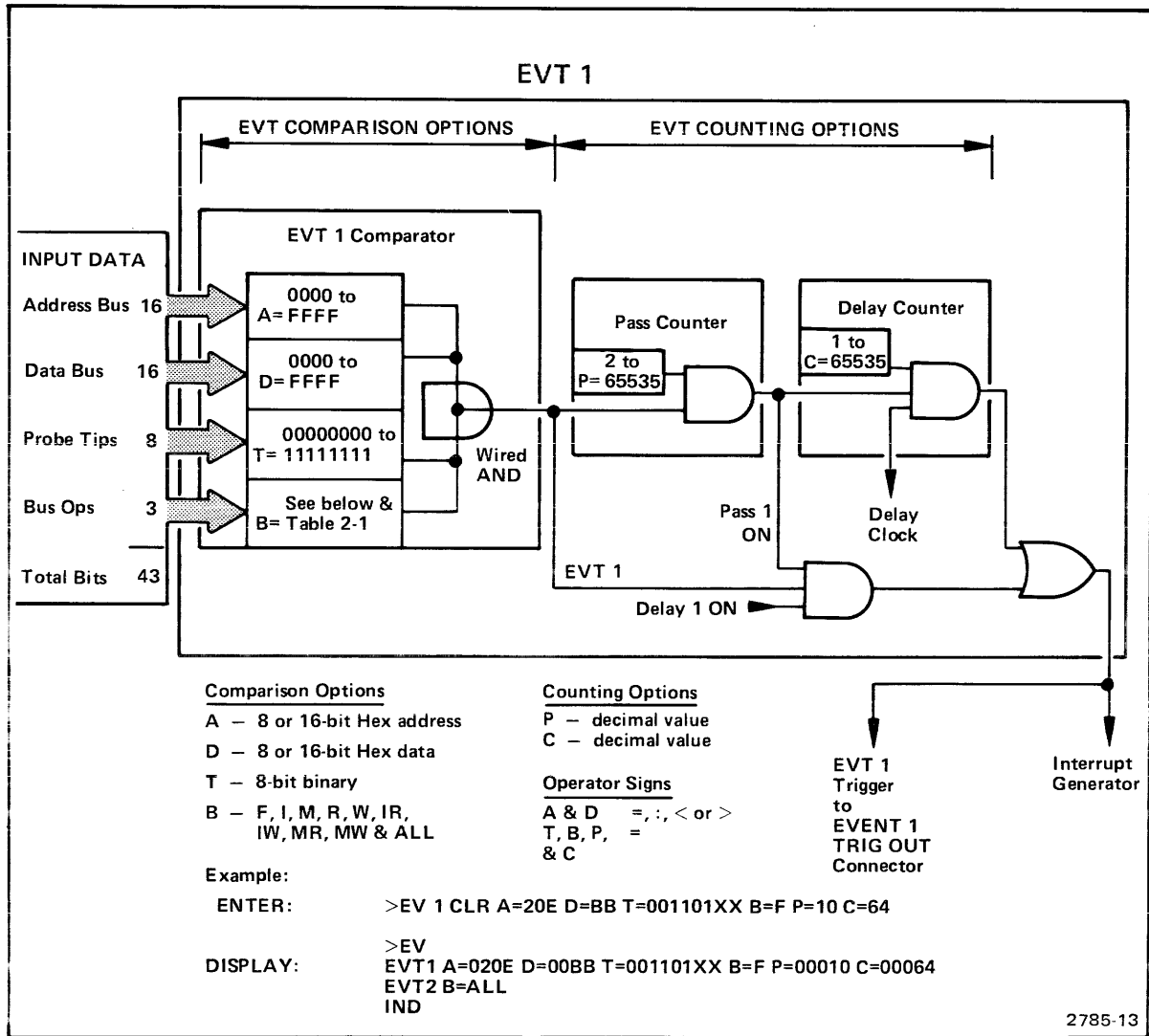


Fig. 2-2. Relationship between EVT 1 Comparison and Counting Options.

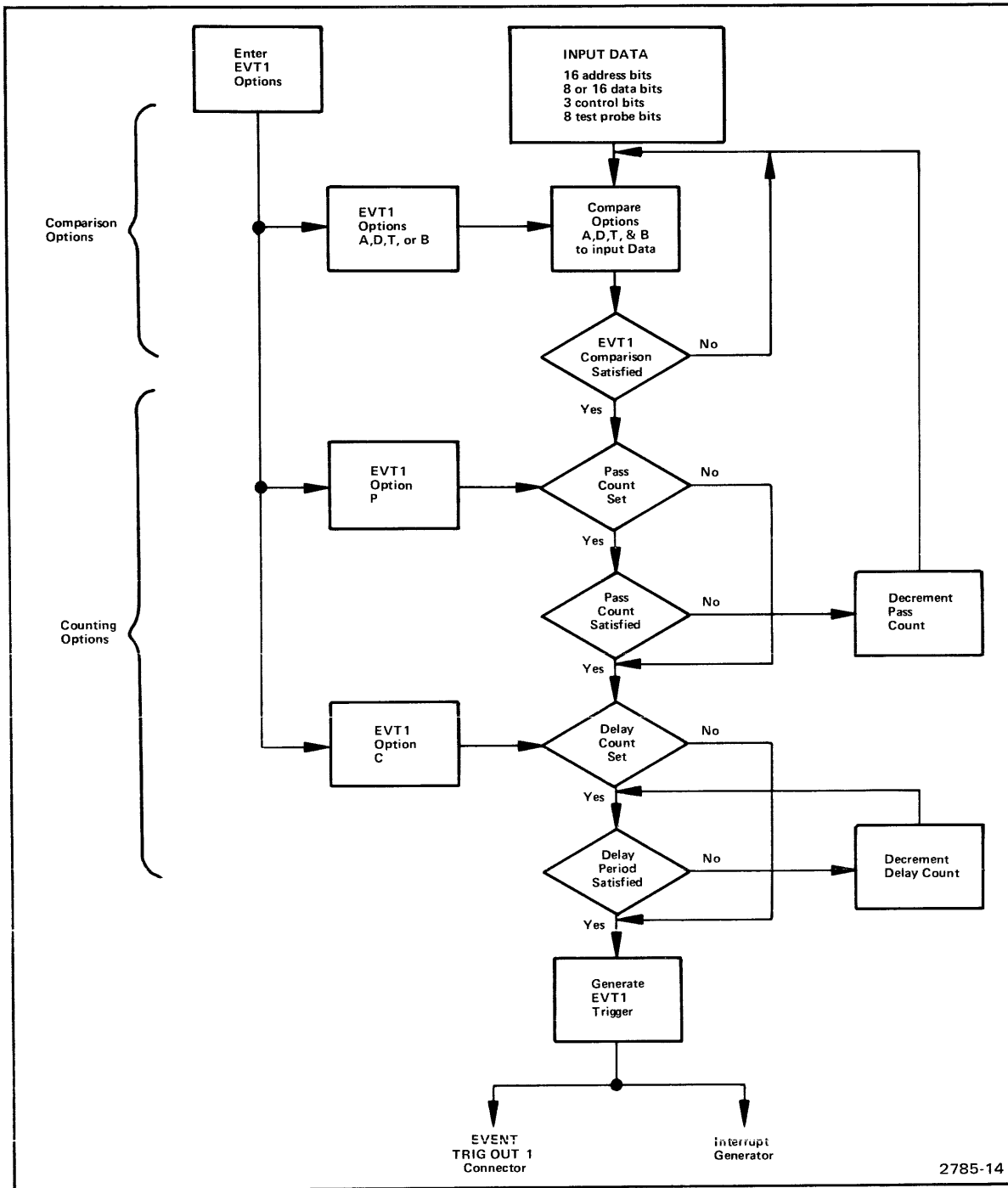


Fig. 2-3. Comparison and counting options flow chart for EVT1.

This flowchart shows the relationship between comparison options and counting options. EVT options A, D, T, and B must match the input data before the pass (option P) and/or delay (option C) counters are enabled.

EVT CLR (Clear) Parameters

The EVT CLR parameter clears all previously assigned EVT options (except option B) from both the EVT1 and EVT2 command lines. In addition, it clears the EVT or BIF mode, which then defaults to the EVT mode IND (independent). See the EVT Modes discussion later in this section.

Example:

```
ENTER:      > EV CLR

            > EV
DISPALY:    EVT1 B=ALL
            EVT2 B=ALL
            IND

            >
```

EV 1 CLR or EV 2 CLR clears only the designated EVT 1 or EVT2 command line. The EVT or BIF mode, if assigned, is not cleared. If the CLR parameter is invoked for one or both command lines, the value of EVT option B defaults to ALL.

Example:

```
ENTER:      > EV 1 CLR

            > EV
DISPLAY:    EVT1 B=ALL
            EVT2 A=0300 B=ALL
            ARM

            >
```

EVT Mode Parameters

The EVT mode parameters establish relationships between the two EVT hardware triggers that are generated when the EVT option parameters match the input data. There are three EVT mode parameters ARM, LIM, and IND. Their relationship to the EVT triggers is described as follows:

EVT MODE	FUNCTION
ARM	ARM is a sequential command: the EVT1 trigger arms or enables EVT2 comparator. The EVT2 trigger then arms EVT1 comparator.
LIM	LIM is a conditional command: both EVT1 and EVT2 comparison parameters must be satisfied at the same time before the EVT1 trigger is generated.
IND	IND is an independent command: the EVT1 and EVT2 triggers are independent of each other. EVT mode defaults to IND mode.

A detailed description of each EVT mode is presented immediately after the following explanation of the EVT options.

EVT Options

There are six option parameters that may be entered in each command line. These parameters are keyword parameters: any one or all six parameters may be entered in the EVT1 and/or EVT2 command lines, in any order. If the same option parameter is entered in a command line more than once, the most recent entry overrides any previous entry.

EVT Option List

Table 2-1 lists all EVT options and their associated operator signs and values. When an event option is assigned, an option - operator sign - value sequence must be entered in a command line, in that order. A delimiter (space or comma) is not used between the option, operator sign, and option value; however, a delimiter is required between options entered in the same command line.

Operator Signs. An operator sign is used as a special delimiter between the option and option value. Only certain operator signs may be used with each option, as listed in Table 2-1. Additional information on the use of operator signs is contained in Table 2-2.

Table 2-1
EVT Option List

Option Identifier	Operator (See Table 2-2)	Option Value	Function	
Comparison Options	A	= : < >	0000 to FFFF Hexadecimal bus address.	
	D	= : < >	0000 to FFFF Hexadecimal bus data.	
	T	=	00000000 to 11111111 Binary value of test probe clips. Use "X" to indicate "don't care" bits, for channels not connected or channels of no interest.	
	B	=		Bus Operations
			F	Instruction fetches only.
			I	I/O accesses only.
			M	Memory accesses only.
			R	Read operations only.
			W	Write operations only.
			IR	I/O Reads only.
			IW	I/O writes only.
MR			Memory reads only.	
MW	Memory writes only.			
ALL		All bus activity (default value).		
Counting Options	P	=	0 to 65535 The pass count is entered and displayed as a decimal value.	
	C	=	0 to 65535 The delay count is entered and displayed as a decimal value. The unit of delay is specified by the CNT command.	

NOTE

Refer to Appendix A for exceptions to the above. Some emulator processors do not require all the option values listed for EVT option B.

Table 2-2
EVT Option Operator Signs

Operator Signs	Used With EVT Option	Display and Function
<	A and D	The sign is entered and displayed as "<" and means less than or equal to.
>	A and D	The sign is entered and displayed as ">" and means greater than or equal to.
:	A and D	The sign is entered and displayed as ":" and is used for 8-bit equality address or data bytes. If this sign is entered for 16-bit bytes, only the 8 least significant bits are compared.
=	A and D	This sign is entered and displayed as "=" and is used for 16-bit equality address and data bytes. This sign may also be used for entering 8-bit address or data bytes. The 8 most significant bits are masked by leading zeros; only the 8 least significant bits are compared.
=	T,B,P and C	This sign is accepted as an equality sign and is displayed as "=".

EVT Options A (Address) and D (Data). EVT options A (address) and D (da) are entered and displayed in the EVT command lines in hexadecimal form. Either 8-bit or 16-bit and data values may be entered for options A or D. The operator signs ":", "=", "<," or ">" used for either 8 or 16-bit address and data values. They are entered as follows:

8-bit address and data

A:75 D:75 or A=75 D=7E

Either operator sign may be used for 8-bit equality values.

16-bit address and data

A=0375 D=307E or A>0375 D<307E

Either 16-bit equality, 16-bit less than, or 16-bit greater than operator signs may be used for 16-bit values.

16-bit address and 8-bit data

A=0375 D:7E or A>0375 D=7E or A<0375 D>7E

Less than or greater than operator signs may also be used for 8-bit data values.

The 8 or 16-bit values are compared in the EVT comparators as follows:

Operator Sign	8-Bit Values Entered	16-Bit Values Entered
: (8-bit equality)	Compares all bits entered.	Compares only the eight lower bits, or least significant bits entered.
= (16-bit equality)	Compares all bits entered; the eight upper, or most significant bits are handled as "zeros".	Compares all bits entered.
< (16-bit less than or equal)	Same as 16-bit equality.	Compares all bits entered.
> (16-bit greater than or equal)	Same as 16-bit equality.	Compares all bits entered.

EVT options A and D are displayed in the EVT command lines as 16-bit hexadecimal values. When entering 8-bit or 16-bit values, you may omit the leading zeros. When the values are displayed, the leading zeros are added.

EVT Option T. EVT option T is entered and displayed as eight binary bits. The eight binary bits represent the expected state of the Probe test clips from the P6451 Data Acquisition Probe. Logic highs are "1" and logic lows are "0". The least significant bit (rightmost bit entered) corresponds to Channel 0 and the most significant bit corresponds to Channel 7 of the P6451 probe. If all probe test clips are not used or if you are interested in looking only at specific data, then the bits corresponding to the undesired or unused probe clips should be entered as "X" (don't care). The commands to enter and display option T are shown in the following example. The probe clips entered as "X" appear in the display as "X".

Example:

```

ENTER:    > EV 1 A=68 D=21 T=XXX10111
          > EV 2 A=104 D=7A T=01001110

          > EV
DISPLAY:  EVT1 A=0068 D=0021 T=XXX10111 B=ALL
          EVT2 A=0104 D=007A T=01001110 B=ALL
          IND
          >

```

All eight bits must be entered for EVT option T. Entering eight "X's" deletes this option from the EVT command line.

EVT Option B. EVT option B permits you to monitor ten different types of bus operations. Only the bus operation specified is compared in the EVT comparators; all other bus operations are ignored. Option B defaults to "ALL" from initial power-on and remains at this value until a new value is entered. Only one bus operation option (as listed in Table 2-1) may be entered at any one time.

It is possible that the emulator processor you are using may not be able to respond to all types of bus operations. Refer to Appendix A of this manual for a listing of the characteristics of your emulator processor.

EVT Counting Options

The EVT counting options (P and C) are not enabled until the EVT comparison options match the input data. Both options P and C present a delaying action to the EVT triggers. EVT option P is a pass count delay and EVT option C is a unit delay.

EVT Option P. EVT option P is a pass count command. The pass count entered in the EVT command line is decremented by one each time the EVT comparison options are satisfied (true). When the number of true EVT comparisons equals the number entered with the EVT P command, the pass count is satisfied, this enables the unit delay counter (if EVT option C is specified) or the interrupt generator (if EVT option C is not specified). Refer to Figures 2-1 and 2-2 for details. Entering zero for the P parameter deletes this option from the EVT command lines.

EVT Option C. EVT option C is a unit delay command that is enabled by the pass count parameters (if option P is specified) or (if option P is not specified) by a true comparison of the EVT comparison options. The delay counter delays the EVT trigger by the designated number of units entered in the EVT option C parameter. The units of delay are specified with the CNT command (discussed later in this section). Entering zero for the C parameter deletes this option from the EVT command lines.

Entering and Displaying EVT Counting Options. EVT options P and C are entered and displayed as decimal values. Entering zero for either option P or C clears any previously entered value and removes the option from the display in the EVT command lines. Entering "1" for option P also clears any previously entered value and clears the option from the EVT command line display. The first true comparison from the EVT comparator decrements the pass counter from 1 to 0 and produces the same effect as if the pass counter was not invoked; therefore, the minimum value that can be displayed for EVT option P is "2" and EVT option C is "1."

The decimal value 65535 is the maximum that can be entered or displayed for EVT option P or C. This value should not be confused with the maximum general purpose counter value 65534, displayed by the CNT command. The CNT command is discussed later in this section.)

Examples of entering and displaying EVT options P and C can be found in Section 3 of this manual.

EVT Modes

As stated previously, the EVT modes establish a relationship between the two hardware triggers that are generated whenever the EVT option parameters match the input data. The EVT modes are limited in that a breakpoint is not generated or the RTT buffer is not frozen, as they are for BIF mode operation. (Breakpoints are discussed under the BIF command. The RTT buffer is discussed under the RTT and DRT commands.) In the EVT ARM mode, the general purpose counter (displayed by the CNT command) may be used for counting between the EVT1 and EVT2 triggers. The EVT hardware triggers are available at the EVENT 1 or EVENT 2 TRIG OUT connectors whenever the EVT1 and EVT2 option parameters match the input data.

Only one EVT mode parameter may be assigned at a time. The EVT mode is displayed on the third line of the EVT display, under the EVT2 command line. Table 2-3 shows the relationship between the EVT mode parameters, the entry commands, and how they are displayed.

EVT ARM

The EVT ARM command sets up a sequential relationship between EVT1 and EVT2. Refer to the flowchart in Fig. 2-4. When the EVT1 command line options match the input data, the EVT1 trigger arms or enables the EVT2 comparator. When the EVT2 options match the input data, the EVT2 trigger arms or enables the EVT1 comparator; EVT1 must be satisfied before a comparison can be made in the EVT2 comparator. The general purpose counter is enabled by the EVT1 trigger and disabled by the EVT2 trigger. Therefore, permitting the counter to count the units between the parameter settings of EVT1 and EVT2. These counting units may be changed with the CNT command (discussed later in this section).

NOTE

The general purpose counter and the EVT1 and EVT2 delay counters (EVT option C) are enabled from the same source. In the EVT ARM mode, this source is the EVT1 trigger. Refer to Fig. 2-4. When EVT ARM mode is set, the EVT1 delay counter cannot be enabled. Therefore, the EVT1 delay counter (option C) must not be specified in the EVT ARM mode.

Table 2-3
EVT Mode Parameters

EVT Mode	Interaction Between Command Lines	General Purpose Counter		RTT Buffer Frozen ^c	Breakpoint Generated ^c	Command Entry	Display
		Enabled	Disabled				
ARM	EVT1 arms EVT2. EVT2 arms EVT1.	EVT1 Trigger	EVT2 Trigger	No	No	EV ARM	ARM
IND	None	Start of program execution. ^a	End of program execution. ^b	No	No	EV IND	IND
LIM	Both EVT1 and EVT2 must be satisfied at the same time.	Start of program execution. ^a	End of program execution. ^b	No	No	EV LIM	LIM

^a The counter is reset and enabled at the start of program execution, when the GO command is entered with an address. When the GO command is entered without an address, the count in the counter is cumulative.

^b The counter is disabled at the end of program execution. There is no breakpoint set with the EVT mode commands; therefore, the end of program execution occurs under one of the following conditions:

- the ESC key is pressed.
- a software breakpoint is set with the 8001/8002A system command—BKPT
- a HALT type command is executed.

^c The RTT buffer is not frozen and a breakpoint is not generated with the EVT mode commands.

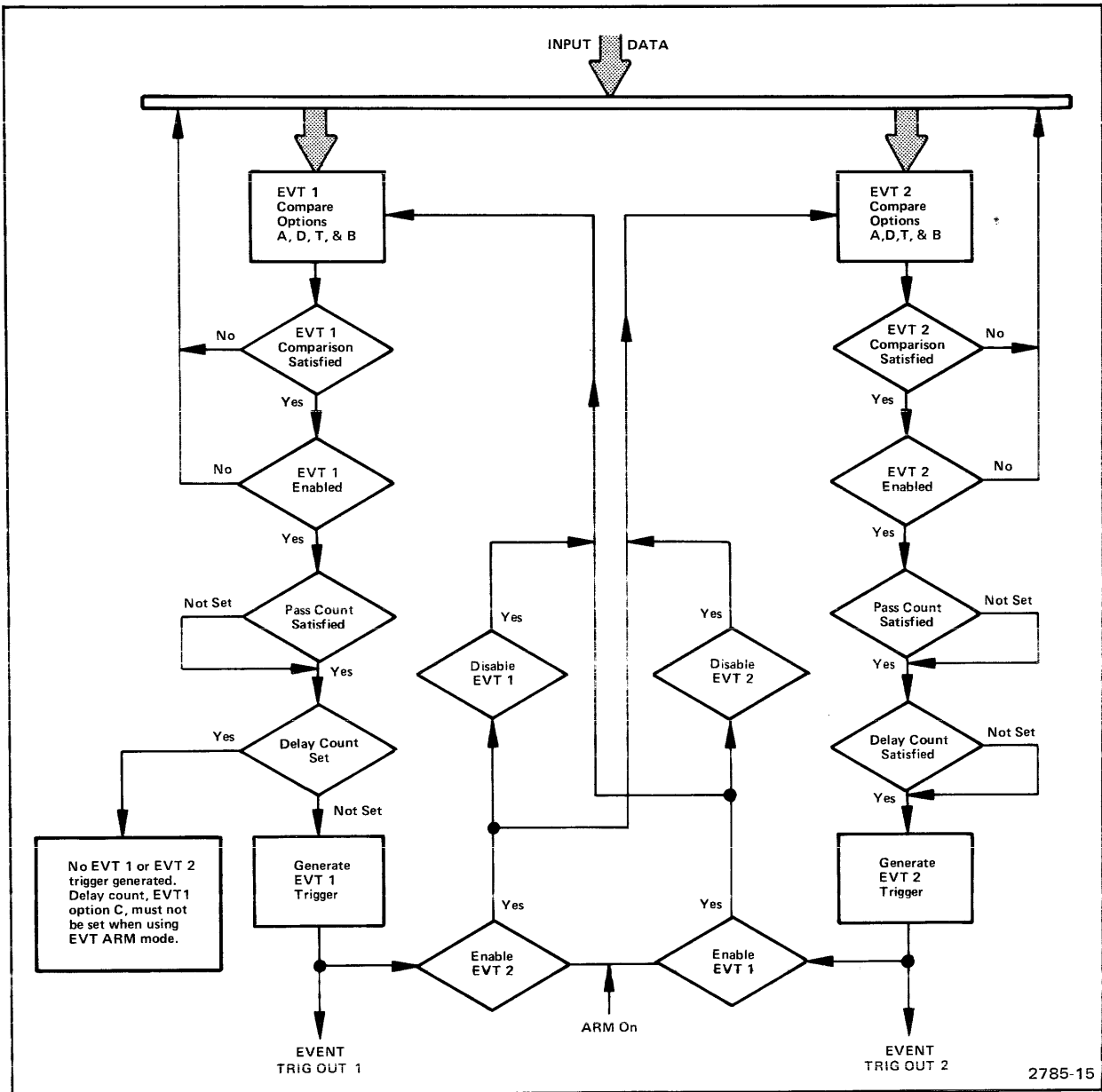


Fig. 2-4. Flow chart for EVT ARM mode.

This flowchart shows the relationship between EVT1 trigger and EVT2 trigger in the EVT ARM mode. EVT2 is enabled by EVT1. Note that the EVT1 option C (delay count) must not be set when operating in the EVT ARM mode. Either the EVT1 or EVT2 triggers may be used for external triggering.

EVT LIM (Limit)

The EVT LIM mode sets up a conditional relationship between EVT1 and EVT2. That is, all the EVT1 comparison options (A, D, T, and B) and EVT2 comparison options (A and B) must match the input data at the same time. If the addresses (option A) for both EVT1 and EVT2 are entered in the command lines with an equal sign (=) as the operator, the above conditions can never be accomplished unless EVT1 and EVT2 addresses are equal (the same address). Therefore, use operator signs < or > with either or both the EVT1 and EVT2 option A parameters. Refer to the BIF LIM mode discussion for a more detailed description on the use of the < and > operator signs.

When the EVT LIM mode is specified, only EVT2 options A and B (address and bus operations) are considered for comparison, regardless of any other comparison options entered in the EVT2 command line. See the flowchart in Fig. 2-5. Any of the EVT1 comparison options (A, D, T, and B) that are entered in the EVT1 command line are ANDed with EVT2 options A and B prior to enabling the EVT1 counting options (if specified) or generating an EVT1 trigger (if EVT1 counting options are not specified).

When using the EVT LIM mode, only the EVT1 trigger should be used for external triggering. Under certain conditions, an EVT2 trigger will also be generated; however, it will not be related to the EVT LIM mode of operation.

EVT IND (Independent)

The EVT IND mode command establishes an independent relationship between EVT1 and EVT2. The EVT1 and EVT2 triggers are independent of each other. The generation of the EVT1 or EVT2 triggers depends solely on satisfying the EVT option parameters in each command line. If the entered EVT comparison options (A, D, T, and B), and the pass and delay counts are satisfied, an EVT trigger will be generated.

The EVT mode defaults to IND (independent) mode any time that:

- EVT IND is entered.
- EVT CLR is entered.
- BIF CLR is entered. (See detailed description under BIF command.)
- 8001/8002A power is turned on.
- 8001/8002A front panel SYSTEM RESTART switch is pressed.

Either the EVT1 or EVT2 triggers may be used for external triggering.

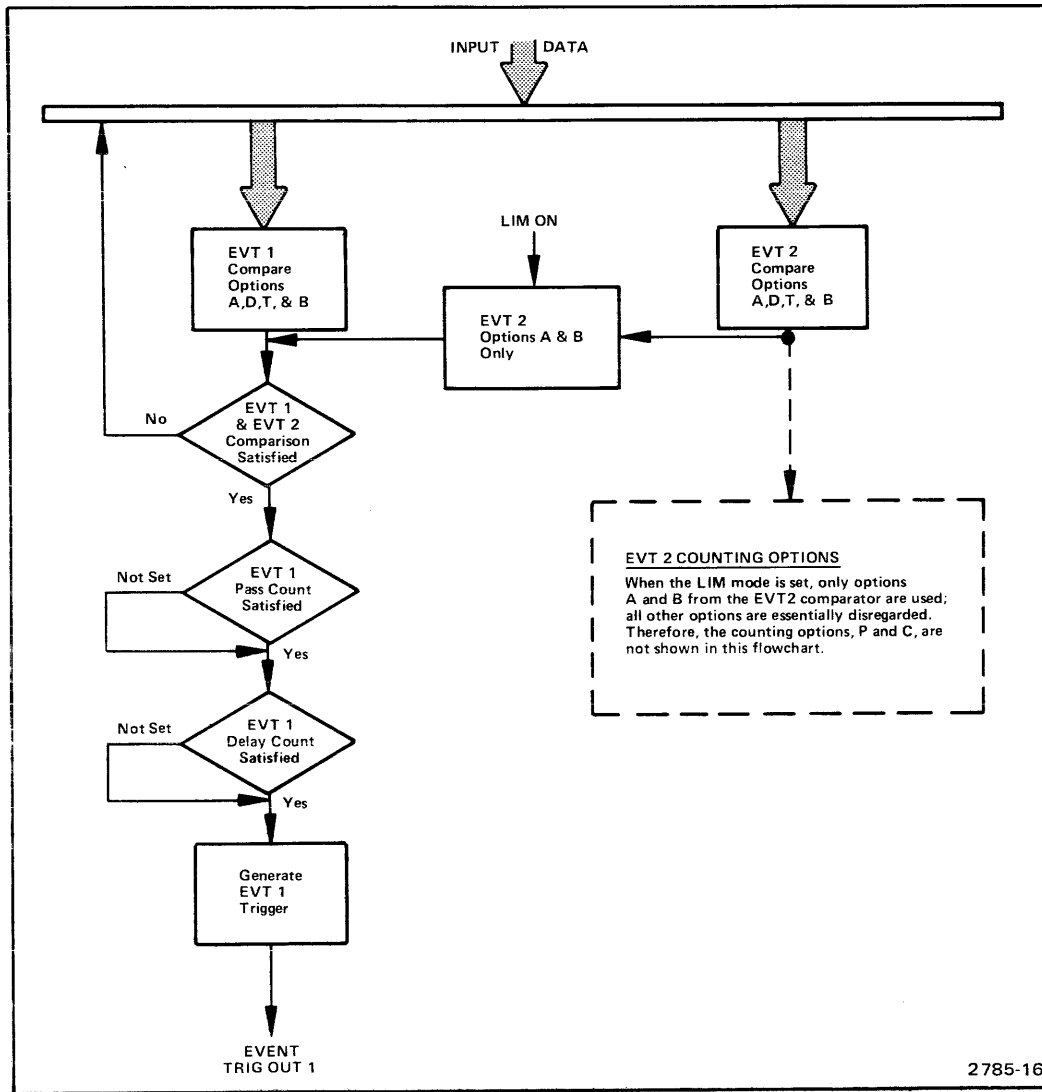


Fig. 2-5. Flow chart for EVT LIM mode.

This flowchart shows that EVT1 options A, D, T, and B together with EVT2 options A and B must be satisfied (true) at the same time before EVT1 counting options P and C are enabled. Only the EVT1 trigger should be used for external triggering.

BIF COMMAND

Syntax

BIF

or

BIF {mode} [S]
[C]

or

BIF [1]
[2] {CLR}

Purpose

The BIF (Break if) command is used to:

- Specify the BIF mode, which under software control can:
 1. cause a break in program execution.
 2. freeze the RTT buffer.
 3. reset, start and stop the general purpose program counter.
 4. return control back to the program or to the 8001/8002A system terminal.
- clear the assigned EVT or BIF mode.
- display the current BIF mode.

When you're using the 8002A, the emulator processor must be in DEBUG mode before you can invoke the BIF command.

BIF With No Parameters

The BIF command entered without parameters will display the currently assigned BIF mode. The currently assigned BIF mode will also be displayed if the EVT command is entered without parameters.

Examples:

```
ENTER:    > BIF
DISPLAY:  BIF-ARM S
          >
```

```
ENTER:    > EV
DISPLAY:  EVT1 A=0200 B=ALL
          EVT2 A=0300 B=ALL
          BIF-ARM S
          >
```

BIF With Parameters

A brief explanation of the BIF modes and clear parameters is included here; then, each BIF mode parameter is described in detail.

BIF Mode Parameters

The purpose of the BIF modes is similar to that of the EVT modes. The BIF modes, like the EVT modes, establish relationships between the two EVT hardware-generated triggers. BIF modes under software control (BIF mode assigned), break or stop program execution, freeze the RTT buffer, and return program execution control back to the 8001/8002A system terminal (for operator control) or back to the program (for continued execution).

When program execution is stopped, the breakpoint is displayed as a line of text on the system terminal. The content of the text in the break line will depend upon the type of emulator in use; however, it will always contain the program address, data, instruction mnemonic and register contents. Refer to Appendix A of this manual for a listing of the characteristics of your emulator processor.

Table 2-4 lists the relationships between the hardware triggers and software control of program execution for the six BIF mode parameters (1, 2, ARM, LIM, IND, and FRZ). These relationships may be summarized as follows: (A detailed description of each BIF mode will follow later in this section.)

BIF Mode	Function
1	When the EVT1 options are satisfied, an EVT1 trigger is generated. The breakpoint occurs at the EVT1 trigger. The EVT2 command line is ignored.
2	When the EVT2 options are satisfied, an EVT2 trigger is generated. The breakpoint occurs at the EVT2 trigger. The EVT1 command line is ignored.
ARM	This is a sequential command similar to EVT ARM, except that a breakpoint occurs at EVT2 trigger. The EVT1 trigger arms the EVT2 comparator. The EVT2 trigger then arms the EVT1 comparator.
LIM	A conditional command similar to EVT LIM, except that a breakpoint occurs at EVT1 trigger, when the EVT1 and EVT2 comparison parameters are satisfied at the same time.
IND	An independent command similar to EVT IND, except that a breakpoint occurs at either the EVT1 or EVT2 trigger.
FRZ	A sequential command similar to BIF ARM, except that the RTT buffer is frozen by the EVT1 trigger. A breakpoint occurs at EVT2 trigger.

Table 2-4
BIF Mode Parameters

BIF Mode	Interaction Between Command Lines	General Purpose Counter		RTT Buffer Frozen	Breakpoint Generated	Command Entry ^d	Display ^d
		Enabled	Disabled				
ARM	EVT1 arms EVT2. EVT 2 arms EVT1.	EVT1 Trigger	EVT2 Trigger	EVT2 Trigger	EVT2 Trigger	BIF ARM BIF ARM C	BIF-ARM S BIF-ARM C
IND	None: EVT1 and EVT2 are independent.	Start of program execution. ^a	EVT1 or EVT2 Trigger ^b	EVT1 or EVT2 Trigger ^b	EVT1 or EVT2 Trigger ^b	BIF IND BIF IND C	BIF-1 S 2 S BIF-1 C 2 C
LIM	Both EVT1 and EVT2 must be satisfied at the same time.	Start of program execution. ^a	EVT1 Trigger ^c	EVT1 Trigger ^c	EVT1 Trigger ^c	BIF LIM BIF LIM C	BIF-LIM S BIF-LIM C
FRZ	EVT1 arms EVT2. EVT2 arms EVT1.	EVT1 Trigger	EVT2 Trigger	EVT1 Trigger	EVT2 Trigger	BIF FRZ C BIF FRZ C	BIF-FRZ S BIF-FRZ C
1	None: EVT2 is ignored.	Start of program execution. ^a	EVT1 Trigger	EVT1 Trigger	EVT1 Trigger	BIF 1 BIF 1 C	BIF-1 S BIF-1 C
2	None: EVT1 is ignored.	Start of program execution. ^a	EVT2 Trigger	EVT2 Trigger	EVT2 Trigger	BIF 2 BIF 2 C	BIF-2 S BIF-2 C

^a The general purpose counter is reset and enabled at the start of program execution, when the go command is entered with an address. When the go command is entered without an address, the count on the counter is cumulative.

^b The general purpose counter is disabled, the RTT buffer is frozen, and a breakpoint is generated if either the EVT1 or EVT2 trigger is generated.

^c The general purpose counter is disabled, the RTT buffer is frozen, and a breakpoint is generated, if the EVT1 and EVT2 options are satisfied at the same time and an EVT1 trigger is generated.

^d The entry commands and displays are shown for both return-options S and C. Return-option S is the default value and is displayed if the return-option C is not specified.

Effects of BIF Modes on the General Purpose Counter

The general purpose counter permits you to count various units between events in an executing program. The units to be counted are specified with the CNT command (described later in this section). The counter will count up to a maximum of 65534. The counter is frozen when this value or a higher value is counted. The CNT command displays the maximum count as:

```
COUNT=65534 (MAX) CYCL
```

The assigned BIF mode affects the starting and stopping count of the counter. Refer to Table 2-4.

BIF CLR (Clear) Parameters

The BIF CLR parameter clears the previous assigned BIF mode and defaults to the EVT IND mode. BIF CLR also clears EVT modes ARM or LIM and defaults to the EVT IND mode. BIF 1 CLR or BIF 2 CLR only clears the designated EVT1 or EVT2 trigger of any breakpoint options.

BIF Return-Option Parameters

Two return-option parameters are available to control program execution after a breakpoint is generated. The return-option is entered into the BIF command line as S or C, after the BIF mode parameter is assigned. The S (step) return-option parameter returns program control to the 8001/8002A system terminal. The C (continue) return-option parameter returns control back to the program for continued program execution. The return-option is in effect only when a BIF mode is assigned. It is displayed to the right of the current BIF mode. Return-option S is the default value if no return-option is specified when assigning a new BIF mode. See Table 2-4 for examples of command entries and how the return-option is displayed.

BIF Modes

As previously stated, the BIF modes establish relationships between the two EVT hardware triggers. Under software control BIF modes break program execution, freeze the RTT buffer, and return program execution control to the 8001/8002A system terminal (for operator control) or back to the program (for continued execution).

A newly assigned BIF mode overrides the current BIF or EVT mode parameter. However there are several exceptions: BIF 1 does not clear BIF 2, and BIF 2 does not clear BIF 1. And, neither BIF 1 or BIF 2 will clear BIF IND. If BIF 1 has been set and BIF 2 is entered, the display will show "BIF-1 S 2 S", which is the same as BIF IND; therefore, it is recommended that you enter BIF CLR to clear the BIF mode before you enter BIF 1 or BIF 2.

The return-option (S or C) parameter must be assigned after the BIF mode but in the same BIF command line. To change BIF-ARM S to BIF-ARM C, the command "BIF ARM C" must be entered.

The BIF CLR parameter and BIF mode parameter cannot be entered in the same command line. Also, only one BIF mode may be assigned at a time.

Table 2-4 shows the relationships of the various BIF modes to the EVT hardware triggers. Refer to this table throughout the following description of the various BIF modes.

BIF 1 AND BIF 2

BIF 1 or BIF 2 is used to break program execution on either the EVT1 or EVT2 option parameters. The breakpoint depends only on the EVT comparison and counting options entered into either the EVT1 or EVT2 command line. When all EVT command line options are satisfied, an EVT trigger is generated. The EVT trigger breaks program execution, freezes the RTT buffer, and stops the general purpose counter.

The general purpose counter is reset and started at program execution, when the GO command with a starting address is entered; the counter stops at the breakpoint. The units are counted from the starting address to the breakpoint. If the GO command is entered without an address, the counter is not reset; subsequent units are added to the current value of the counter until the maximum count value is reached.

Before you enter BIF 1, BIF 2, or BIF IND, make sure you enter BIF CLR on a separate command line.

BIF ARM

The BIF ARM command establishes a sequential relationship between EVT1 and EVT2, similar to the EVT ARM command. When all EVT1 option parameters are satisfied, an EVT1 trigger is generated. The EVT1 trigger arms the EVT2 comparator and starts the general purpose counter. When the EVT2 option parameters are satisfied, an EVT2 trigger is generated. The EVT2 trigger breaks program execution, freezes the RTT buffer, and stops the general purpose counter. Refer to the BIF ARM flowchart in Fig. 2-6.

Since the general purpose counter is started by an EVT1 trigger and stopped by an EVT2 trigger, the BIF ARM command may be used to determine timing relationships within a program.

NOTE

The general purpose counter and the EVT1 and EVT2 delay counters (EVT option C) are enabled from the same source. In BIF ARM or EVT ARM mode, this source is the EVT1 trigger. Refer to Fig. 2-6. When the BIF ARM mode is set, you should not enable the EVT1 delay counter. Therefore, the EVT1 delay counter (option C) must not be specified in the BIF ARM mode or the EVT ARM mode.

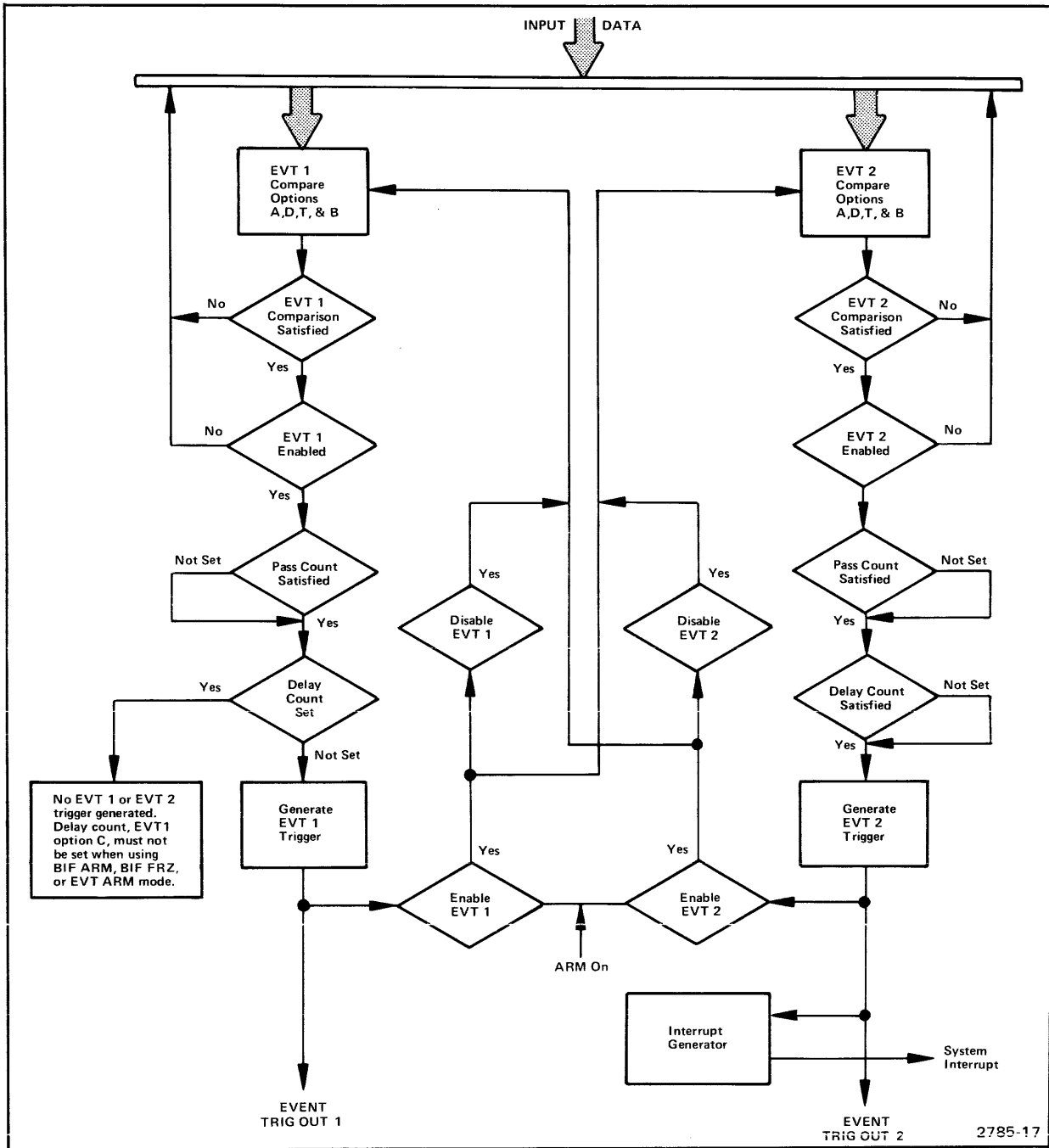


Fig. 2-6. Flow chart for BIF ARM mode.

This flowchart shows that the EVT2 comparator is enabled by the EVT1 trigger and the interrupt generator is enabled by the EVT2 trigger. Either the EVT1 or EVT2 triggers may be used for external triggering.

BIF LIM (Limit)

When BIF LIM is invoked, all the EVT1 comparison options (A, D, T, and B) and only comparison options A and B from the EVT2 command line are used for a combined EVT1 and EVT2 comparison. These six comparison options (A, D, T, and B from EVT1 and A and B from EVT2) must be satisfied (true) at the same time before the EVT1 counting options are enabled. Refer to BIF LIM flowchart in Fig. 2-7.

If the addresses (option A) for both EVT1 and EVT2 are entered in the command lines with an equal (=) operator sign, the above conditions cannot be true unless the EVT1 and EVT2 addresses are equal (the same address). This means if option A is used for both EVT1 and EVT2 command lines, either or both A options should use < or > operator signs.

As you refer to Fig. 2-7 and Table 2-4 note that the EVT1 trigger generates the breakpoint, freezes the RTT buffer, and stops the general purpose counter. Under certain conditions, the EVT2 command line options can also be satisfied (true) and an EVT2 hardware trigger generated. This EVT2 trigger is not related to the BIF LIM operation. Only the EVT1 trigger should be used for external triggering.

When assigning the < or > operator signs for EVT1 options A and D or EVT2 option A, give considerable forethought so as to avoid setting up a "no comparison" situation, thereby preventing a breakpoint from being generated. For instance, your program runs between addresses 0000 and 03FF. EVT1 option A is set to a program address near the beginning of the program. EVT2 option A is set to an address near the end of the program. The various combinations of operator signs that may be used are described in the following three examples.

In the following examples, the following assumptions apply:

Address value for EVT1 option A is 00B0.

Address value for EVT2 option A is 03F5.

BIF LIM is set.

Option B is ALL (default value).

Example 1.

Assign the < or > operator signs so that a range comparison will be made between EVT1 and EVT2, with the breakpoint at EVT2 address of 03F5. Since address 03F5 is larger than 00B0 (EVT1 address), EVT1 address must be set >00B0. The following is a display of the EVT command lines:

```
ENTER:      > EV
DISPLAY:    EVT1  A>00B0 B=ALL
            EVT2  A=03F5 B=ALL
            BIF-LIM S
```

>

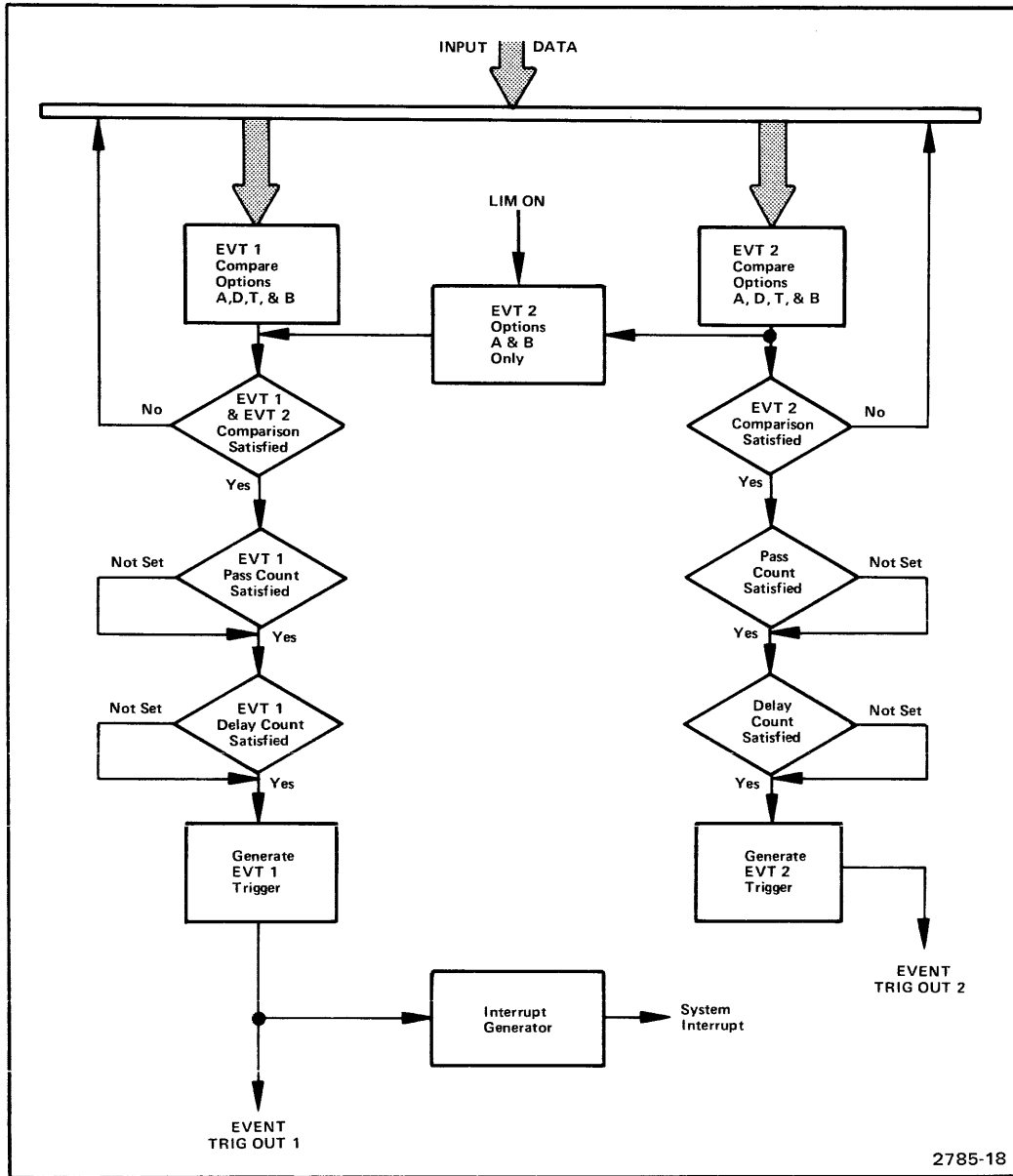


Fig. 2-7. Flow chart for BIF LIM mode.

This flowchart shows that the combined comparison of EVT1 (options A,D,T, and B) and EVT2 (options A and B) must be true before the EVT1 counting options (P and C) are enabled, if set; or EVT1 trigger generated, if options P and C are not set. The EVT1 trigger enables the interrupt generator. Only the EVT1 trigger should be used for external triggering in this mode.

When the program is executed, the EVT1 address (option A) comparison is satisfied for all addresses greater than (>) 00B0 (a match between EVT1 option A and the input data is made). When the program reaches EVT2 address (03F5), both EVT1 and EVT2 comparison options are satisfied at the same time (a match is made for both EVT parameters); triggers for both EVT1 and EVT2 are generated, and the EVT1 trigger creates a breakpoint at address 03F5.

Example 2.

Assign the < or > operator signs so that a range comparison will be made between EVT1 and EVT2 with the breakpoint at EVT1 address of 00B0. Since address 00B0 is less than 03F5, EVT2 address must be set <03F5. The following is a display of the EVT command lines:

```
ENTER:      > EV
DISPLAY:    EVT1  A=00B0 B=ALL
            EVT2  A<03F5 B=ALL
            BIF-LIM S
            >
```

When the program is executed, the EVT2 address (option A) comparison is satisfied for all addresses less than 03F5, beginning at the start of program execution (address 0000). When the program reaches EVT1 (address 00B0), both EVT1 and EVT2 command line comparisons are satisfied at the same time. Both triggers are generated and the EVT1 trigger creates a breakpoint at address 00B0.

Example 3.

Assign bus operation values to option B while maintaining the same EVT1 and EVT2 option A addresses and operators used in the preceding example. Assign EVT1 option B to compare fetch instruction cycles only (option value = F). The following is a display of the EVT command lines:

```
ENTER:      > EV 1 B=F
            > EV
DISPLAY:    EVT1  A=00B0 B=F
            EVT2  A<03F5 B=ALL
            BIF-LIM S
            >
```

When the program is executed, the same conditions are present as in example 2 until the program reaches EVT1. At address 00B0, both EVT1 and EVT2 A options are satisfied. EVT1 and EVT2 B options are also satisfied since the ALL parameter assigned to EVT2 contains all the F (fetch) instructions of EVT1. Both triggers are generated and the EVT1 trigger creates a breakpoint at address 00B0.

If EVT1 is assigned B=ALL and EVT2 is assigned B=F (the reverse of example 3), the EVT1 and EVT2 B option parameters are still satisfied, since the F parameter assigned to EVT2 is contained in the ALL parameter assigned to EVT1. However, if EVT1 were assigned B=F and EVT2 assigned B=MW, the option B parameters would not be satisfied; and no triggers or breakpoint will be generated.

The <, >, or = operator signs may also be used for EVT1 data comparison (option D). If EVT1 and EVT2 option A parameters are assigned to break on the specified EVT1 address, as shown in the display for example 2, then EVT1 option D may be set equal to (=) the actual program data value at the EVT1 address. When EVT1 option A, D, and B comparison parameters and EVT2 option A and B comparison parameters are satisfied, the EVT1 trigger will create a breakpoint at address EVT1. If EVT1 and EVT2 option A parameters are assigned to break on the specified EVT2 address, as shown in the display for example 1, then EVT1 option D may be entered as less than, greater than, or equal to the actual data in the program at EVT2 address.

In BIF LIM mode, the EVT2 compares only the A and B options and ignores the D, T, P and C options. However, when the breakpoint is set to the EVT2 address, the EVT1 comparison options must be satisfied at the EVT2 address also. The EVT1 comparison options are no longer associated with the EVT1 address, but are compared to the actual program address, data, test clips and bus operation (A, D, T, and B) values at EVT2 address.

Option T (test probe clips) may also be entered in the EVT1 command line. The operation is very similar to the D option except that equal to (=) is the only operator sign that may be assigned to option T. When the breakpoint is set for EVT1, option T must equal the actual test clip value at the EVT1 address. When the breakpoint is set for EVT2, option T must equal the actual test clip value at the EVT2 address.

BIF IND (Independent)

The BIF IND mode command establishes an independent relationship between EVT1 and EVT2 similar to the EVT IND mode command. All the parameters of EVT1 command line are independent of the parameters of EVT2 command line. If the comparison and counting options of either command line are satisfied, a trigger is generated and a breakpoint established. This breakpoint stops program execution, freezes the RTT buffer, and returns control either to the system terminal or back to the program. Any one or all of the option parameters may be entered in either command line.

The BIF IND mode is entered as "BIF IND" and displayed as "BIF-1 S 2 S". This indicates that both EVT1 and EVT2 triggers are available to generate a system interrupt (breakpoint). Since both triggers are independent, the return-option (S or C) is displayed for each EVT trigger. The return-option parameter for each trigger may be changed. For example, if you desired to let the program continue execution after the EVT1 occurs and stop execution at EVT2, the following BIF command should be entered:

Example:

```
ENTER:      > BIF IND
            > BIF 1 C
            > BIF
DISPLAY:    BIF-1 C 2 S
            >
```

BIF FRZ (Freeze)

The BIF FRZ mode command establishes a sequential relationship, similar to the BIF ARM mode command, except that the RTT buffer is frozen by the EVT1 trigger. When the EVT1 option parameters are satisfied, the EVT1 trigger arms the EVT2 comparator, starts the general purpose counter, and freezes the RTT buffer. When the EVT2 option parameters are satisfied, the EVT2 trigger breaks program execution, stops the counter, arms EVT1 comparator, enables the RTT buffer, and returns control to the system terminal or back to the program.

NOTE

When operating in the BIF FRZ mode, specifying an EVT1 delay count (option c) will make it impossible to generate an EVT1 trigger. See the note under the BIF ARM discussion.

When the program is executed, if the EVT1 option parameters are satisfied, BIF 1 generates a system interrupt. However, since the return-option C is set, the control of program execution is returned to the program. Execution continues until the EVT2 option parameters are satisfied, causing BIF 2 to generate a system interrupt. Since BIF 2 has return-option S assigned, program execution is suspended and control returned to the system terminal. The EVT2 trigger freezes the RTT buffer and stops the general purpose counter.

RTT COMMAND

Syntax

RTT [option]

Purpose

The RTT (Real-time trace) command is used to:

- select the type of bus transactions to be stored in the RTT buffer.
- display the current RTT option.

When you're using the 8002A, the emulator processor must be under DEBUG control before you invoke the RTT command.

RTT With No Parameters

When the RTT command is entered without an option parameter, the currently assigned RTT option is displayed.

RTT With Parameters

The RTT command selects or displays the type of bus transactions to be stored in the RTT buffer. A space delimiter is used between the RTT command and the option parameter. When the RTT command is entered with an option parameter, it overrides the currently assigned RTT option. Only one RTT option parameter can be specified at a time. The RTT option defaults to ALL parameter (all bus transactions are stored).

The following is a list of RTT options that may be selected. Some emulator processors do not require some of these option values; refer to Appendix A for exceptions.

RTT Option	Function
F	Instruction fetches only.
I	I/O accesses only
M	Memory accesses only.
R	Read operations only.
W	Write operations only.
IR	I/O reads only.
IW	I/O writes only.
MR	Memory reads only.
MW	Memory writes only.
ALL	All bus transactions.

RTT Options and the RTT Buffer

The RTT buffer has a storage capacity of only 128 bus transactions at any given time. Your choice of an RTT option determines what bus transactions are stored in the RTT buffer. If you are interested in analyzing only a certain type of bus transaction, your choice of an RTT option allows you to most efficiently fill the RTT buffer with that bus transaction.

RTT Storage

The transactions are stored in the RTT buffer sequentially, from the oldest to the most recent transaction. When a breakpoint is reached and the RTT buffer is frozen (except in the BIF FRZ mode), the last instruction stored is the break line. If the break line is a multi-byte instruction (occupying more than one program address), the entire break instruction will be stored in the buffer before it is frozen. This is true for any of the addresses within the instruction that is entered in the EVT option A command line.

When BIF FRZ is invoked, the buffer is frozen on the EVT1 trigger and not on the breakpoint. If EVT1 is a multi-byte instruction (occupying more than one program address), only that part of the instruction associated with EVT1 option A (address) will be stored in the buffer; any remaining bytes of the instruction will be lost (not stored in the buffer).

DRT COMMAND

Syntax

```
DRT [ *  
      number ]
```

Purpose

The DRT (Display real-time trace) command is used to display the RTT buffer contents or any portion thereof. When you're using the 8002A, the emulator processor must be in DEBUG mode before, you can invoke the DRT command.

DRT With No Parameters

When the DRT command is entered without an option parameter, the buffer contents are displayed from the start of the last executed program instruction, to where the RTT buffer is frozen. If more than 128 bus transactions have passed since the start of program execution, the entire contents of the buffer (128 transactions) are displayed.

DRT With Parameters

if an asterisk (*) is entered as the DRT parameter, the entire contents (128 bus transactions) of the RTT buffer are displayed. (A space is required between DRT and asterisk.) If a number (n) is entered as a parameter, the most recent n transactions in the RTT buffer are displayed.

DRT Display

All transactions are displayed sequentially, from the oldest to the most recent transaction stored in the RTT buffer. Blank lines separate the start of the most recent program execution from any previously executed program.

Each line in the display contains only one bus transaction, and lists the address, data, instruction mnemonic, the binary value of the eight test clips (from the P6451 probe), and the type of bus transaction. The DRT display format for the 8080A emulator processor appears as follows:

```

ENTER:      > DR 6

DISPLAY:    ADDR DATA MNEMONIC  EXTERNAL  BUS
            0379 50          00000000 M R
            037A 03          00000000 M R
            037B 36 MVI    M  00000000 M R F
            037C 00          00000000 M R
            0350 00          00000000 M W
            037D 7D MOV    A,L 00000000 M R F

            >

```

In the above example, when "DRT 6" is entered in the command line, the last six transactions stored in the RTT buffer are displayed. The headings for each column are displayed at the start of each DRT display and repeated every 22 transactions thereafter.

The display format and headings of each column differ slightly with each emulator processor.

CNT COMMAND

Syntax

CNT [option]

Purpose

The CNT (Count) command is used to:

- set the units of:
 1. count for the general purpose counter.
 2. delay for EVT option C (delay count).
- display the:
 1. current general purpose counter value.
 2. units of count and delay.

CNT With No Parameter

When the CNT command is entered without option parameters, the count value and units of count are displayed for the last executed program. The counter is reset each time a program is executed with a GO command followed by a starting address. The count is accumulative if the program is executed without a starting address. The counter will "freeze" at a count of 65534 if the count reaches or exceeds this value. The following is an example of the CNT display showing the maximum count being exceeded on the counter.

Example:

```
ENTER:      > CN
DISPLAY:    COUNT=65534 (MAX) USEC

           >
```

The CNT command entered with no option parameters will display the count value and units of count from the start of the last executed program or from EVT1 (if in the ARM or FRZ mode) to the stop of program execution; at this time the RTT buffer is frozen (except FRZ mode) and a breakpoint is established

CNT With Parameter

The CNT option parameter sets the units of count for the general purpose counter and the units of delay for EVT option C (delay count). The following units of count or delay may be selected:

CNT Option	Function
F	Bus instruction fetches. Counts each bus fetch cycle.
C	Bus cycles. Counts each transaction of the bus.
E	Emulator clocks. Counts the number of emulator clock cycles.
T	RTT buffer stores. Counts the number of transactions stored in the RTT buffer.
U	Microseconds. Counts the program running time in microseconds.
M	Milliseconds. Counts the program running time in milliseconds.
1	EVT1 occurrences. Counts how many times EVT1 option parameters are satisfied.
2	EVT2 occurrences. Counts how many times EVT2 option parameters are satisfied.

NOTE

The method of counting emulator clocks (CNT option E) is not the same for all emulator processors. Refer to Appendix A of this manual for a listing of the characteristics of your emulator processor.

The CNT option should be specified before executing a program. If a new CNT option is entered, the program must be executed again to obtain a valid count. Only one CNT option can be specified at a time. The default option for the CNT command is M (milliseconds).

Resetting the Counter

The counter is reset when one of the following conditions occurs:

1. Entering a new CNT command.
2. Executing the program with a starting address.
3. Changing the BIF or EVT mode.
4. Changing any EVT option parameter.

Section 3

HOW TO USE THE RTPA COMMANDS

	Page
Introduction	3-1
Assumptions	3-1
Limitations of Some Emulator Processors	3-1
Equipment Required	3-2
Emulation Mode	3-2
Procedures	3-2
Loading a Sample Program	3-3
Procedure 1—EVT Comparison Options With BIF 1 or BIF 2 Mode	3-7
Procedure 2—EVT Counting Options With BIF 1 or BIF 2 Mode	3-16
Procedure 3—EVT Comparison and Counting Options With BIF ARM Mode	3-27
Procedure 4—EVT Comparison and Counting Options With BIF IND Mode	3-35
Procedure 5—EVT Comparison and Counting Options With BIF FRZ Mode	3-36
Procedure 6—EVT Comparison and Counting Options With BIF LIM Mode	3-42
Procedure 7—Pre, Center, and Post-Triggering	3-50
Pre-Triggering	3-50
Center Triggering	3-51
Post-Triggering	3-52

ILLUSTRATIONS

Fig. No.		Page
3-1	Program memory location of sample 8080A program	3-4
3-2	Source file of sample 8080A program	3-5
3-3	List file of sample 8080A program	3-6

Section 3

HOW TO USE THE RTPA COMMANDS

INTRODUCTION

This section contains exercises that show you how to enter the various RTPA commands, how to display them on the system terminal, and how to interpret the results. Additional comments are provided within the text to explain environment conditions that are not obvious at first glance.

ASSUMPTIONS

These examples assume that you have read Sections 1 and 2 of this manual and that you are familiar with the following RTPA concepts.

- How the RTPA compares the input data to the EVT command line parameters to obtain the event triggers.
- How the EVT and BIF modes establish a relationship between the EVT triggers.
- How the RTT buffer stores the input data and how this data can be displayed on the system terminal.

It is also assumed that you are familiar with the basic commands of the 8001/8002A and that you have developed some skills in their use and execution.

LIMITATIONS OF SOME EMULATOR PROCESSORS

The RTPA commands may be used with all emulator processors supported by the 8001/8002A. However, some emulator processors do not require (or have a need for) some of the commands. For instance, some of the emulator processors have no requirement for all the bus operations listed for the EVT option B and RTT commands. Appendix A lists these differences for each emulator processor supported by the 8001/8002A.

EQUIPMENT REQUIRED

The following equipment is used in these procedures to demonstrate the functions and capabilities of the RTPA. For demonstration purposes, the 8080A emulator processor was chosen to be used throughout this manual. If you are using another emulator processor, the RTPA commands are entered and displayed in the same manner; however, the sample program included herein cannot be used.

- 8001/8002A μ Processor Lab with:
 - Real-Time Prototype Analyzer
 - 8080A Emulator Processor
- System Terminal—CT 8100, 4024/4025 CRT Terminal or CT 8101 Printing Terminal (only one terminal required)

EMULATION MODE

All of the RTPA commands may be used in any of the three emulation modes. To avoid the confusion of changing modes while learning to use the RTPA commands, emulation mode 0 is used throughout this section for all demonstrations and examples. Differences between the emulation modes are discussed in Section 5.

The use of the EVT option T command is not covered in this section. The EVT option T command requires that the probe test clips of the P6451 probe be connected to an 8-bit data source.

In emulation Mode 0, the P6451 probe is not generally used. The EVT option T command is discussed in detail in Section 5.

PROCEDURES

The following seven procedures are discussed in this section:

1. EVT comparison options with BIF 1 or BIF 2 mode set.
2. EVT counting options with BIF 1 or BIF 2 mode set.
3. EVT comparison and counting options with BIF ARM mode set.
4. EVT comparison and counting options with BIF IND mode set.
5. EVT comparison and counting options with BIF FRZ mode set.
6. EVT comparison and counting options with BIF LiM mode set.
7. Pre-, center, and post-triggering.

Each procedure contains an outline of the procedure goals, and provides step-by-step examples for achieving these goals. Each example, contains an explanation or comment note with a complete description for that step.

The experienced user will be able to follow the general description and perform most of the tasks outlined; however, a user with less experience may be required to follow the detailed step-by-step examples associated with each procedure to achieve the procedure goals.

Most procedures are based on information from previous procedures. Only the necessary command entries are entered for each example. When using an example in the middle of a procedure, review the previous examples to ensure that all parameters are entered in the command lines.

Each example has a notation along the left margin. The notation indicates whether it is an ENTRY command, a DISPLAY, or a function key is activated (PRESS ESC KEY). Each example shows prompt characters (>) as they appear on your system terminal.

Examples are referenced within this section by a three digit number: for example, 4.B.5. The first digit refers to the procedure number, the second to the major subjects under the procedure, and the third digit refers to the examples under the major subject.

NOTE

The examples used in this section are not intended to have a practical or useful application, but are used to demonstrate the capabilities of each command or combination of commands. In some instances there may be several solutions that will produce the same results.

Loading A Sample Program

The procedures in this section are based on a sample program that is loaded into the 8001/8002A program memory.

This sample program is shown in Figs. 3-1, 3-2 and 3-3:

- Fig. 3-1 shows the data bytes for each program memory address, beginning at address 0200. This program may be entered into the program memory from the keyboard of your system terminal with the PATCH or EXAM commands.
- Fig. 3-2 shows the source file of the sample program. This source file can be entered into the 8002A with the EDIT commands. It can then be assembled into an object file, which can be loaded into the program memory with the LOAD command.
- Fig. 3-3 shows the list file that results from assembling the sample program source file shown in Fig. 3-2. This list file is used throughout the following procedures as a guide in setting the various RTPA command parameters. Fig. 3-3 is reproduced in Appendix D for ready reference when following the examples in this section.

NOTE

The program memory in the 8001 and 8002A can be loaded either from a paper tape reader or external computer. See the 8001 μ Processor Lab System User's Manual or the 8002A μ Processor Lab System User's Manual for additional information.

```
0200=F3 11 75 02 21 50 02 36 00 7D BB C2 11 02 C3 75
0210=03 23 C3 07 02 FF FF FF FF FF FF FF FF FF FF
0300=00 76 FF FF FF FF FF FF FF FF FF FF FF FF FF
0370=FF FF FF FF FF 01 25 03 21 50 03 36 00 7D B9 C2
0380=85 03 C3 00 03 2B C3 7B 03 FF FF FF FF FF FF FF
```

2785-19

Fig. 3-1. Program memory location of sample 8080 program.

```

;*****
;
;
;          SAMPLE PROGRAM FOR 8080A EMULATOR
;          USED FOR RTPA DEMONSTRATIONS
;
;
; THIS PROGRAM STARTS AT 0200H; IT LOADS MEMORY LOCATIONS
; 0250H TO 0275H WITH ZEROS, THEN JUMPS TO 0375H AND
; LOADS MEMORY LOCATIONS 0375H TO 0350H WITH ZEROS; IT
; THEN JUMPS TO 0300H AND STOPS AT 0301H.
;
;*****
START      ORG      0200H
           DI
           LXI     D,0275H ;LOAD MEMORY CHECK
           LXI     H,0250H ;GET START OF MEMORY
LOOP1      MVI     M,0     ;CLEAR MEMORY LOC 0250H
           MOV     A,L     ;CHECK LOW BYTE
           CMP     E       ;IS IT 75?
           JNZ     CONT1   ;NO? - CONTINUE
           JMP     NEXT    ;IT IS? JUMP TO MEM LOC 0375H
CONT1      INX     H       ;INR MEMORY REGISTER
           JMP     LOOP1   ;KEEP ZEROING

           ORG      0375H
NEXT       LXI     B,0325H ;LOAD MEMORY CHECK
           LXI     H,0350H ;GET START OF MEMORY
LOOP2      MVI     M,0     ;CLEAR MEMORY LOC 0350H
           MOV     A,L     ;CHECK LOW BYTE
           CMP     C       ;IS IT 25?
           JNZ     CONT2   ;NO? - CONTINUE
           JMP     STOP    ;IT IS? JUMP TO MEM LOC 0300H
CONT2      DCX     H       ;DCR MEMORY REGISTER
           JMP     LOOP2   ;KEEP ZEROING

           ORG      0300H
STOP       NOP          ;LAST RTPA BKPT BEFORE HALT
           HLT          ;SUSPENDS EXEC AWAITING INT
           END      START

```

2785-20

Fig. 3-2. Source file of sample 8080 program.

```

00001 ;*****
00002 ;*
00003 ;*
00004 ;*          SAMPLE PROGRAM FOR 8080A EMULATOR          *
00005 ;*          USED FOR RTPA DEMONSTRATIONS                *
00006 ;*
00007 ;* THIS PROGRAM STARTS AT 0200H; IT LOADS MEMORY LOCATIONS *
00008 ;* 0250H TO 0275H WITH ZEROS, THEN JUMPS TO 0375H AND   *
00009 ;* LOADS MEMORY LOCATIONS 0375H TO 0350H WITH ZEROS; IT *
00010 ;* THEN JUMPS TO 0300H AND STOPS AT 0301H.              *
00011 ;*
00012 ;*****
00013 ;
00014      0200  >      ORG      0200H
00015 0200 F3      START      DI
00016 0201 117502   LXI      D,0275H ;LOAD MEMORY CHECK
00017 0204 215002   LXI      H,0250H ;GET START OF MEMORY
00018 0207 3600     LOOP1     MVI      M,0 ;CLEAR MEMORY LOC 0250H
00019 0209 7D       MOV      A,L ;CHECK LOW BYTE
00020 020A BB       CMP      E ;IS IT 75?
00021 020B C21102  >      JNZ     CONT1 ;NO? - CONTINUE
00022 020E C37503  >      JMP     NEXT ;IT IS? JUMP TO MEM LOC 0375H
00023 0211 23       CONT1     INX     H ;INR MEMORY REGISTER
00024 0212 C30702  >      JMP     LOOP1 ;KEEP ZEROING
00025
00026      0375  >      ORG      0375H
00027 0375 012503   NEXT     LXI      B,0325H ;LOAD MEMORY CHECK
00028 0378 215003   LXI      H,0350H ;GET START OF MEMORY
00029 037B 3600     LOOP2     MVI      M,0 ;CLEAR MEMORY LOC 0350H
00030 037D 7D       MOV      A,L ;CHECK LOW BYTE
00031 037E B9       CMP      C ;IS IT 25?
00032 037F C28503  >      JNZ     CONT2 ;NO? - CONTINUE
00033 0382 C30003  >      JMP     STOP ;IT IS? JUMP TO MEM LOC 0300H
00034 0385 2B       CONT2     DCX     H ;DCR MEMORY REGISTER
00035 0386 C37B03  >      JMP     LOOP2 ;KEEP ZEROING
00036
00037      0300  >      ORG      0300H
00038 0300 00       STOP     NOP ;LAST RTPA BKPT BEFORE HALT
00039 0301 76       HLT ;SUSPENDS EXEC AWAITING INT
00040      0200  >      END     START

```

2785-21

Fig. 3-3. List file of sample 8080 program.

When the sample program (shown in Figs. 3-1, 3-2 and 3-3), is executed, it fills the memory locations 0250 to 0275 and 0325 to 0350 with zeros. First before entering or loading the program into program memory, fill all the memory locations between 0200 and 03FF with "FF". Then, when your program executes you can more readily see the zeros.

When you have completed loading the sample program into program memory, you are ready to start the following procedures. If you are using an 8002A, it must be under Debug control before the RTPA commands BIF, RTT, and DRT can be invoked. To place the 8002A under Debug control, enter DEBUG. Make sure the Debug Trace is off when using either the 8001 or 8002A: enter the command line TRACE OFF.

Load the sample program as follows:

```
ENTER:      > F 200 400 FF
DISPLAY:    *FILL* EOJ

ENTER:      > LO RTPA4
DISPLAY:    TRANSFER ADDRESS: 0200
            *LOAD* EOJ

ENTER:      > DEB

            >
```

NOTE

For Demonstration purposes the sample program is assigned an object file name RTPA4.

Procedure 1—EVT Comparison Options With BIF 1 or BIF 2 Mode

This procedure demonstrates how to enter and display the EVT1 and EVT2 comparison options (options A, D, and B) using the BIF 1 or BIF 2 mode. It also shows the display capabilities and the timing relationships with the DRT, RTT, and CNT commands.

Exercises

- 1.A. Set the BIF 1 mode breakpoint to break on EVT1 options A and D.
 - 1.A.1. Enter the EVT1 options A and D from an address near the start of the sample program. (Address 0201 is used in this example. Refer to Fig. 3-3.)

Example:

```
ENTER      > EV 1 CLR A=201 D=11

            >
```

1.A.2. Set the BIF mode to break on EVT1.

Example:

```

ENTER:      > BIF CLR
            > BIF 1
            >

```

1.A.3. Display the settings of EVT options and BIF mode.

Example:

```

ENTER:      > EV
DISPLAY:    EVT1 A=0201 D=0011 B=ALL
            EVT2 B=ALL
            BIF-1 S
            >

```

Explanation:

The EVT command (without parameters) displays the status of the EVT1 and EVT2 command lines and the BIF or EVT mode. You can use this command frequently, to obtain the current status of the EVT command lines. In the following procedures and examples, the use of the EVT display is used sparingly to save space.

1.A.4. Execute the sample program to verify that program execution is broken at the specified EVT1 option settings.

Example:

```

ENTER:      > G 200

DISPLAY:    LOC  INST  MNEM  OPER      SP   RF  RA  RB  RC  RD  RE  RH  RL
            0201 117502 LXI   D,0275   0000 02  00 00 00 02 75 00 00
            0201 BREAK
            >

```

- 1.A.5. Display the last five storage transactions in the RTT buffer to verify that the program execution stopped at the EVT1 option settings.

Example:

```

ENTER:      > DR 5

DISPLAY:    ADDR DATA MNEMONIC  EXTERNAL  BUS
            FBA8 00  NOP          00000000 M R F
            0200 F3  DI           00000000 M R F
            0201 11  LXI   D      00000000 M R F
            0202 75          00000000 M R
            0203 02          00000000 M R

            >
    
```

} Blank space indicates start of new program.

Explanation:

In example 1.A.4, the break line appears at address 0201. The LXI instruction is a three byte instruction that occupies the addresses 0201—0203. EVT1 was set to break on address 0201, the first byte in this instruction. The RTT buffer display (example 1.A.5) shows that address 0203 was the last address stored in the buffer before program execution was suspended by the EVT1 trigger. When the breakpoint is set to any address of a multibyte instruction, the entire instruction and related bus transactions (stack storage, memory accesses, etc.) are stored in the buffer before the buffer is frozen and emulation stopped. (Note: BIF FRZ mode is an exception. See Procedure 5.)

- 1.B. Set the BIF 1 mode breakpoint to break on EVT1 options A, D, and B.

- 1.B.1. Enter the EVT1 option B to provide a breakpoint on a fetch instruction at address 0201.

Comment:

Refer back to example 1.A.5. The bus operations at address 0201 are M (memory access only), R (read operations only), and F (instruction fetches only). If option B is set to either F, M, R, MR (memory read only), or ALL, a comparison of the EVT option settings will be satisfied and a breakpoint established.

Example:

```

ENTER:      > EV 1 B=F

            >
    
```

Comment:

Since the EVT1 options A and D were entered in example 1.A.1, only option B has to be entered.

1.B.2. Display the settings of the EVT options and BIF mode.**Example:**

```

ENTER:      > EV
DISPLAY:    EVT1  A=0201 D=0011 B=F
            EVT2  B=ALL
            BIF-1 S

```

>

Comment:

Since BIF 1 was set in example 1.A.2, it remains in effect.

1.B.3. Execute the sample program to verify that program execution is broken at the specific EVT1 option settings.**Example:**

```

ENTER:      > G 200
DISPLAY:    0201 117502 LXI  D,0275  0000 02 00 00 00 02 75 00 00
            0201 BREAK

```

>

1.B.4. Change the EVT1 option B to B=W (write operations only).**Example:**

```

ENTER:      > EV 1 B=W

```

>

Results:

When the EVT1 option B is changed to W, a comparison of the EVT option settings will not be satisfied; therefore, no breakpoint will be present at address 0201. The program will run until it reaches the halt instruction at address 0301.

1.B.5. Execute the program to verify that a breakpoint does not occur at address 0201.**Example:**

```

ENTER:      > G 200
DISPLAY:    0301 76  HLT  0000 56 25 03 25 02 75 03 25
            0301 BREAK

```

>

1.B.6. Return the EVT1 option B back to B=ALL (all bus activity).

Example:

```
ENTER:      > EV 1 B=ALL
           >
```

1.C. Set the BIF 2 mode breakpoint to break on EVT2 options A and D.

1.C.1. Enter the EVT2 options A and D from the ending address of the sample program.

Comment:

Note the NOP (no operation) instruction in the sample program at address 0300 is before the halt instruction at address 0301. This address (0300) is considered to be the end of the program for the setting of a RTPA breakpoint.

Example:

```
ENTER:      > EV 2 CLRA=300 D=00
           >
```

1.C.2. Set the BIF mode to break on EVT2.

Example:

```
ENTER:      > BIF CLR
           > BIF 2
           >
```

Comment:

Always enter BIF CLR before entering BIF 1 or BIF 2. All the BIF modes when entered, except BIF 1 and BIF 2, will clear the previous BIF mode. BIF 1 or BIF 2 when entered will also clear the previous BIF mode unless that mode is BIF IND, BIF 1 or BIF 2. That is, BIF 1 will not clear BIF 2 and BIF 2 will not clear BIF 1. When either is entered without the CLR command, the display will read "BIF-1 S 2 S," which is also the equivalent of BIF IND.

1.C.3. Display the EVT options and BIF mode settings.

Example:

```

ENTER:      > EV
DISPLAY:    EVT1 A=0201 D=0011 B=ALL
            EVT2 A=0300 D=0000 B=ALL
            BIF-2 S
            >

```

1.C.4. Execute the sample program to verify that program execution is broken at the specified EVT2 option settings.

Example:

```

ENTER:      > G 200
DISPLAY:    0300 00      NOP                0000 56 25 03 25 02 75 03 25
            0300 BREAK
            >

```

1.C.5. Display the last five transactions in the RTT buffer to verify that execution was suspended on the EVT2 option settings.

Example:

```

ENTER:      > DR 5

DISPLAY:    ADDR DATA MNEMONIC  EXTERNAL  BUS
            0381 03                00000000 M R
            0382 C3  JMP           00000000 M R F
            0383 00                00000000 M R
            0384 03                00000000 M R
            0300 00  NOP           00000000 M R F
            >

```

1.C.6. Count the number of bus transactions and measure the program run time.

Example:

```

ENTER:      > CN C
            > CN
DISPLAY:    COUNT=00000 CYCL
            >

```

(Example continued on next page)

Execute the program again.

```
ENTER:    > G 200
DISPLAY:  0300 00      NOP          0000 56 25 03 25 02 75 03 25
          0300 BREAK
          >
```

Display the number of bus cycles in the program.

```
ENTER:    > CN
DISPLAY:  COUNT=00996 CYCL
          >
```

Change the CNT units to milliseconds.

```
ENTER:    > CN M
          >
```

Execute the program again.

```
ENTER:    > G 200
DISPLAY   0300 00      NOP          0000 56 25 03 25 02 75 03 25
          0300 BREAK
          >
```

Display the number of milliseconds run time.

```
ENTER:    > CN
DISPLAY:  COUNT=00001 MSEC
          >
```

Change the CNT units to microseconds.

```
ENTER:    > CN U
          >
```

(Example continued on next page)

Execute the program again.

```
ENTER:      > G 200
DISPLAY:    0300 00      NOP          0000 56 25 03 25 02 75 03 25
            0300 BREAK
            >
```

Display the number of microseconds run time.

```
ENTER      > CN
DISPLAY:   COUNT=01546 USEC
            >
```

Explanation:

Experiment with the other CNT units. Remember to execute the program each time the CNT units are changed. When BIF 1 or BIF 2 is set, the counter starts counting at the start of program execution and stops when the corresponding EVT trigger is generated.

- 1.D. Set the BIF mode 2 breakpoint to break on EVT2 options A, D, and B.
- 1.D.1. Enter the EVT2 option B to provide a breakpoint on the memory read instruction at address 0300.

Comment:

Example 1.C.5 shows that the bus transactions at address 0300 are M, R, and F. If option B is set to MR (memory read only), a comparison of the EVT option settings will be satisfied and a breakpoint established.

Example:

```
ENTER:     > EV 2 B=MR
            >
```

- 1.D.2. Execute the program to verify that emulation is broken at the specified EVT2 option settings.

Example:

```
ENTER:      > G 200
DISPLAY:    0300 00      NOP                0000 56 25 03 25 02 75 03 25
           0300 BREAK
           >
```

- 1.D.3. Set the EVT2 option B to B=F and set the RTT command to store only the fetch bus transactions. Display the last five transactions in the RTT buffer.

Example:

```
ENTER:      > EV 2 B=F
           > RT F
           > G 200
DISPLAY:    0300 00      NOP                0000 56 25 03 25 02 75 03 25
           0300 BREAK
ENTER:      > DR 5
DISPLAY:    ADDR DATA MNEMONIC  EXTERNAL  BUS
           037D 7D  MOV   A,L  00000000  M R F
           037E B9  CMP   C    00000000  M R F
           037F C2  JNZ           00000000  M R F
           0382 C3  JMP           00000000  M R F
           0300 00  NOP           00000000  M R F
           >
```

NOTE

Return the RTT command to the store ALL option by entering "RT ALL."

Procedure 2—EVT Counting Options With BIF or BIF 2 Mode

This procedure demonstrates how to enter and display the EVT1 and EVT2 counting options (options P and C) using the BIF 1 or BIF 2 mode. It also shows the display capabilities and the timing relationships of the CNT command.

Exercises

- 2.A. Set the BIF 1 mode breakpoint to break on an EVT1 comparison of options A, D and B, plus EVT1 counting option P.

Comment:

When option P is entered in either EVT command line, the EVT comparison options (A, D, B, and T) must be satisfied (true) at least the number of times entered in the option P parameter.

- 2.A.1. Clear both EVT command lines. Enter the EVT1 options A, D, and B parameters for an address in LOOP1 of the sample program (between addresses 0207 to 0214; refer to Fig. 3-3). Set BIF to break on EVT1. Then, display the EVT options and BIF mode.

Example:

```

ENTER:      > EV CLR
            > EV 1 A=209 D=7D B=F
            > BIF CLR
            > BIF 1
            > EV
DISPLAY:    EVT1 A=0209 D=007D B=F
            EVT2 B=ALL
            BIF-1 S
            >
  
```

- 2.A.2. Determine the number of bus cycles and microseconds between the first and second occurrence of a true EVT1 comparison, using the option parameters set in example

Explanation:

The first occurrence of a true event comparison satisfies a pass count of 0 or 1. The second occurrence is equivalent to a pass count of 2. Entering P=0 or P=1 clears any previous option P entry, but is not displayed in the EVT command lines. The number 2 is the lowest parameter that may be entered or displayed for option P.

2.A.3. Change the CNT command to bus cycles (CNT C). Execute the program to determine how many bus cycles there are to the first true EVT1 comparison at address 0209 and how many microseconds.

Example:

```
ENTER:      > CN C

                > G 200
DISPLAY:     0209 7D      MOV  A,L      0000 12 50 03 25 02 75 02 50
                0209 BREAK

ENTER:      > CN
DISPLAY:     COUNT=00011 CYCL

ENTER:      > CN U

                > G 200
DISPLAY:     0209 7D      MOV  A,L      0000 12 50 03 25 02 75 02 50
                0209 BREAK

ENTER:      > CN
DISPLAY:     COUNT=00017 USEC

                >
```


2.A.4. Enter a pass count of 2 (for the second occurrence of a true EVT1 comparison). Display the EVT options and BIF mode. Execute the program, and determine how many bus cycles and microseconds there are from the start of the program until the second true EVT1 comparison.

Example:

```

ENTER:      > EV 1 P=2

              > EV
DISPLAY:    EVT1 A=0209 D=007D B=F P=00002
            EVT2 B=ALL
            BIF-1 S

ENTER:      > G 200
DISPLAY:    0209 7D      MOV  A,L      0000 87 51 03 25 02 75 02 51
            0209 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00036 USEC

ENTER:      > CN C

              > G 200
DISPLAY:    0209 7D      MOV  A,L      0000 87 51 03 25 02 75 02 51
            0209 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00023 CYCL

              >
    
```

Explanation:

The first true EVT1 comparison occurred 11 bus cycles after the start of the program as shown in example 2.A.3. The second true comparison occurred 23 bus cycles after the start of the program. Thus, 12 bus cycles elapsed between the first and second true EVT1 comparisons. Likewise, the program run time for one cycle of LOOP1 is $(36 - 17) = 19$ microseconds.

Comment:

The measured program run time may vary one or two microseconds when the program is executed repeatedly.

2.A.5. How many bus cycles occur if the pass count is set to 20?

Explanation:

The first pass count is satisfied on the first occurrence of a true EVT1 comparison, 11 bus cycles after the program starts. Each successive comparison requires 12 bus cycles. 19 EVT1 true comparisons later the pass count of 20 is satisfied; therefore, the total bus cycles for a pass count of 20 is $(19 \times 12 =) 228 + 11 = 239$ cycles.

Example:

```

ENTER:      > EV 1 P=20

            > G 200
DISPLAY:    0209 7D   MOV   A,L       0000 87 63 03 25 02 75 02 63
            0209 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00239 CYCL

            >
    
```

Comment:

If the P parameter exceeds the number of normal program loops, the pass counter will never be decremented to zero: no EVT trigger or breakpoint will ever occur. The break line in these examples at address 0209 is the same except for registers A and L (the first and last registers in the break line). These registers are incremented by one each time the program runs through LOOP1. Remember, the register values are in hexadecimal.

2.B. Set the BIF 1 mode breakpoint to break on an EVT1 comparison of options A, D, and B, plus EVT1 counting options P and C.

Explanation:

EVT option C is a delay command. The EVT command line entry for option C specifies the number of units of delay. The delay units are specified with the CNT command. The most recent CNT command remains in effect until changed.

EVT option C may be invoked with or without option P. If option C is invoked with option P, option P must be satisfied first before the delay of option C is started. If option C is invoked without option P, the delay is started when the EVT comparison options are satisfied (true).

- 2.B.1. Using the EVT1 option parameters specified in example 2.A, delay the EVT1 trigger and breakpoint another 500 bus cycles.

Example:

```

ENTER:      > EV 1 C=500

            > EV
DISPLAY:    EVT1 A=0209 D=007D B=F P=00020 C=00500
            EVT2 B=ALL
            BIF-1 S

ENTER:      > G 200
DISPLAY:    037F C28503 JNZ    0385          0000 12 3A 03 25 02 75 03 3A
            037F BREAK

ENTER:      > CN
DISPLAY:    COUNT=00739 CYCL

            >

```

Explanation:

The total bus cycles from the start of the program to the 20th EVT1 true comparison, 239 bus cycles elapse (refer to example 2.A.5). 239 cycles plus another 500 cycles delay equals 739 cycles.

- 2.B.2. Using only EVT1 option C, determine the breakpoint address if 12 bus cycles of delay are added to the EVT1 comparison options of address 0209.

Comments:

1. The delay of 12 bus cycles is equal to the program run time for one cycle of LOOP1 (see example 2.A.4). Therefore, the breakpoint address should be 0209, which is the next time through LOOP1. This is equivalent to setting a pass count of 2 (see example 2.A.4).
2. Options P and C may be deleted from the EVT command line by entering "P = 0" or "C = 0". Options P and C may be changed by entering a new parameter in the EVT command line.

Delete option P and change option C to C = 12.

Example:

```
ENTER:      > EV 1 P=0 C=12

            > EV
DISPLAY:    EVT1 A=0209 D=007D B=F C=00012
            EVT2 B=ALL
            BIF-1 S

ENTER:      > G 200
DISPLAY:    0209 7D      MOV  A,L      0000 87 51 03 25 02 75 02 51
            0209 BREAK

            >
```

2.C. Set the BIF 2 mode breakpoint to break on an EVT2 comparison of options A, D, and B, plus EVT2 counting option P.

2.C.1. Enter EVT2 options A, D, and B parameters for an address in LOOP2 of the sample program (between addresses 037B and 037E). Set BIF to break on EVT2; display the EVT options and BIF mode.

Example:

```
ENTER:      > EV 2 A=37E D=B9 B=MR

            > BIF CLR

            > BIF 2

            > EV
DISPLAY:    EVT1 A=0209 D=007D B=F C=00012
            EVT2 A=037E D=00B9 B=MR
            BIF-2 S

            >
```

2.C.2. Determine the number of bus cycles from the start of the program to address 037E.

Example:

```

ENTER:      > G 200
DISPLAY:    037E B9      CMP   C           0000 06 50 03 25 02 75 03 50
           037E BREAK

```

```

ENTER:      > CN
DISPLAY:    COUNT=00473 CYCL

```

>

2.C.3. Enter a pass count of 10 (the tenth occurrence of the EVT2 comparison). Display the EVT options and BIF mode, execute the program, and determine how many bus transactions there are from the start of the program to the tenth EVT2 comparison.

Explanation:

In example 2.A.4, it was determined that there are 12 bus transactions for each pass through LOOP1. LOOP1 and LOOP2 are equivalent. Therefore, 108 bus transactions (9×12) represent a pass count of 10, this added to the previous count of 473 from the start of the program to address 037E is equal to 581 total bus cycles.

Example:

```

ENTER:      > EV 2 P=10

```

```

           > EV
DISPLAY:    EVT1  A=0209 D=007D B=F C=00012
           EVT2  A=037E D=00B9 B=MR P=00010
           BIF-2 S

```

```

ENTER:      > G 200
DISPLAY:    037E B9      CMP   C           0000 16 47 03 25 02 75 03 47
           037E BREAK

```

```

ENTER:      > CN
DISPLAY:    COUNT=00581 CYCL

```

>

2.D. Set the BIF breakpoint on an EVT2 comparison of options A, D, and B, plus EVT2 counting options P and C.

2.D.1. Using the EVT2 option parameters in example 2.C, delay the EVT2 trigger and breakpoint another 150 bus cycles.

Example:

ENTER: > EV 2 C=150

DISPLAY: > EV
EVT1 A=0209 D=007D B=F C=00012
EVT2 A=037E D=00B9 B=MR P=00010 C=00150
BIF-2 S

ENTER: > G 200

DISPLAY 0386 C37B03 JMP 037B 0000 12 3B 03 25 02 75 03 3A
0386 BREAK

ENTER: > CN

DISPLAY: COUNT=00731 CYCL

>

2.D.2. Refer to example 2.D.1. Delete option P, and add an additional delay to option C, so that the total execution of the 731 cycles is unchanged.

Comment:

The additional delay equals the EVT2 pass count of 10 ($9 \times 12 = 108$ cycles), plus the delay set in option C. This equals a total delay of $108 + 150 = 258$ cycles.

Example:

ENTER: > EV 2 P=0 C=258

DISPLAY: > EV
EVT1 A=0209 D=007D B=F C=00012
EVT2 A=037E D=00B9 B=MR C=00258
BIF-2 S

ENTER: > G 200

DISPLAY: 0386 C37B03 JMP 037B 0000 12 3B 03 25 02 75 03 3A
0386 BREAK

ENTER: > CN

DISPLAY: COUNT=00731 CYCL

>

- 2.E. The BIF 1 or BIF 2 mode can be used together with the CNT 1 or CNT 2 commands to determine how many passes are made through a program loop, when a program is executed.
- 2.E.1. Set the EVT1 comparison options to an address within LOOP1 of the sample program. Set the EVT2 comparison options to an address outside the program loop. Set BIF 2 and CNT 1.

Explanation:

When EVT1 comparison options are set to an address within a program loop and CNT 1 is set, the counter will count the number of times EVT1 true comparisons are satisfied, the count is incremented by one for each pass through the loop.

Example:

```

ENTER:      > EV 1 CLR A=207 D=36
            > EV 2 CLR A=300 D=00

            > EV
DISPLAY:    EVT1 A=0207 D=0036 B=ALL
            EVT2 A=0300 D=0000 B=ALL
            BIF-2 S

ENTER:      > CN 1

            > G 200
DISPLAY:    0300 00      NOP                0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00038 EVT1

            >

```

- 2.E.2. Determine how many passes are made through LOOP2 of the sample program when the program is executed. Set the EVT1 comparison options to an address within LOOP2. Set the EVT2 comparison options to an address outside the program loop.

Example:

ENTER: > EV 1 CLR A=37B D=36

DISPLAY: > EV
EVT1 A=037B D=0036 B=ALL
EVT2 A=0300 D=0000 B=ALL
BIF-2 S

ENTER: > G 200

DISPLAY: 0300 00 NOP 0000 56 25 03 25 02 75 03 25
0300 BREAK

ENTER: > CN

DISPLAY: COUNT=00044 EVT 1

>

Comment:

The count can also be made using CNT 2, if the EVT2 comparison options are set to an address within a loop. When EVT2 is set within a loop, the BIF command should be cleared so that EVT2 does not generate the breakpoint. The sample program ends in a halt instruction that will generate the breakpoint when BIF is cleared.

- 2.E.3. Determine the pass count for both LOOP1 and LOOP2 of the sample program. Set the EVT1 address within LOOP1 and the EVT2 address within LOOP2. Clear the BIF mode. Enter CNT 1, and execute the program. Then, enter CNT 2 and execute the program again.

Example:

```

ENTER:      > EV CLR
            > EV 1 A=208
            > EV 2 A=37B
            > EV
DISPLAY:    EVT1 A=0208 B=ALL
            EVT2 A=037B B=ALL
            IND

ENTER:      > CN 1

            > G 200
DISPLAY:    0301 76      HLT                0000 56 25 03 25 02 75 03 25
            0301 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00038 EVT1

ENTER:      > CN 2

            > G 200
DISPLAY:    0301 76      HLT                0000 56 25 03 25 02 75 03 25
            0301 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00044 EVT2

            >

```

- 2.F. Most of the examples used in procedures 1 and 2 contain the starting address of the program for EVT1 and the ending address for EVT2. These are used to reduce the number of examples and still provide an adequate description of each parameter within a command line. By this time you have probably already experimented by setting the A, D, B, P, and C options in EVT1 and EVT2 command lines to other values. If not, go back over the various exercises and examples in these procedures and substitute values other than those used in the examples. This will not only demonstrate to you the versatility of the RTPA commands, but also give you practice in their usage.

Procedure 3—EVT Comparison and Counting Options With BIF ARM Mode

This procedure allows you to enter and display the EVT1 and EVT2 comparison and counting options (options A, D, B, P, and C) with the BIF ARM mode set. In addition, it shows you the display capabilities and the timing relationships of the DRT and CNT commands.

Exercises

- 3.A. Set the EVT1 and EVT2 comparison options A, D, and B to the first and last instruction of the sample program, with BIF ARM set. Determine the program run time and the total bus transactions between EVT1 and EVT2.

Comment:

BIF ARM is a sequential command. The EVT1 trigger arms EVT2. The EVT2 trigger breaks emulation and freezes the RTT buffer. The general purpose counter is started by the EVT1 trigger and stopped by the EVT2 trigger; therefore, the counter counts between EVT1 and EVT2 parameters.

- 3.A.1. Enter the EVT1 options A, D, and B from the start address of the sample program. See Fig. 3-3. Enter the EVT2 options A, D, and B from the program instruction preceding the halt instruction. Set the BIF mode to ARM. Then display the EVT options and BIF mode. Set the CNT command to microseconds, execute the program, and display the run time between EVT1 and EVT2 in microseconds. Then set the CNT command to bus cycles; execute the program again, and display the total bus cycles between EVT1 and EVT2.

Example:

```

ENTER:      > EV CLR
            > EV 1 A=201 D=11
            > EV 2 A=300 D=00
            > BIF ARM
            > EV
DISPLAY:    EVT1 A=0201 D=0011 B=ALL
            EVT2 A=0300 D=0000 B=ALL
            BIF-ARM S

ENTER:      > CN U
            > G 200
DISPLAY:    0300 00      NOP                0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=01541 USEC

ENTER:      > CN C
            > G 200
DISPLAY:    0300 00      NOP                0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00994 CYCL

            >

```

Explanation:

The EVT1 and EVT2 option parameters used in example 3.A.1 are identical to the parameters used in examples 1.C.3, 1.C.4, and 1.C.6. The only difference in the two exercises is the setting of the BIF mode. Note that when BIF 2 mode is used, the run time is 1545 usec and the bus cycles are 996 cycles. When BIF ARM mode is used, the run time is 1541 usec and the bus cycles are 994 cycles. When BIF 2 is set, the general purpose counter starts counting from the start of program execution; however, when BIF ARM is set, the counter starts after the EVT1 trigger is generated. This accounts for the differences in program timing and bus cycles between BIF 2 and BIF ARM modes.

- 3.B. Set the pass and delay count options P and C in the EVT1 and EVT2 command lines, with BIF ARM controlling the breakpoints. Using the `PORT` and `CNT` commands, verify that the breakpoint delay corresponds to the pass and delay count option parameters.

Comment:

When entering EVT option P, you must be careful to avoid creating a situation where an EVT trigger cannot be generated. The EVT comparison options A, D, B, and T must be satisfied at least the number of times designated by the option P parameter. If option A is entered, the address will usually occur within a program loop.

- 3.B.1. Enter the EVT1 option A and D parameters from an address within LOOP1 of the sample program and the EVT2 option A and D parameters from an address within LOOP2 of the sample program. Set the BIF mode to ARM. Display the EVT option parameters and the BIF mode. Execute the program and display the number of bus cycles between EVT1 and EVT2.

Comment:

This bus cycle count will be used later to determine whether the delays added by options P and C are correctly executed.

Example:

```

ENTER:      > EV 1 CLR A=207 D=36
            > EV 2 CLR A=37B D=36
            > EV
DISPLAY:    EVT1 A=0207 D=0036 B=ALL
            EVT2 A=037B D=0036 B=ALL
            BIF-ARM S

ENTER:      > G 200
DISPLAY:    037B 3600 MVI M,00      0000 56 75 03 25 02 75 03 50
            037B BREAK

ENTER:      > CN
DISPLAY:    COUNT=00461 CYCL

            >

```

- 3.B.2 Enter a pass count delay of 10 in the EVT1 command line. Execute the program, and display the number of bus cycles between EVT1 and EVT2.

Example:

```

ENTER:      > EV 1 P=10
            > G 200 .
DISPLAY:    037B 3600 MVI M,00      0000 56 75 03 25 02 75 03 50
            037B BREAK

ENTER:      > CN
DISPLAY:    COUNT=00353 CYCL

            >

```

Explanation:

In example 2.A.3., it was determined that there are 12 bus cycles in both LOOP1 and LOOP2. In this example, EVT1 is delayed by a pass count of 10; this represents 108 cycles ($9 \times 12 = 108$ cycles). Therefore, the number of cycles between EVT1 and EVT2 should decrease by 108 cycles to $(461 - 108 =)$ 353 cycles.

- 3.B.3 Enter a pass count delay of 20 in the EVT2 command line. Execute the program, and display the number of bus cycles between EVT1 and EVT2.

Example:

```

ENTER:      > EV 2 P=20

              > G 200
DISPLAY:    037B 3600   MVI   M,00       0000 12 3E 03 25 02 75 03 3D
              037B BREAK

ENTER:      > CN
DISPLAY:    COUNT=00581 CYCL

              >
    
```

Explanation:

In this example, EVT2 is delayed by a pass count of 20 (19 program loops); this represents 228 bus cycles. The number of cycles between EVT1 and EVT2 is therefore increased by 228 cycles over the count in 3.B.2 (353 + 228 = 581 cycles).

- 3.B.4. If a delay count (option C) is added to the EVT1 command line (with or without option P invoked), an EVT1 trigger cannot be generated; therefore, when the program is executed it will run until the halt instruction is reached. If your program does not have a halt instruction, press the ESC key to stop program execution.

Example:

```

ENTER:      > EV 1 C=100

              > G 200
DISPLAY:    0301 76     HLT           0000 56 25 03 25 02 75 03 25
              0301 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00000 CYCL

              >
    
```

- 3.B.5. Enter a delay count (option C) of 100 bus cycles in the EVT2 command line. Delete the EVT1 delay count entered in the preceding example. Execute the program, and display the number of bus cycles between EVT1 and EVT2.

Example:

```

ENTER:      > EV 1 C=0
            > EV 2 C=100

            > EV
DISPLAY:    EVT1 A=0207 D=0036 B=ALL P=00010
            EVT2 A=037B D=0036 B=ALL P=00020 C=00100
            BIF-ARM S

ENTER:      > G 200
DISPLAY:    037E B9      CMP   C          0000 12 35 03 25 02 75 03 35
            037E BREAK

ENTER:      > CN
DISPLAY:    COUNT=00681 CYCL

            >

```

Explanation:

In the above example, the break occurred at address 037E instead of address 037B (the EVT2 option A parameter). Address 037E occurs 100 bus cycles after address 037B; this corresponds to the 100-cycle delay entered for EVT2 option C.

3.B.6. In exercises 2.A.3 and 2.A.4, it was determined that the program run time and the number of bus cycles for one pass of LOOP1 (sample program) are 19 microseconds or 12 bus cycles. This was demonstrated by using BIF1 mode and the EVT pass count option P. This same information can be obtained more readily by using the BIF ARM mode, and entering the same option parameters in both EVT1 and EVT2 command lines for any address within a program loop.

Example:

```

ENTER:      > EV 1 CLR A=207 D=36
            > EV 2 CLR A=207 D=36

            > EV
DISPLAY:    EVT1 A=0207 D=0036 B=ALL
            EVT2 A=0207 D=0036 B=ALL
            BIF-ARM S

ENTER:      > G 200
DISPLAY:    0207 3600 MVI M,00      0000 87 50 00 00 02 75 02 51
            0207 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00012 CYCL

ENTER:      > CN U

            > G 200
DISPLAY:    0207 3600 MVI M,00      0000 87 50 00 00 02 75 02 51
            0207 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00020 USEC

            >
    
```

Explanation:

As previously stated, the reading obtained when measuring the program run time may vary one or two microseconds when the program is executed repeatedly. Thus, the difference between the two readings 19, and 20 microseconds.

3.B.7. Display the last 25 lines of the RTT buffer, to observe the start of the most recent program execution and the first pass through LOOP1. Enter the DRT command with a parameter of 25.

Example:

ENTER: > DR 25

DISPLAY:	ADDR	DATA	MNEMONIC	EXTERNAL	BUS
	0207	36	MVI M	00000000	M R F
	0208	00		00000000	M R
	0251	00		00000000	M W
	0200	F3	DI	00000000	M R F
	0201	11	LXI D	00000000	M R F
	0202	75		00000000	M R
	0203	02		00000000	M R
	0204	21	LXI H	00000000	M R F
	0205	50		00000000	M R
	0206	02		00000000	M R
	0207	36	MVI M	00000000	M R F
	0208	00		00000000	M R
	0250	00		00000000	M W
	0209	7D	MOV A,L	00000000	M R F
	020A	BB	CMP E	00000000	M R F
	020B	C2	JNZ	00000000	M R F
	020C	11		00000000	M R
	020D	02		00000000	M R
	0211	23	INX H	00000000	M R F
	0212	C3	JMP	00000000	M R F
	0213	07		00000000	M R
	0214	02		00000000	M R
	0207	36	MVI M	00000000	M R F
	0208	00		00000000	M R
	0251	00		00000000	M W

12 Bus Cycles

>

Procedure 4—EVT Comparison and Counting Options with BIF IND Mode

In this procedure you will enter and display the EVT1 and EVT2 comparison and counting options (options A, D, B, P, and C) using the BIF IND mode. In addition, the display capabilities and the timing relationships of the CNT command are demonstrated.

Exercises

4.A. The BIF IND mode operation is equivalent to setting both BIF 1 and BIF 2 to ON. Both breakpoints are independent of each other. Demonstrate the effects of the BIF IND mode in conjunction with the BIF return-option C. (Control of the program execution is returned back to the program.)

4.A.1. Enter the EVT1 and EVT2 comparison and counting option parameters A, D, B, P, and C. Set the BIF mode to IND and the BIF return-option to C. Display the EVT option settings and the BIF mode, and execute the program.

Example:

```

ENTER:      > EV 1 CLR A=209 D=7D B=F P=10 C=20
            > EV 2 CLR A=37E D=B9 B=MR P=10 C=150
            > BIF IND C
            > EV
DISPLAY:    EVT1 A=0209 D=007D B=F P=00010 C=00020
            EVT2 A=037E D=00B9 B=MR P=00010 C=00150
            BIF-1 C 2 C
ENTER:      > CN U
            > G 200
DISPLAY:    020A BB      CMP  E      0000 93 5A 00 00 02 75 02 5A
            020A BB      CMP  E      0000 83 64 00 00 02 75 02 64
            020A BB      CMP  E      0000 97 6E 00 00 02 75 02 6E
            037F C28503 JNZ  0385    0000 12 3F 03 25 02 75 03 3F
            037F C28503 JNZ  0385    0000 12 35 03 25 02 75 03 35
            037F C28503 JNZ  0385    0000 16 2B 03 25 02 75 03 2B
            0301 76      HLT                    0000 56 25 03 25 02 75 03 25
            0301 BREAK
ENTER:      > CN
DISPLAY:    COUNT=01544 USEC
    
```

(Example continued on next page)

```

ENTER:      > CN C

            > G 200
DISPLAY:    0212 C30702 JMP 0207      0000 93 5A 03 25 02 75 02 5B
            0212 C30702 JMP 0207      0000 83 64 03 25 02 75 02 65
            0212 C30702 JMP 0207      0000 97 6E 03 25 02 75 02 6F
            0386 C37B03 JMP 037B      0000 12 3B 03 25 02 75 03 3A
            0386 C37B03 JMP 037B      0000 12 27 03 25 02 75 03 26
            0301 76      HLT          0000 56 25 03 25 02 75 03 25
            0301 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00995 CYCL

            >

```

Comments:

1. When return-option C is invoked, the program execution control is automatically returned to the program after each breakpoint. The program continues to run until the halt command is reached or the ESC key is pressed (if your program does not have a halt command).
2. In example 4.A.1, since the EVT1 and EVT2 options are set to breakpoints within LOOP1 and LOOP2 respectively, each time the EVT parameters are satisfied (true), a break line is displayed and the program control is returned to the program until the next EVT parameters are true.
3. In example 4.A.1, note the difference in the breakpoint addresses when the delay units (option C) are changed from microseconds to bus cycles.

Procedure 5—EVT Comparison and Counting Options with BIF FRZ Mode

In this procedure you will enter and display the EVT1 and EVT2 comparison and counting options (options A, D, B, P, and C) using the BIF FRZ mode. The display capabilities and the timing relationships of the DRT and CNT commands are also demonstrated.

Exercises

- 5.A. Set the EVT1 and EVT2 comparison options. Execute the program; determine when the RTT buffer is frozen, and when the program execution is stopped.

5.A.1. Set the EVT1 comparison options to address 0209, the 11th bus cycle from the start of program execution. Set the EVT2 comparison options to address 0300, the last RTPA breakpoint in the sample program. Execute the program, display the last 11 lines in the RTT buffer, and determine when the general purpose counter is started and stopped.

Example:

ENTER: > EV 1 CLR A=209 D=7D

> EV 2 CLR A=300 D=00

> BIF FRZ

> EV

DISPLAY: EVT1 A=0209 D=007D B=ALL
 EVT2 A=0300 D=0000 B=ALL
 BIF-FRZ S

ENTER: > CN C

> G 200

DISPLAY: 0300 00 NOP 0000 56 25 03 25 02 75 03 25
 0300 BREAK

ENTER: > DR 11

ADDR	DATA	MNEMONIC	EXTERNAL	BUS
0200	F3	DI	00000000	M R F
0201	11	LXI D	00000000	M R F
0202	75		00000000	M R
0203	02		00000000	M R
0204	21	LXI H	00000000	M R F
0205	50		00000000	M R
0206	02		00000000	M R
0207	36	MVI M	00000000	M R F
0208	00		00000000	M R
0250	00		00000000	M W
0209	7D	MOV A,L	00000000	M R F

ENTER: > CN

DISPLAY: COUNT=00985 CYCL

ENTER: > CN U

> G 200

DISPLAY: 0300 00 NOP 0000 56 25 03 25 02 75 03 25
 0300 BREAK

(Example continued on next page)

```

ENTER:      > CN
DISPLAY:    COUNT=01529 USEC

```

```
>
```

Explanation:

The BIF FRZ mode is similar to the BIF ARM mode, except that the RTT buffer is frozen at EVT1. This can be seen when the last 11 lines of the RTT buffer are displayed. The last address in the buffer is 0209, the address of EVT1. The general purpose counter starts counting at EVT1 and stops at EVT2, the same as in BIF ARM mode. Program execution is stopped at EVT2.

- 5.A.2. Change the EVT2 address to 037E; add a pass count (option P) of 5 to each of the EVT command lines, and a delay count (option C) of 15 to the EVT2 command line. Execute the program, and display the run time between the settings of EVT1 and EVT2.

Example:

```
ENTER:      > EV 2 CLR A=37E D=B9 P=5 C=15
```

```
> EV 1 P=5
```

```
> EV
```

```
DISPLAY:    EVT1 A=0209 D=007D B=ALL P=00005
            EVT2 A=037E D=00B9 B=ALL P=00005 C=00015
            BIF-FRZ S
```

```
ENTER:      > G 200
```

```
DISPLAY:    037B 3600 MVI M,00      0000 16 4C 03 25 02 75 03 4B
            037B BREAK
```

```
ENTER:      > CN
```

```
DISPLAY:    COUNT=00732 USEC
```

```
>
```

Comment:

BIF FRZ reacts the same as BIF ARM to an EVT1 delay count. With an EVT1 delay count, no breakpoints can be generated until the halt instruction is reached or the ESC key is pressed. Therefore, in the preceding example a delay count (option C) was not added to EVT1 command line.

- 5.A.3. Add the return-option C to the BIF FRZ mode to demonstrate that the general purpose counter is stopped at EVT2.

Example:

```

ENTER:      > BIF FRZ C

              > G 200
DISPALY:    037B 3600   MVI   M,00      0000 16 4C 03 25 02 75 03 4B
              0301 76   HLT                   0000 56 25 03 25 02 75 03 25
              0301 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00732 USEC

              >
    
```

Comment:

When control is returned back to the program, the program continues to run after the EVT2 breakpoint is reached, and stops at the halt instruction. Note that the program run time is the same as in example 5.A.2; the general purpose counter was stopped at EVT2 and counts only the run time between EVT1 and EVT2.

- 5.A.4. When the return-option C is added to the BIF FRZ mode, the EVT2 trigger breaks the program execution and stops the general purpose counter. In addition, the EVT2 trigger arms the EVT1 comparator and enables the RTT buffer. This permits the RTT buffer to store the data again until the next EVT1 trigger is generated.

To illustrate how this works, set both the EVT1 and EVT2 parameters within a program loop, where both event parameters occur on each pass through the loop. Set EVT1 to address 0209 and EVT2 to address 0212 in LOOP1 of the sample program. Set the BIF FRZ mode for return-option C and execute the program. After several break lines are displayed, press the ESC key to stop the program execution.

Example:

```

ENTER:      > EV 1 CLR A=209 D=7D
            > EV 2 CLR A=212 D=C3
            > BIF FRZ C
            > EV
DISPLAY:    EVT1 A=0209 D=007D B=ALL
            EVT2 A=0212 D=00C3 B=ALL
            BIF-FRZ C

ENTER:      > G 200
DISPLAY:    0212 C30702 JMP 0207      0000 87 50 03 25 02 75 02 51
            0212 C30702 JMP 0207      0000 83 51 03 25 02 75 02 52
            0212 C30702 JMP 0207      0000 87 52 03 25 02 75 02 53
            0212 C30702 JMP 0207      0000 87 53 03 25 02 75 02 54
PRESS ESC  0212 C30702 JMP 0207      0000 83 54 03 25 02 75 02 55
KEY:       0212 BREAK
            >

```

5.A.5. Since the EVT1 and EVT2 addresses are near the beginning of the sample program, display the last 30 transactions from the RTT buffer. You will be able to determine if the RTT buffer was permitted to store information between each setting of EVT2 and the next occurrence of EVT1.

Example:

ENTER: > DR 30

DISPLAY:	ADDR	DATA	MNEMONIC	EXTERNAL	BUS
	0208	00		00000000	M R
	0256	00		00000000	M W
	0209	7D	MOV A,L	00000000	M R F
	0200	F3	DI	00000000	M R F
	0201	11	LXI D	00000000	M R F
	0202	75		00000000	M R
	0203	02		00000000	M R
	0204	21	LXI H	00000000	M R F
	0205	50		00000000	M R
	0206	02		00000000	M R
	0207	36	MVI M	00000000	M R F
	0208	00		00000000	M R
	0250	00		00000000	M W
	0209	7D	MOV A,L	00000000	M R F
	0207	36	MVI M	00000000	M R F
	0208	00		00000000	M R
	0251	00		00000000	M W
	0209	7D	MOV A,L	00000000	M R F
	0207	36	MVI M	00000000	M R F
	0208	00		00000000	M R
	0252	00		00000000	M W
	0209	7D	MOV A,L	00000000	M R F
	0207	36	MVI M	00000000	M R F
	0208	00		00000000	M R
	0253	00		00000000	M W
	0209	7D	MOV A,L	00000000	M R F
	0207	36	MVI M	00000000	M R F
	0208	00		00000000	M R
	0254	00		00000000	M W
	0209	7D	MOV A,L	00000000	M R F

From the start of program execution to the first EVT1 address.

The RTT buffer is enabled by EVT2. Bus transactions following EVT2 are stored until the next EVT1 trigger freezes the RTT buffer again.

>

Explanation:

Note in the preceding display of the RTT buffer that the buffer is frozen at address 0209. The EVT2 address 0212 contains a jump instruction. When the execution of this instruction is completed the EVT2 trigger is generated. Thus, the RTT buffer is again enabled, and will start storing the next instruction, the jump-to-address at 0207.

Procedure 6—EVT Comparison and Counting Options with BIF LIM Mode

In this procedure you will enter and display the EVT1 and EVT2 comparison and counting options A, D, B, P, and C, using the BIF LIM mode. The display capabilities and the timing relationships of the DRT and CNT commands will also be demonstrated.

The BIF LIM mode is probably the most misused and misunderstood BIF mode command. A number of examples are included here to better acquaint you with the BIF LIM mode. If you do not remember how the EVT1 and EVT2 parameters interact with each other when BIF LIM is invoked, review the BIF LIM mode discussion in Section 2 of this manual.

Exercises

6.A. Set the EVT1 breakpoint using BIF LIM mode.

Comment:

When the BIF LIM mode is set, EVT1 comparison options A, D, B, and T and EVT2 comparison options A and B, must be satisfied at the same time. If an option A parameter is entered in both of the EVT command lines, one or both of the option A parameters must be set to either < or >.

6.A.1. Using only the option A parameters for EVT1 and EVT2, set the parameters to break on EVT1.

Example:

```

ENTER:      > EV 1 CLR A=209
            > EV 2 CLR A<300
            > BIF LIM

            > EV
DISPLAY:    EVT1  A=0209 B=ALL
            EVT2  A<0300 B=ALL
            BIF-LIM S

ENTER:      > G 200
DISPLAY:    0209 7D      MOV  A,L      0000 83 50 03 25 02 75 02 50
            0209 BREAK

            >

```

Explanation:

In the above example, when the EVT2 option A parameter is set to <0300, the EVT2 comparison options A and B are satisfied at all addresses less than 0300. The program is executed at address 0200. When address 0209 is reached both the EVT1 and EVT2 option parameters are satisfied. The EVT1 trigger generates a breakpoint that stops execution and freezes the RTT buffer at address 0209.

6.B. Set the EVT2 breakpoint using the BIF LIM mode.

6.B.1. Using only the option A parameters for EVT1 and EVT2, set the parameters to break on EVT2.

Example:

```

ENTER:    > EV 1 CLR A>209
          > EV 2 CLR A=300
          > EV
DISPLAY:  EVT1 A>0209 B=ALL
          EVT2 A=0300 B=ALL
          BIF-LIM S

ENTER:    > G 200
DISPLAY:  0300 00      NOP                0000 56 25 03 25 02 75 03 25
          0300 BREAK

ENTER:    > DR 5
DISPLAY:  ADDR DATA MNEMONIC  EXTERNAL  BUS
          0381 03                00000000 M R
          0382 C3  JMP           00000000 M R F
          0383 00                00000000 M R
          0384 03                00000000 M R
          0300 00  NOP           00000000 M R F

          >
    
```

Explanation:

In the above example, the EVT1 comparison options are satisfied for all addresses greater than 0209, when the program is executed. When address 0300 is reached, both the EVT1 and EVT2 option parameters are satisfied. The EVT1 trigger breaks emulation and freezes the RTT buffer at address 0300.

- 6.C. Set the EVT1 breakpoint and add EVT options P and C to both the EVT1 and EVT2 command lines.
- 6.C.1. When a pass or delay count (options P or C) is added to the EVT1 command line, there is a delaying effect on the EVT1 trigger and breakpoint. However, a pass or delay count added to the EVT2 command line has no effect on the EVT1 trigger or breakpoint. Recall that only the EVT1 trigger can break program execution, freeze the RTT buffer, and stop the general purpose counter.

Example:

```
ENTER:    > EV 1 CLR A=209
          > EV 2 CLR A<37E

          > EV
DISPLAY:  EVT1 A=0209 B=ALL
          EVT2 A<037E B=ALL
          BIF-LIM S
```

Set the counter units to bus cycles.

```
ENTER:    > CN C

          > G 200
DISPLAY:  0209 7D      MOV  A,L      0000 56 50 03 25 02 75 02 50
          0209 BREAK

ENTER:    > CN
DISPLAY:  COUNT=00011 CYCL

          >
```

Add EVT option P to the EVT1 command line.

```
ENTER:    > EV 1 P=5

          > G 200
DISPLAY:  0209 7D      MOV  A,L      0000 87 54 03 25 02 75 02 54
          0209 BREAK

ENTER:    > CN
DISPLAY:  COUNT=00059 CYCL

          >
```

(Example continued on next page)

Add EVT option C to the EVT1 command line.

```
ENTER:    > EV 1 C=10

          > G 200
DISPLAY:  0207 3600 MVI M,00      0000 83 54 03 25 02 75 02 55
          0207 BREAK

ENTER:    > CN
DISPLAY:  COUNT=00069 CYCL

          >
```

Add EVT option P to the EVT2 command line.

```
ENTER:    > EV 2 P=10

          > G 200
DISPLAY:  0207 3600 MVI M,00      0000 83 54 03 25 02 75 02 55
          0207 BREAK

ENTER:    > CN
DISPLAY:  COUNT=00069 CYCL

          >
```

Add EVT option C to the EVT2 command line.

```
ENTER:    > EV 2 C=10

          > G 200
DISPLAY:  0207 3600 MVI M,00      0000 83 54 03 25 02 75 02 55
          0207 BREAK

ENTER:    > CN
DISPLAY:  COUNT=00069 CYCL

          >
```

Comments:

1. When the counting options P and C are added to the EVT1 command line the EVT1 trigger and breakpoint are affected. However, when the counting options P and C are added to the EVT2 command line there is no effect on the breakpoint.
2. When triggering an oscilloscope or logic analyzer with the EVT1 or EVT2 triggers, make sure that only the EVT1 trigger is used when BIF LIM mode is invoked.
3. The same results may be obtained if you rerun this example with the breakpoint set on EVT2 parameters.

6.D. Demonstrate the effects of adding EVT option D to both EVT command lines.

Comment:

Recall that in BIF LIM mode, the EVT1 command line compares all the comparison options (options A, D, B, and T), while EVT2 compares only options A and B. Therefore, any value entered into the EVT2 command line for option D will have no effect on the EVT1 trigger or breakpoint. When adding option D values to EVT1 command line, you should be careful not to set up a situation where the D values are not satisfied. When the EVT1 breakpoint is set, the option D value should equal the actual data at EVT1 program address. If you enter any other value for EVT1 option D, the correct < or > sign must be used. Remember that the value entered will be compared to the actual data value at the program address. If the EVT2 breakpoint is set, the actual data value at the EVT2 program address is compared to the option D value entered in the EVT1 command line and not to the value entered into EVT2 command line.

6.D.1. Set the EVT1 breakpoint and enter an arbitrary value for EVT2 option D.

Example:

```

ENTER:      > EV 1 CLR A=209 D=7D
            > EV 2 CLR A<37E D=FFFF

            > EV
DISPLAY:    EVT1 A=0209 D=007D B=ALL
            EVT2 A<037E D=FFFF B=ALL
            BIF-LIM S

ENTER:      > G 200
DISPLAY:    0209 7D      MOV  A,L      0000 83 50 03 25 02 75 02 50
            0209 BREAK

            >

```

Comment:

The EVT2 command line value FFFF for option D has no effect on the EVT1 breakpoint.

- 6.D.2. Set the breakpoint for EVT2 and enter the actual data value from the EVT2 program address as option D in the EVT1 command line. (At program address 037E the data value is B9. See Fig. 3-3.)

Example:

```

ENTER:      > EV 1 CLR A>209 D=B9
            > EV 2 CLR A=37E D=FFFF
            > EV
DISPLAY:    EVT1 A>0209 D=00B9 B=ALL
            EVT2 A=037E D=FFFF B=ALL
            BIF-LIM S

ENTER:      > G 200
DISPLAY:    037E B9      CMP   C           0000 06 50 03 25 02 75 03 50
            037E BREAK

            >
    
```

Comment:

Since the EVT1 option D value equals the actual data value at the EVT2 address, the operator sign may be =, < or >.

- 6.E. Demonstrate the effects of changing option B in both of the EVT command lines.

Comment:

The EVT option B is the only option that cannot be removed from the EVT command lines. Option B defaults to the ALL parameter and will remain as such until changed.

- 6.E.1. Set the breakpoint for EVT1 and change both the EVT1 and EVT2 option B parameters from ALL to F.

Example:

```

ENTER:      > EV 1 CLR A=209 B=F
            > EV 2 CLR A<37E B=F

            > EV
DISPLAY:    EVT1 A=0209 B=F
            EVT2 A<037E B=F
            BIF-LIM S

ENTER:      > G 200
DISPLAY:    0209 7D      MOV  A,L      0000 06 50 03 25 02 75 02 50
            0209 BREAK

            >

```

Now change the EVT2 option B from F to W.

Example:

```

ENTER:      > EV 2 B=W

            > G 200
DISPLAY:    0301 76      HLT          0000 56 25 03 25 02 75 03 25
            0301 BREAK

            >

```

Explanation:

When the EVT2 option B is changed to W, the comparison options of the EVT1 and EVT2 command lines are no longer satisfied at the EVT1 breakpoint; the program will run until it reaches the halt instruction or the ESC key is pressed.

- 6.F. Demonstrate the use of the BIF LIM command to find or observe specific data over a range of the program addresses.

Comment:

The BIF LIM command can be very useful in finding specific data, bus operations and data from the eight test probe clips. For instance, the sample program has two jump instructions (C3) in each program loop. By using the BIF LIM command, either jump instruction can be located and a breakpoint for that instruction established.

- 6.F.1. Set EVT1 option A greater than (>) the program starting address (0200). Set the EVT1 option D equal to the desired data (C3) to be found. Set EVT2 option A less than (<) address 0300, an address between the two loops in the sample program.

Example:

```

ENTER:      > EV 1 CLR A>200 D=C3
            > EV 2 CLR A<300

            > EV
DISPLAY:    EVT1 A>0200 D=00C3 B=ALL
            EVT2 A<0300 B=ALL
            BIF-LIM S

ENTER:      > G 200
DISPLAY:    0212 C30702 JMP   0207       0000 87 50 03 25 02 75 02 51
            0212 BREAK

            >
    
```

Explanation:

Data C3 was found at address 0212. Since address 0212 is greater than 0200 and less than address 0300, all comparison conditions of EVT1 and EVT2 were satisfied. An EVT1 trigger and breakpoint were generated.

- 6.F.2. Change EVT2 option A to greater than (>) address 0300. Execute the program again.

Example:

```

ENTER:      > EV 2 A>300

            > G 200
DISPLAY:    0386 C37B03 JMP   037B       0000 06 50 03 25 02 75 03 4F
            0386 BREAK

            >
    
```

Comment:

Since EVT2 option A is set to greater than address 0300, all comparison conditions are now satisfied at address 0386. Note that the first program breakpoint at address 0212 could also have been obtained by using BIF 1 instead of BIF LIM. However, with the BIF LIM command set, the additional conditions imposed by EVT2 give greater control over the establishment of a breakpoint.

Procedure 7—Pre-, Center, and Post-Triggering

This procedure demonstrates how you can simulate pre-, center, and post-triggering conditions around a desired event by entering various combinations of the RTPA commands.

NOTE

Recall, from earlier discussions of the RTPA triggering modes, that the data is continually being stored in the RTT buffer. The simulation of pre-, center, and post-triggering depends on the timing relationship of the event occurrence to the freezing of the RTT buffer.

Pre-Triggering

7.A. Pre-triggering is the capturing of data that has occurred prior to a desired event. The two most common pre-triggering methods are as follows:

7.A.1. Set EVT1 option A to the desired event address. Set the BIF mode to BIF 1. (You could optionally use EVT2 and BIF 2 instead of EVT1 and BIF1.) Execute the program. When the program address matches the specified option A address, trigger 1 will generate a system interrupt and freeze the RTT buffer on the desired event. The RTT buffer will then contain 128 bus transactions that were stored prior to the desired event. The following example shows the entry commands for this method of pre-triggering.

Example:

```

ENTER:      > EV 1 A=300
            > BIF CLR
            > BIF 1
            > G 200
DISPLAY:    0300 00      NOP          0000 56 25 03 25 02 75 03 25
            0300 BREAK
            >

```

Comment:

To display any portion of the RTT buffer, use the DRT command.

7.A.2. Set EVT1 option A to the program starting address and the EVT2 option A to the desired event address. Set the BIF mode to BIF ARM and execute the program. The program stops execution and the buffer is frozen at the desired event address. The following example shows the entry commands for this method of pre-triggering.

Example:

```

ENTER:      > EV 1 A=200
            > EV 2 A=300
            > BIF ARM
            > G 200
DISPLAY:    0300 00      NOP          0000 56 25 03 25 02 75 03 25
            0300 BREAK
            >
    
```

Comment:

To display any portion of the RTT buffer, use the DRT command.

Center Triggering

7.B. Center triggering is the capturing of data around a desired event; the desired event should occur in the middle of the RTT storage buffer. There are several combinations of RTPA commands that will provide center triggering. The most common method uses either EVT1 or EVT2 comparison options set to the desired event parameters. The CNT command is set to count bus transactions, and a count delay of 64 is added to the desired event parameters. The following example shows the entry commands for this method of center triggering.

Example:

```

ENTER:      > EV 1 A=375 C=64
            > CN C
            > BIF CLR
            > BIF 1
            > G 200
DISPLAY:    0386 C37B03 JMP 037B      0000 16 4C 03 25 02 75 03 4B
            0386 BREAK
            >
    
```

Explanation:

The EVT1 trigger is delayed by 64 bus transactions. The RTT buffer holds 128 bus transactions; when the buffer is frozen the desired event parameters will occur approximately in the middle of the RTT storage buffer.

Post-Triggering

- 7.C. Post-triggering is the capturing of data that occurs after a desired event. This method of triggering can be accomplished similar to the center triggering method, by changing the count delay to 128. Retain the BIF and CNT command settings from the previous example. The following example shows the entry commands for this method of post-triggering.

Example:

```
ENTER:      > EV 1 A=200 C=128

            > G 200
DISPLAY:    0207 3600 MVI M,00      0000 97 59 03 25 02 75 02 5A
            0207 BREAK
            >
```

Comment:

Since the CNT and BIF commands set in the previous example remain unchanged, you only need to make the above entries. If you enter "DR *", the complete buffer is displayed. The address 0200 should appear near the start of the buffer display, as follows:

Example:

```
ENTER:      > DR *

DISPLAY:    ADDR DATA MNEMONIC EXTERNAL BUS
            0202 75          00000000 M R
            0203 02          00000000 M R
            0204 21 LXI H 00000000 M R F
            0205 50          00000000 M R
            0206 02          00000000 M R
            0207 36 MVI M 00000000 M R F
            0208 00          00000000 M R
            0250 00          00000000 M W
            0209 7D MOV A,L 00000000 M R F
PRESS ESC   020A BB CMP E 00000000
KEY:
            >>
```

Explanation:

Note in the above display that address 0200 is not the first address in the buffer display. This is caused by two factors:

1. The 128-cycle delay is started after the instruction cycle of 0200 is finished.
2. When the buffer is frozen, the complete instruction cycle is stored in the buffer.

Section 4 RTPA COMMAND INTERACTIONS

	Page
Introduction	4-1
Interaction With 8001 System Commands	4-1
Interaction With 8002A System Commands	4-1
Interaction With 8001/8002A Debug Commands	4-1
Interactions Between RTPA Commands	4-2

Section 4

RTPA COMMAND INTERACTIONS

INTRODUCTION

This section describes how the RTPA commands interact with the 8001/8002A commands and with the other RTPA commands.

INTERACTION WITH 8001 SYSTEM COMMANDS

The 8001 is always under Debug control and all the RTPA commands may be invoked at any time. With the 8001, there is no RTPA interaction between RTPA commands and 8001 System commands.

INTERACTION WITH 8002A SYSTEM COMMANDS

The RTPA commands BIF, RTT, and DRT cannot be invoked until the 8002A is under Debug control. The CNT and EVT commands can be invoked at any time, whether or not the 8002A is under Debug control.

When using the RTPA commands, there are times when it is necessary to abort Debug control, to invoke a System command. This is done with the ABORT (* or /) command. If ABORT * is used, the EVT command lines and the currently assigned BIF or EVT mode are cleared. If ABORT / is used, only the currently assigned BIF or EVT mode is cleared; the EVT command lines are not cleared. Therefore, it is recommended that you use ABORT /, to preserve the EVT command line entries.

INTERACTION WITH 8001/8002A DEBUG COMMANDS

The RTPA expands the capabilities of the Debug system. As such, if either the TRACE or BKPT Debug command is invoked, the RTPA commands are ineffective.

The TRACE command must be OFF to enable the RTPA to perform any of its functions.

The BKPT command can set two breakpoints. These breakpoints can be set to break on address and optionally, read/write qualifiers. If a BKPT command breakpoint is set at the same program address as a RTPA breakpoint, the BKPT command breakpoint has priority and controls the program breaking actions.

INTERACTIONS BETWEEN RTPA COMMANDS

There are a few minor interactions between the RTPA commands. The two that cause the most confusion are the use of BIF CLR and EVT CLR, and the use of EVT1 option C when the ARM or FRZ mode is set.

Referring back to Section 2, the BIF CLR command clears only the currently assigned EVT or BIF mode. The EVT CLR command also clears the currently assigned EVT or BIF mode, and in addition, clears the EVT command lines. Either command causes the EVT or BIF mode to default to EVT IND mode.

The EVT1 option C command must not be invoked if either the ARM or FRZ mode is set. The EVT1 delay counter (set by EVT1 option C) cannot be enabled if either ARM or FRZ mode is set.

Section 5 RTPA APPLICATIONS

	Page
Introduction	5-1
Equipment Required	5-1
Emulation Mode 0	5-2
Software Debugging Procedure	5-2
Software Debbing Demonstration	5-2
Isolating the Program Bus	5-11
Software Development	5-14
Timing Considerations in Mode 0	5-15
Optomizing the Code in Time Critical Situations	5-15
Emulation Modes 1 and 2	5-16
Prototype Control Probe	5-16
Software/Hardware Integration Procedure	5-16
Software/Hardware Integration Demonstration	5-17
Timing Comparisons for Modes 0, 1 and 2	5-26
Using the Test Probe Clips	5-27
Test Probe Demonstration	5-29
Using the Logic Analyzer	5-32
Logic Analyzer Demonstration No. 1	5-35
Connections Between the RTPA, Logic Analyzer and Prototype	5-35
Set-up Procedures for the Logic Analyzer	5-36
Set-up Procedures for the 8001/8002A	5-38
Triggering the Logic Analyzer With the RTPA	5-39
Logic Analyzer Demonstration No. 2	5-42
Connections Between the RTPA, Logic Analyzer and Prototype	5-42
Set-up Procedures for the Logic Analyzer	5-43
Set-up Procedures for the 8001/8002A μ Processor Lab	5-44
Triggering the Logic Analyzer With the RTPA	5-44
Summary	5-51

ILLUSTRATIONS

Fig. No.		Page
5-1	Data Acquisition Interface simplified block diagram	5-28
5-2	Data Acquisition Interface panel controls and connectors	5-29
5-3	Typical interconnection showing the RTPA triggering the logic analyzer	5-33
5-4	Initial display for adjusting oscilloscope controls	5-37
5-5	Timing diagram of data bus and lower eight bits of the address bus	5-39
5-6	State table display of the 16 states between the cursor and trigger	5-40
5-7	Timing diagram showing relationships between timing lines from the microprocessor	5-45
5-8	Timing diagram showing cursor and cursor word on the data byte of 7D	5-46
5-9	Timing diagram showing the cursor and cursor word at the start of the instruction cycle MVI M,0	5-48
5-10	Timing diagram showing the added wait state to each machine cycle when the program runs in the program memory in emulation Mode 1	5-49
5-11	An expanded timing diagram of channels 0–3. Each step of the FINE CURSOR control is equal to 10 ns, permitting timing measurements within the display	5-50

TABLES

Table No.		Page
5-1	Program Run Time Comparisons for Modes 0, 1, and 2	5-26
5-2	Breakdown of T States for Program Instruction MVI M,0	5-48

Section 5

RTPA APPLICATIONS

INTRODUCTION

The RTPA is a powerful debugging tool, with many applications in the software/hardware design and integration phases of a microprocessor-based system. This section describes the basic steps required to use the RTPA commands in the debugging and evaluation of the software/hardware design. As you become more experienced in using the RTPA commands, your own understanding of microprocessor-based system design and imagination will lead you to additional uses of the RTPA.

The first part of this section defines the applications of the RTPA commands in relation to the three 8001/8002A emulation modes. The examples show you how to enter the various RTPA commands and how to obtain the desired displays to determine the location of the software/hardware bugs. Also included in this section are narrative descriptions of the practical uses of the RTPA in software debugging. The last part of this section is devoted to the use of the Tektronix 7D01 Logic Analyzer in conjunction with the RTPA.

EQUIPMENT REQUIRED

The following is a list of the equipment required to demonstrate debugging the software/hardware design in a prototype system. The sample program is shown in Fig. 3-3 and is reproduced in Appendix D for ready reference. This program is used for the demonstrations in this section. If you are using an emulator processor other than the 8080A, the RTPA commands are entered and displayed in a similar fashion; however, the equipment associated with the 8080A microprocessor should be replaced with equipment for the microprocessor you are using.

- 8001/8002A μ Processor Lab
 - Real-Time Prototype Analyzer (RTPA)
 - 8080A Emulator Support
 - 8080A Prototype Control Probe
- System Terminal: CT 8100 or 4024/4025 CRT Terminal, or CT 8101 Printing Terminal (only one required).
- 8080A Prototype System—(simulated by an 8080A demonstration aid.)
- 7D01/DF2 Logic Analyzer and Display Formatter plug-in, inserted in a Tektronix 7000-series mainframe oscilloscope.

EMULATION MODE 0

In emulation Mode 0, the RTPA is normally used to debug the user's prototype software program. The following debugging procedures are centered around this use of the RTPA in emulation Mode 0.

When operating in emulation Mode 0, the RTPA can be used to debug the prototype program from the early stages of software development through the final evaluation of the program. The prototype program, when loaded into the program memory, is contained wholly within the 8001/8002A. The prototype control probe, the 7D01 Logic Analyzer, and the prototype system (hardware), from the preceding equipment list are not used for this mode of operation.

Software Debugging Procedure

One of the first steps after loading a new program into the program memory is to see whether the program will execute. If it executes properly, debugging may not be necessary; however, if the program does not execute, or if you suspect improper execution, then you must use debugging tools to find the problem.

If the program is fairly short, you may want to obtain a line-by-line display of each bus transaction in the program. To obtain such a display, you would enter the Debug command TRACE ALL. This TRACE display permits you to check the code instructions for accuracy, and provides the register status for every bus transaction. When the program is long, this procedure is very time-consuming; however, with the proper use of the RTPA commands, program bugs can be isolated more readily.

Your prototype program will usually consist of modules and/or subroutines that provide convenient groupings within the program for the setting of breakpoints. Starting with the first module, set the RTPA breakpoints to determine if each module or group of modules is executed properly. When you come to a module that does not execute as expected, set breakpoints within that module or at the various subroutine within the module, until the program bug is isolated to a small program segment. By displaying the RTT buffer contents and/or tracing each line with the TRACE ALL command, you should be able to isolate the incorrect data byte or logic flaw.

Software Debugging Demonstration

The following demonstration of software debugging contains simplified step-by-step examples to show how a program bug can be detected using the RTPA commands.

The sample 8080A program contained in Appendix D is used for this demonstration. Refer to Section 3 for instructions on how to load this sample program into the 8001/8002A program memory.

Load the sample program.

Example:

```
ENTER:      > LO RTPA4
DISPLAY:    TRANSFER ADDRESS: 0200
            *LOAD* EOJ

            >
```

NOTE

For demonstration purposes, the sample program is assigned an object file name of RTPA4.

A simulated bug is introduced in the program by changing one of the data bytes, to prevent the program from executing properly. In a practical situation, such a bug could result from:

- A typographical error made when entering your program into the program memory with the EXAM or PATCH commands.
- Assigning the wrong program instruction in your program.

You have loaded the sample program in program memory; now a program bug is simulated by changing the data byte at address 037D from 7D to 7E with the EXAM command as follows:

Example:

```
ENTER:      > E 37D
```

When the CR (carriage return) is pressed, the data byte at 037D is displayed:

Example:

```
DISPLAY:    037D=7D
```

Now enter 7E to change the data byte from 7D to 7E.

Example:

```
DISPLAY:    037D=7D-7E B9
```

Execute the program.

NOTE

If you are using the 8002A, enter the DEBUG command before you execute the program.

Example:

ENTER: > DEB

> G 200

DISPLAY:	LOC	INST	MNEM	OPER	SP	RF	RA	RB	RC	RD	RE	RH	RL
PRESS ESC	0202	00	NOP		F8FE	87	00	03	25	02	75	03	87
KEY:	0202	BREAK											

>>

NOTE

The program did not indicate a program break at the halt instruction. Thus, the program does not appear to have executed properly. Press the ESC key to break program execution.

The sample program zeros the data byte at program memory locations 0250—0275 and 0325—0350. Dump the program memory from address locations 0200—0400. Check to see if the data bytes at the proper memory address locations were zeroed.

Example:

ENTER:

DISPLAY:

```
>>D 200 400
0200=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0210=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0220=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0230=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0240=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0250=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0260=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0270=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0280=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0290=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02A0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02B0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02C0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02D0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02E0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02F0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0300=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0310=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0320=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0330=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0340=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0350=00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0360=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0370=FF FF FF FF FF 01 25 03 21 50 03 36 00 7E B9 C2
0380=85 03 C3 00 03 2B C3 7B 00 00 00 00 00 00 00 00
0390=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03A0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03B0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03C0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03D0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03E0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03F0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0400=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

>

An examination of program memory locations 0200—0400 indicates that all the data below address 0350 and all the data above address 0388 are changed to zeroes. The first half of the program is destroyed.

Reload the program and re-enter the simulated bug at address 037D.

NOTE

Prior to reloading the program, fill all memory locations between addresses 0200 and 0400 with "FF", so that a change in any address location will be readily noticeable. Also, before reloading the program, the DEBUG command must be terminated. This is done with the "A /" command. After reloading the program re-enter the DEBUG command.

Example:

```

ENTER:      > A /
            > F 200 400 FF
DISPLAY:    *FILL* EOJ

ENTER:      > LO RTPA4
DISPLAY:    TRANSFER ADDRESS: 0200
            *LOAD* EOJ

ENTER:      > E 37D

DISPLAY:    037D=7D-7E B9

ENTER:      > DEB
            >

```

Now set the RTPA breakpoints within or just after the first half of the sample program. Refer to Appendix D for setting breakpoints in the sample program. Set the EVT1 address to 0375. This is the first address in the last half of the program. Set BIF to "1" and execute the program.

Example:

```
ENTER: > EV 1 A=375
      > BIF 1
      > G 200
```

```
DISPLAY: LOC INST MNEM OPER SP RF RA RB RC RD RE RH RL
          0375 012503 LXI B,0325 0000 56 75 03 25 02 75 02 75
          0375 BREAK
      >
```

The break line in the preceding example indicates that program execution was stopped at address 0375. It appears that the first half of the program executed properly. To verify this, dump the program memory from addresses 0200—0375 and see if the data bytes at addresses 0250—0275 contain zeroes.

Example:

```
ENTER: > D 200 375
DISPLAY: 0200=F3 11 75 02 21 50 02 36 00 7D BB C2 11 02 C3 75
          0210=03 23 C3 07 02 FF FF FF FF FF FF FF FF FF FF
          0220=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0230=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0240=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0250=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
          0260=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
          0270=00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF
          0280=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0290=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          02A0=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          02B0=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          02C0=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          02D0=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          02E0=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          02F0=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0300=00 76 FF FF FF FF FF FF FF FF FF FF FF FF FF
          0310=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0320=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0330=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0340=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0350=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0360=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0370=FF FF FF FF FF 01 25 03 21 50 03 36 00 7E B9 C2
      >
```

Since the program has zeroed memory locations 0250—0275, the first half of the program did execute properly.

Now set EVT2 to an address within the last half of the program (LOOP 2); this permits the program to run through LOOP 2 once. Address 0385 is the last address before the jump instruction. Set BIF to "2". Display the settings of both EVT lines, and execute the program from address 0375.

Example:

```

ENTER:      > EV 2 A=385
            > BIF CLR
            > BIF 2
            > EV
DISPLAY:    EVT1 A=0375 B=ALL
            EVT2 A=0385 B=ALL
            BIF-2 S
ENTER:      > G 375
DISPLAY:    0385 2B      DCX  H          0000 87 00 03 25 02 75 03 4F
            0385 BREAK
            >

```

In the preceding example the program did break on address 0385; this indicates the program executed properly. Dump the program memory from addresses 0340—0350 to see if the data at memory location 0350 was properly cleared.

Example:

```

ENTER:      > D 340 350
DISPLAY:    0340=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
            0350=00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
            >

```

The data at program memory address 0350 is zeroed; thus the program did execute the instructions in LOOP 2 the first time. Enter a pass count of 2 to EVT2; this will cause the program to run through LOOP 2 one more time. Execute the program from address 0375, then dump the contents of program memory from address 0340 to see if the data at memory location 034F is properly cleared.

Example:

```

ENTER:      > EV 2 P=2

            > EV
DISPLAY:    EVT1 A=0375 B=ALL
            EVT2 A=0385 B=ALL P=00002
            BIF-2 S

ENTER:      > G 375
DISPLAY:    0385 2B      DCX  H      0000 87 00 03 25 02 75 03 4E
            0385 BREAK

ENTER:      > D 340
DISPLAY:    0340=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00

            >
    
```

The program has apparently executed properly through LOOP 2 the second time. Clear the BIF mode, and execute the program again from address 0375 to see if the program stops execution on the Halt instruction at address 0301.

Example:

```

ENTER:      > BIF CLR

            > EV
DISPLAY:    EVT1 A=0375 B=ALL
            EVT2 A=0385 B=ALL P=00002
            IND

ENTER:      > G 375
PRESS ESC  0082 00      NOP      D632 87 00 03 25 02 75 03 87
KEY:       0082 BREAK

            >>
    
```

This time the program did not break; the ESC key is pressed to break program execution. Dump the program memory between address locations 0200 and 0400 to determine if the program is still loaded in the program memory.

Example:

```

ENTER: >>D 200 400
DISPLAY: 0200=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0210=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0220=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0230=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0240=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0250=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0260=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0270=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0280=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0290=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02A0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02B0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02C0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02D0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02E0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02F0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0300=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0310=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0320=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0330=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0340=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0350=00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0360=FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0370=FF FF FF FF FF 01 25 03 21 50 03 36 00 7E B9 C2
0380=85 03 C3 00 03 2B C3 7B 00 00 00 00 00 00 00 00
0390=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03A0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03B0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03C0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03D0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03E0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03F0=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0400=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

>

It can be seen from this display that the contents of the program memory locations are the same as when the program was originally executed (before the debugging began). This indicates that the bug must be somewhere within LOOP 2. There are several courses of action that can be taken at this point; let's examine them separately.

Isolating the Program Bug

Using the TRACE Display. Since LOOP 2 is fairly short and contains only a few instructions, the TRACE ALL command can be used to display a line-by-line trace of LOOP 2 when the program is executed from address location 0375. Watch the line-by-line trace display, and when LOOP 2 is traced, press the ESC key to stop the program execution.

Example:

```

ENTER:      > TR ALL

              > G 375
DISPLAY:    0375 012503 LXI   B,0325   D632 87 00 03 25 02 75 03 87
            0378 215003 LXI   H,0350   D632 87 00 03 25 02 75 03 50
            037B 3600  MVI   M,00      D632 87 00 03 25 02 75 03 50
            037D 7E    MOV   A,M       D632 87 00 03 25 02 75 03 50
            037E B9    CMP   C         D632 87 00 03 25 02 75 03 50
            037F C28503 JNZ   0385   D632 87 00 03 25 02 75 03 50
            0385 2B    DCX   H         D632 87 00 03 25 02 75 03 4F
            0386 C37B00 JMP   007B    D632 87 00 03 25 02 75 03 4F
            007B 00    NOP                    D632 87 00 03 25 02 75 03 4F
            007C 00    NOP                    D632 87 00 03 25 02 75 03 4F
            007D 00    NOP                    D632 87 00 03 25 02 75 03 4F
            007E 00    NOP                    D632 87 00 03 25 02 75 03 4F
            007F 00    NOP                    D632 87 00 03 25 02 75 03 4F
            0080 00    NOP                    D632 87 00 03 25 02 75 03 4F
            0081 00    NOP                    D632 87 00 03 25 02 75 03 4F
PRESS ESC   0082 00    NOP                    D632 87 00 03 25 02 75 03 4F
KEY:        0082 BREAK
              >

```

Note in the preceding line-by-line trace display that the "jump to" address at program address 0386 has been changed to 007B; in the sample program of Appendix D, this "jump to" address is 037B. A closer examination of the trace display also reveals that the program instruction at address 037D has been changed from MOV A, L to MOV A, M. The data byte has also, been changed from 7D to 7E. (This is the simulated bug that you planted in your sample program.) Change the data byte at program address 037D back to 7D with the EXAM command. Load and execute the program again to make sure it executes properly.

Setting a RTPA Breakpoint. In many cases, a module or subroutine can contain many more program instructions than the sample program. In such instances, using a line-by-line trace would be a tedious undertaking. Note that in the last dump of the program memory from addresses 200 to 400, the data at program address memory locations 0350 to 0200 were all changed to zeroes. The program should zero the data at program memory locations from 0350 down to 0325. If a write-protection breakpoint is set for memory location 0324, the program will stop execution whenever an attempt is made to change the data at memory location 0324. A write-protection breakpoint is set with the EVT option B = MW command (memory writes only).

Reload the program and set the EVT1 breakpoint to break on a memory write bus transaction at program location 0324.

NOTE

Don't forget to terminate the DEBUG command before you load the program, and re-enter the simulated bug into the program.

Example:

```

ENTER:      > A /

            > F 200 400 FF
DISPLAY:    *FILL* EOJ

ENTER:      > LO RTPA4
DISPLAY:    TRANSFER ADDRESS: 0200
            *LOAD* EOJ

ENTER:      > E 37D

DISPLAY:    037D=7D-7E B9

ENTER:      > DEB

            > EV CLR

            > EV 1 A=324 B=MW

            > BIF 1

            > G 200

DISPLAY:    LOC  INST  MNEM  OPER      SP    RF  RA RB RC RD RE RH RL
            037B 3600  MVI   M,00    0000  87  00 03 25 02 75 03 24
            037B BREAK

            >

```

In the preceding display, program execution was stopped on program address 037B. The program instruction MVI M,00 at this address moves the data byte "00" to the memory location indicated by the H and L registers. Note in the displayed break line that the H and L registers contain the address 0324; therefore, the data byte "00" is written into program memory address 0324. Since the EVT1 comparison options A and B are looking for address 0324 and a memory write instruction, a match is made and the EVT1 trigger breaks program execution. A display of the RTT buffer contents shows that the above statements are valid.

Display the last 30 bus transactions from the RTT buffer.

Example:

ENTER: > DR 30

```

DISPLAY: ADDR DATA MNEMONIC EXTERNAL BUS
          0388 03          00000000 M R
          037B 36 MVI M 00000000 M R F
          037C 00          00000000 M R
          0326 00          00000000 M W ---Memory address
          037D 7E MOV A,M 00000000 M R F 0326 changed
          0326 00          00000000 M R to "00".
          037E B9 CMP C 00000000 M R F
          037F C2 JNZ          00000000 M R F
          0380 85          00000000 M R
          0381 03          00000000 M R
          0385 2B DCX H 00000000 M R F
          0386 C3 JMP          00000000 M R F
          0387 7B          00000000 M R
          0388 03          00000000 M R
          037B 36 MVI M 00000000 M R F
          037C 00          00000000 M R
          0325 00          00000000 M W ---Memory address
          037D 7E MOV A,M 00000000 M R F 0325 changed
          0325 00          00000000 M R to "00".
          037E B9 CMP C 00000000 M R F
          037F C2 JNZ          00000000 M R F
          0380 85          00000000 M R
          0381 03          00000000 M R
          0385 2B DCX H 00000000 M R F
          0386 C3 JMP          00000000 M R F
          0387 7B          00000000 M R
          0388 03          00000000 M R
          037B 36 MVI M 00000000 M R F
          037C 00          00000000 M R
          0324 00          00000000 M W ---Memory address
          >                                     0324 changed
                                                to "00".

```

In the preceding display, the last address stored in the RTT buffer, before program execution was stopped and the buffer frozen, was address 0324. The display also shows that the program is continuing to zero out each memory location.

A quick look at the order of program execution in the preceding RTT buffer display shows that the program was executed in the following order:

1. The data byte "00" is moved into a memory location with the MVI M instruction.
2. That same data byte "00" is then read into the accumulator with the MOV A,M instruction.
3. Register C is compared to the data byte in the accumulator with the CMP C instruction

NOTE

The accumulator contents are always "00" and register C always contains "25"; when the two are subtracted the remainder will never be zero. Thus, the execution of the program continues around the loop.

4. The jump (JNZ) instruction causes the program to jump ahead to address 0385.
5. The H-L register pair is decremented by "1" with the DCX H instruction.
6. The jump (JMP) instruction causes the program to jump back to address 037B.
7. The program execution cycle starts over again.

This order of execution is not the same as the correct order of execution for the sample program. Compare the RTT display closely with the sample program (Appendix D). Note that the MOV A, M instruction is incorrect. Change the data byte at program address 037D back to 7D with the EXAM command. Load and execute the program again to insure that it executes properly.

These two demonstrations describe two methods for isolating a bug in a program. As you become more proficient at using the RTPA commands, you will discover other methods for accomplishing the same results. You might consider arbitrarily changing other data bytes in the sample program, to give yourself practice in debugging a program.

Software Development

During the earlier stages of software development, and prior to the availability of a hardware system, the modules and/or subroutines in a program can be executed and debugged with the RTPA command. For instance, suppose your program issues interrupts to the hardware system that cause certain addresses to be dumped on the address bus, for the purpose of vectoring your program to a certain routine. When operating in emulation Mode 0, these interrupts are not permitted; however, you can still debug that portion of the program by forcing the program to jump to the vector location. The GO command followed by the vector address, forces the emulator to jump to that location and execute the routine. By setting the RTPA breakpoints and displaying the contents of the RTT buffer, you can observe the actual execution of the routine. Most of the logical bugs can be detected and corrected in this mode prior to the final integration of the software and hardware, and before the hardware design is completed.

Timing Considerations in Mode 0

As previously stated, the prototype program is executed entirely within the environment of the 8001/8002A, when operating in emulation Mode 0. The speed of execution or run time of the program is governed by the 8001/8002A system clock. With the proper settings of the RTPA commands, any portion of the program can be timed in milliseconds or microseconds. The time for the execution of any portion of the program in Mode 0 will be different from the execution time in emulation Modes 1 and 2. Emulation Modes 1 and 2 use the prototype clock; the difference in execution time between Mode 0 and Modes 1 or 2 corresponds directly to the relationship between the 8001/8002A clock and the prototype clock.

NOTE

With some emulator processors, an additional factor affects the expected and actual time measurements. Some emulator processors add a wait state to each machine cycle. These wait states must be subtracted from the actual measurement to arrive at the expected measurement. Appendix A of this manual contains the characteristics for each 8001/8002A emulator processor.

Optimizing the Code in Time Critical Situations

You'll often want to reduce or optimize the code within a program, so that the overall run time is reduced. With the proper settings of the various RTPA commands, you can determine how much time is spent in each subroutine, compared to the overall program run time. Even though the actual time measurements in Mode 0 may differ from the measurements obtained in Modes 1 or 2, the ratio of the time spent in a certain subroutine compared to the overall program run time is a constant ratio.

This ratio or percentage of time spent in each subroutine is very important in time-critical situations. If you know that a large percentage of time is spent in a particular subroutine, you have a good place to start optimizing the program code.

The decision to optimize code depends to a great extent on what the program is actually doing. In an operating system consisting of a number of monitoring terminals or sensors, the monitoring routines should be as short as possible. In this case, the RTPA commands can be used to show you which routines are candidates for optimizing. Remember, the RTPA cannot optimize the code; that's the job of the programmer. However, the RTPA can direct you to the area in your program where optimizing may be desirable.

EMULATION MODES 1 AND 2

In the design of a microprocessor-based system, a software designer and a hardware designer can start the software/hardware integration phases when the following tasks are complete:

- The prototype program is fully debugged in emulation Mode 0 and is executing properly within the environment of the program memory. (However, the execution of interrupt routines cannot be tested in Mode 0.)
- The hardware designer has a completed prototype system.

The first encounter of the prototype software with the prototype system hardware is made while operating in emulation Mode 1. Any part or all of the prototype program, located in program memory, can be mapped and moved to the prototype system memory.

There are three primary differences between the three emulation modes: the location of the program; the source of the clock; and the execution of interrupt routines. Remember that the clock for Mode 0 is provided by the 8001/8002A and the clock for Modes 1 and 2 is provided by the prototype system. In Mode 0, the program can execute interrupt routines by a forced jump. In Modes 1 and 2, interrupts from the prototype can be executed properly.

Prototype Control Probe

When operating in either emulation Mode 1 or 2, the microprocessor device in the prototype is removed and replaced by the Prototype Control Probe's plug. Each emulator processor has its own unique Prototype Control Probe. Refer to Appendix C for information on how to install the Prototype Control Probe.

In emulation Modes 1 and 2, with the Prototype Control Probe attached to the prototype system, the emulator processor and the RTPA (in the 8001/8002A) have full control over the prototype system.

Software/Hardware Integration Procedure

After the prototype program has been debugged in emulation Mode 0, the Prototype Control Probe can be connected to the prototype system and the emulation mode changed to emulation Mode 1. Before the program is mapped to the prototype memory, you should execute the program again to ensure that it will run in emulation Mode 1, which uses the prototype system's clock.

You might want to measure the program run time in Mode 1 and compare it to the program run time in Mode 0. If the clock frequencies are different, the run times will also be different. This will be shown in a demonstration later in this section.

In emulation Mode 1, the memory mapping capabilities of the 8001/8002A can be used to map the various portions of the prototype program to the prototype memory, for integration with the prototype hardware. When all of the prototype program is mapped to the prototype memory, the program runs entirely within the prototype system.

If the program does not execute properly in the prototype's program memory, the memory accesses can be mapped back to the program memory and executed again, to determine whether the bug is in the prototype software or prototype hardware.

If you determine that the bug is in the prototype hardware, you can isolate the problem by setting the desired event breakpoints and displaying the RTT buffer contents. You may need to use a logic analyzer to further isolate the problem to a particular circuit.

Software/Hardware Integration Demonstration

The following demonstration contains simplified step-by-step examples that show how to integrate the software design (prototype program) with the hardware design (prototype system). Also included are examples showing how to establish the breakpoints and display the RTT buffer contents.

The sample 8080A program in Appendix D is used for this demonstration. Refer to Section 3 in this manual for instructions on how to load the sample program into program memory.

The following demonstration is based on the following assumptions:

1. The sample program in Appendix D is correctly loaded into program memory.
2. The sample program executes properly in emulation Mode 0.
3. The Prototype Control Probe is properly connected between the prototype system (hardware) and the emulator processor (in the 8001/8002A).
4. If you are using the 8002A, the Debug mode has been invoked (you have entered the DEBUG command).

Before you leave emulation Mode 0, record the program run time in microseconds for LOOP1, LOOP2, and the overall program. Use the last address in LOOP1 as address 0210 and the last address in LOOP2 as address 0300. Set the breakpoint for EVT1. Display the EVT options and breakpoint.

Example:

```

ENTER:      > EV CLR

             > EV 1 A=210

             > EV 2 A=300

             > BIF 1

             > EV
DISPLAY:    EVT1 A=0210 B=ALL
            EVT2 A=0300 B=ALL
            BIF-1 S

             >

```

Set the counter to count microseconds. Execute the program from its starting address (0200), and display the run time for LOOP1.

Example:

```

ENTER:      > CN U

             > G 200
DISPLAY:    020E C37503 JMP 0375 0000 56 75 03 25 02 75 02 75
            020E BREAK

ENTER:      > CN
DISPLAY:    COUNT=00717 USEC

             >

```

Now set the breakpoint for EVT2. Execute the program from the LOOP2 starting address (0375), and display the run time for LOOP2.

Example:

```
ENTER:      > BIF CLR
            > BIF 2
            > G 375
DISPLAY:    0300 00      NOP          0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=00829 USEC

            >
```

Keep the breakpoint at EVT2; execute the program from its starting address (0200), and display the total run time for the program.

Example:

```
ENTER:      > G 200
DISPLAY:    0300 00      NOP          0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=01545 USEC

            >
```

In the preceding examples, it can be seen that if the LOOP1 run time is added to the LOOP2 run time, the sum equals the total run time of the program. (Remember, from previous examples, that differences of one or two microseconds can often result when the program run time for several modules are compared to the overall program run time.) The program run time in emulation Mode 0 will be compared later to the run time of the same program operating in emulation Modes 1 and 2.

Change the emulation mode to Mode 1.

Example:

```
ENTER:      > EM 1
DISPLAY:    *EMULATE* EOJ

            >
```

Set the breakpoint for EVT1. Execute the program from its starting address (0200), and display the run time for LOOP1, when operating in emulation Mode 1.

Example:

```

ENTER:      > BIF CLR

            > BIF 1

            > G 200
DISPLAY:    020E C37503 JMP   0375      0000 56 75 03 25 02 75 02 75
            020E BREAK

ENTER:      > CN
DISPLAY:    COUNT=02867 USEC

            >

```

Set the breakpoint for EVT2. Execute the program from the LOOP2 starting address (0375) and display the run time for LOOP2. Then execute the program again from its starting address at 0200, and display the total run time for the program. Remember that the program is executing in emulation Mode 1.

Example:

```

ENTER:      > BIF CLR

            > BIF 2

            > G 375
DISPLAY:    0300 00      NOP           0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=03314 USEC

ENTER:      > G 200
DISPLAY:    0300 00      NOP           0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=06180 USEC

            >

```

NOTE

The program run times that you measure in emulation Mode 1 may not correspond exactly to these shown here. The exact program run time depends upon the prototype system or demonstration aid you are using.

Note that the run time for the program is considerably longer in Mode 1 than it was for the same program in emulation Mode 0. This means that the prototype clock is at a much slower frequency than the 8001/8002A system clock.

It is important that you establish a relationship between the two clocks. If the prototype clock is slower than the 8001/8002A clock, and if your prototype program executes properly in emulation Mode 0, you can be assured that your program will also run in the prototype's memory (if the hardware is properly designed). However, if the prototype's clock is faster than the 8001/8002A clock, critical timing routines in your program may require additional tuning, when the program runs in the prototype's memory with a faster clock.

You are now ready to transfer the data contained in program memory to the prototype memory. This is done with the MAP and MOVE commands.

Use the MAP command to map the program memory. Then, display the mapping assignments to insure that the correct map assignments were made. Remember that memory is mapped in increments of 128 data bytes. In the sample program, each program address contains one data byte.

Example:

```
ENTER:      > MA U 200-2FF

           > MA R
DISPLAY:    0000-01FF=P  0200-02FF=U
           0300-FFFF=P

           >
```

Use the MOVE command to transfer the data from the program memory addresses associated with LOOP1 to prototype memory. (These are the same addresses that you just mapped to the user's prototype.)

Example:

```
ENTER:      > MOV PU 200 2FF 200
DISPLAY:    *MOVE* EOJ

           >
```

In these examples, you have transferred LOOP1 of the program from program memory to prototype memory. The first half of the program is in prototype memory; the last half of the program is still in program memory. Retain the same EVT1 and EVT2 breakpoints, set in the previous examples (when the program run time was determined). Set the breakpoint for EVT1, execute the program from its starting address (0200), and display the run time for LOOP1 in prototype memory.

Example:

```

ENTER:      > BIF CLR
            > BIF 1
            > G 200
DISPLAY:    020E C37503 JMP   0375      0000 56 75 03 25 02 75 02 75
            020E BREAK

ENTER:      > CN
DISPLAY:    COUNT=02252 USEC

            >

```

This time, set the breakpoint for EVT2, execute the program from the LOOP2 starting address (0375), and display the run time for LOOP2. Remember, LOOP2 still resides in program memory.

Example:

```

ENTER:      > BIF CLR
            > BIF 2
            > G 375
DISPLAY:    0300 00      NOP           0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=03313 USEC

            >

```

Execute the program from its starting address (0200) and display the total run time for the program.

Example:

```

ENTER:      > G 200
DISPLAY:    0300 00      NOP           0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=05565 USEC

            >

```

The first half of the program executes properly in prototype memory, thus, you are now ready to transfer the complete program to the prototype memory. Map the program memory addresses of the complete program to the prototype memory, using the MAP command. Then transfer the data at these program memory addresses to the prototype memory, using the MOVE command.

Example:

```
ENTER:      > MA U 200-3FF

            > MA R
DISPLAY:    0000-01FF=P  0200-03FF=U
            0400-FFFF=P

ENTER:      > MOV PU 200 3FF 200
DISPLAY:    *MOVE* EOJ

            >
```

The complete program now resides in prototype memory. Set the breakpoint for EVT1, execute the first half of the program from the starting address 0200, and display the run time for LOOP1. This run time should be equal to the previous execution time of LOOP1.

Example:

```
ENTER:      > BIF CLR

            > BIF 1

            > G 200
DISPLAY:    020E C37503 JMP  0375      0000 56 75 03 25 02 75 02 75
            020E BREAK

ENTER:      > CN
DISPLAY:    COUNT=02253 USEC

            >
```

Set the breakpoint for EVT2, execute the program from LOOP2 starting address (0375), and display the run time for LOOP2. Since both LOOP1 and LOOP2 execute properly, execute the program again from its starting address (0200), and display the run time for the complete program in the prototype memory.

Example:

```

ENTER:      > BIF CLR
            > BIF 2
            > G 375
DISPLAY:    0300 00   NOP           0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=02602 USEC

ENTER:      > G 200
DISPLAY:    0300 00   NOP           0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=04854 USEC

            >

```

The complete program has executed in prototype memory. Although we are still operating in emulation Mode 1, we are simulating emulation Mode 2. To verify this, change the emulation mode to Mode 2; then, execute the program from its starting address (0200), and display the run time for the complete program in emulation Mode 2. This run time should equal the program run time in Mode 1.

Example:

```

ENTER:      > EM 2
DISPLAY:    *EMULATE* EOJ

ENTER:      > G 200
DISPLAY:    0300 00   NOP           0000 56 25 03 25 02 75 03 25
            0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=04853 USEC

            >

```


Note that, even though the program has been transferred to the prototype memory, it is not erased from the program memory. To verify this, change the emulation mode back to Mode 0; execute the program again (in program memory) from the starting address, and display the program run time for Mode 0.

Example:

```

ENTER:      > EM 0
DISPLAY:    *EMULATE* EOJ

ENTER:      > G 200
DISPLAY:    0300 00      NOP      0000 56 25 03 25 02 75 03 25
           0300 BREAK

ENTER:      > CN
DISPLAY:    COUNT=01545 USEC

           >

```

This demonstration on the software/hardware integration has taken you through all three emulation modes of operation. You have set various breakpoints and obtained timing measurements for each mode. The same breakpoints were used throughout the demonstration in an attempt to shorten the demonstration. Bugs were not introduced in the prototype making it possible for you to develop a better understanding of what is actually taking place.

If the program does not execute properly in the prototype, you should be able to visualize the steps required to isolate the bug. Here are some of the steps you could take:

1. Reset the breakpoint to other addresses or data within the program.
2. Determine which RTPA breakpoint should be used and select the proper settings of the BIF commands.
3. Vary the start of program execution, from the start of the program to other addresses within the program.
4. Obtain the proper display of the RTT buffer. This is very important, since the RTT buffer contains all the actual transactions that have taken place on the data bus during program execution.
5. Use the general purpose counter to obtain timing measurements and other counting features.
6. Use a logic analyzer or the test probe clips, to further define a hardware bug. These steps are covered later in this section.

Timing Comparisons For Modes 0, 1 and 2

Some of the 8001/8002A emulator processors add wait states to every machine cycle when operating in Mode 0 or Mode 1; that is when the program runs in program memory. These wait states affect program run times (in milliseconds or microseconds) and the counting of emulator clock cycles. Appendix A of this manual contains the characteristics for each 8001/8002A emulator processor, and the number of wait states added in emulation Modes 0 and 1. Refer to Appendix A for the characteristics of your emulator processor.

The 8080A emulator processor used in the preceding demonstration adds one wait state to each machine cycle, when operating in emulation Modes 0 and 1. This variation in the program execution time is evident when you compare the run time in program memory to the run time in prototype memory. The addition of the one wait state for each machine cycle accounts for this variation when the program is executed in program memory.

Table 5-1 shows the variations in program run times of the sample program and of the two loops in the sample program.

**Table 5-1
Program Run Time Comparisons for Modes 0, 1 and 2**

Sample Program	Emulation Mode 0	Emulation Mode 1		Emulation Mode 2
		Program Memory	Prototype Memory	
Loop 1 Run Time	717	2867	2253	2252
Loop 2 Run Time	829	3314	2602	2603
Overall Program Run Time	1545	6180	4854	4853
Clock Source	8001/8002A System			

Note from the program run times listed in Table 5-1, and from the software/hardware integration demonstration, that the run times for emulation Modes 1 and 2 are three to four times greater than the run times in emulation Mode 0. In emulation Modes 1 and 2, the system clock is supplied by the prototype; therefore, the prototype clock rate is considerably slower than the 8001/8002A clock rate. Remember that the program run times for Modes 1 and 2 in Table 5-1 may differ somewhat from those you obtain. Any differences depends upon which 8080A prototype system or demonstration aid you are using.

The differences in the Mode 1 program run time for program memory and prototype memory will be explained in a demonstration later in this section. Appendix A will show you how to calculate the two readings obtained in emulation Mode 1.

Recall that the RTPA can be used to point out areas where the program code may be optimized, if the timing of a program is critical. Compare the program run times in Table 5-1; you will see that the final check on these timing measurements should be done in emulation Mode 2, or emulation Mode 1, with the program executing in the prototype memory. This is the actual program run time as measured with the RTPA. Refer to Appendix A for an explanation of actual program run time.

USING THE TEST PROBE CLIPS

The P6451 Data Acquisition Probe has eight data channels and one clock channel. These channels are terminated in probe test clips, which can be attached to any 8-bit data source and to the clock source of the 8-bit data. It is not necessary to connect all eight probe clips; however, the clock probe clip must be connected to the clock source of the data if any of the probe test clips are used. Any unused probe clips should be connected to logic ground.

The P6451 probe connector is inserted into the Data Acquisition Probe socket on the Data Acquisition Interface panel (installed in the rear panel of the 8001/8002A). The clock channel in the P6451 probe is used to clock the data channels through the Data Acquisition Interface. Refer to the simplified block diagram in Fig. 5-1. On each clock pulse, the data storage registers are enabled and the eight data channels D0—D7 are presented to the Event comparators and RTT buffer in the Real-Time Trace module. Without this clock pulse, the probe data would never leave the Data Acquisition Interface unit.

The binary states of each data channel from the probe are stored in the RTT buffer on the trailing edge of the SLV OPREQ signal. To refresh your memory on how the probe data is stored in the RTT buffer, refer back to Fig. 1-5 and the associated text.

When the probe clock is synchronized with the SLV OPREQ signal, each time the data storage registers in the Data Acquisition Interface are clocked, new probe data is also stored in the RTT buffer. A display of the RTT buffer shows this new data for each bus transaction.

If the source of the clock from the probe is faster than the SLV OPREQ signal, the new probe data is presented to the RTT buffer at a faster rate than it is being stored. Therefore, some of the new probe data will not be stored in the RTT buffer. However, if the source of the clock from the probe is slower than the SLV OPREQ signal, the new probe data is presented to the RTT buffer at a slower rate, and the same probe data will be stored on two or more consecutive SLV OPREQ signals. A display of the RTT buffer will show the same data on more than one consecutive bus transaction.

Therefore, the frequency of the probe clock should be equal or less than the SLV OPREQ signal; preferably within 50%. When the frequencies of the probe clock and the SLV OPREQ signal are not equal, the data from the probe test clips has no correlation to the program addresses and bus transactions. A dump of the RTT buffer will, however, still display the digital states for each of the eight data bits.

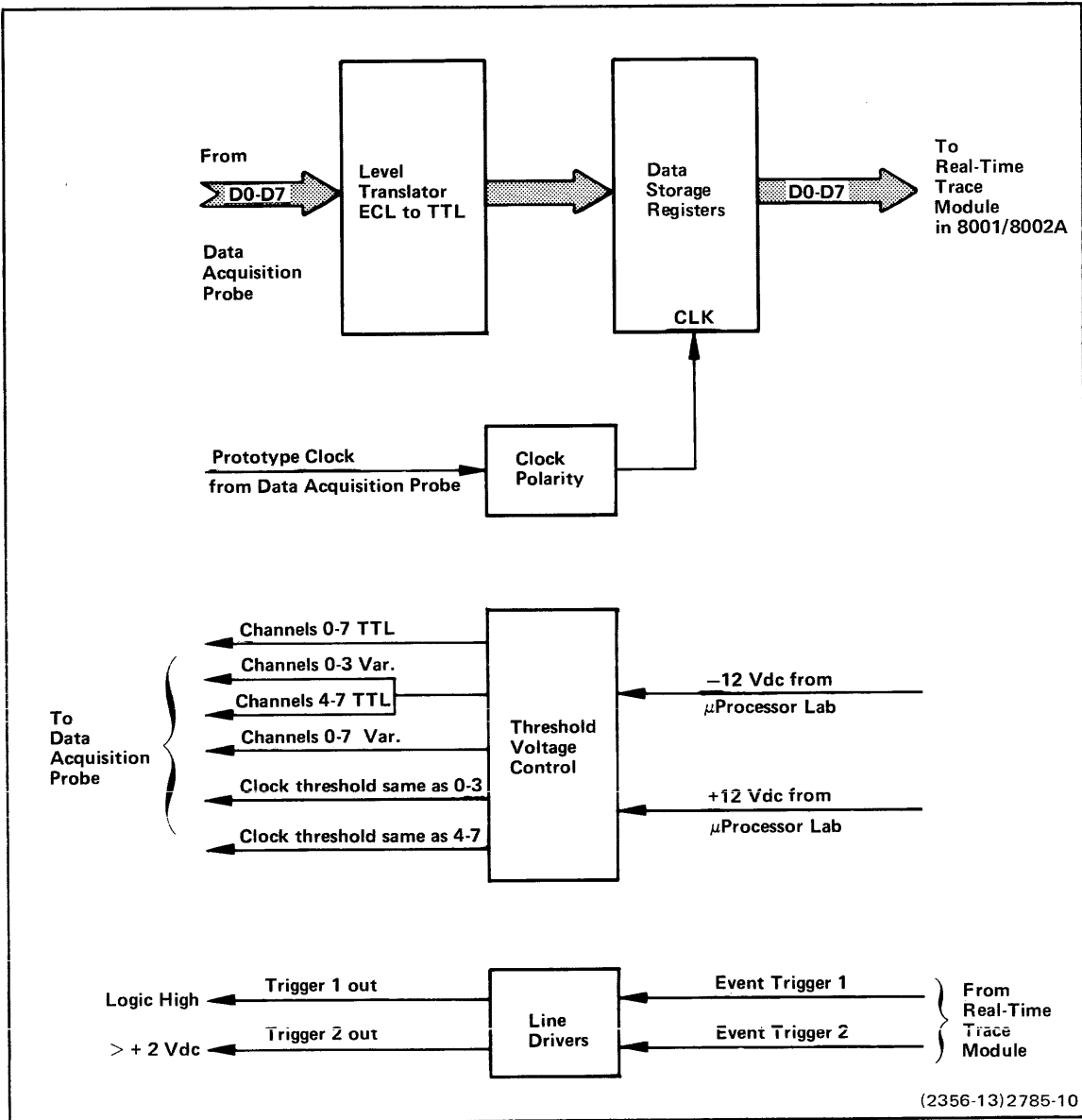


Fig. 5-1. Data Acquisition Interface simplified block diagram.

The EVT command option T acts as a word recognizer. When the eight digits of option T are specified, they are compared to the eight data channels from the P6451 probe. When a match is made in the comparators and the proper breakpoint is set, the EVT trigger stops program execution, and the data around this recognized word can be displayed. If not all eight data channels are used, or if some channels are not desired, enter "X" (don't care) in the EVT option T parameters. Remember that all eight digits must be entered in the EVT command option T parameters.

During normal Mode 0 operation, the prototype control probe is not used. All operations occur within the 8001/8002A. For this reason, the EVT option T and the test probe clips are seldom used in emulation Mode 0. However, in emulation Modes 1 and 2, the test probe clips and EVT option T can be very useful in debugging the prototype hardware.

Test Probe Demonstration

The following demonstration will show you how to connect the P6451 test probe clips and how to obtain a display of the RTT buffer.

Refer to Fig. 5-2 for the location of the controls on the Data Acquisition Interface panel. Set the Threshold Level Switch to the down position (TTL CH 0—7) Set the External Clock Polarity Switch to the up position (\lrcorner).

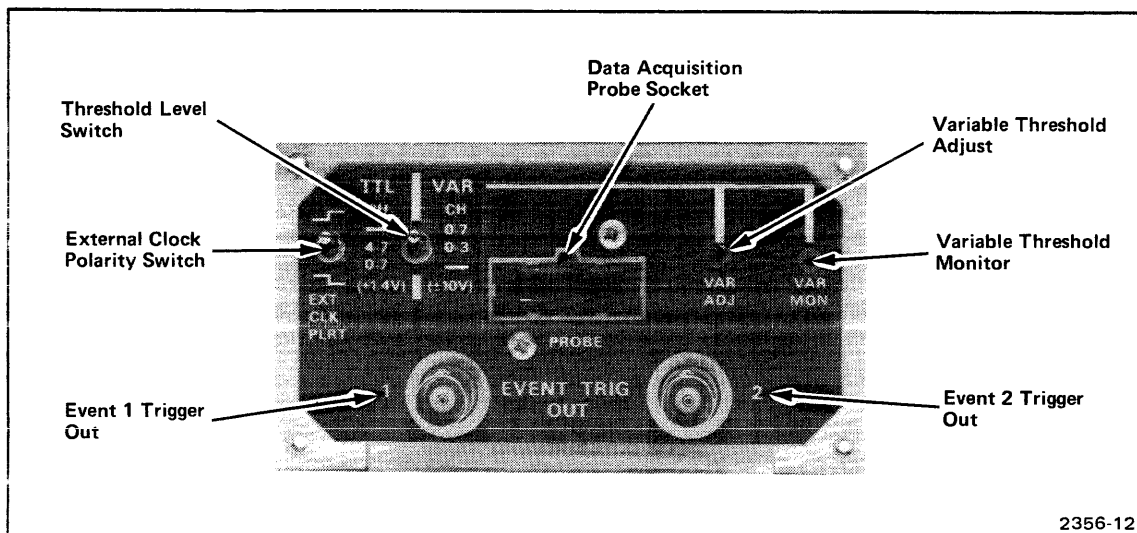


Fig. 5-2. Data Acquisition Interface panel controls and connectors.

Connect the P6451 probe connector to the Data Acquisition Probe Socket (labeled "PROBE"; refer to Fig. 5-2). Connect eight probe test clips to the prototype system data bus. Make sure the channels are connected properly; Channel 0 to D0, Channel 1 to D1, etc. Connect the ground (GND) test clip to logic ground in the prototype system. Connect the clock (C) test clip to the prototype system clock.

The sample 8080A program contained in Appendix D is used in this demonstration. Refer to Section 3 of this manual for instructions on how to load the sample program into program memory.

Change the emulation mode to Mode 1. Map the complete program to the prototype memory and move the program from program memory to the prototype memory. Set EVT2 to break on the last RTPA breakpoint address in the program (at address 0300). Execute the program from its starting address to make sure the program runs in the prototype memory.

Example:

```

ENTER:      > EM 1
DISPLAY:    *EMULATE* EOJ

ENTER:      > MA U 200-3FF

                > MA R
DISPLAY:    0000-01FF=P  0200-03FF=U
            0400-FFFF=P

ENTER:      > MOV PU 200 3FF 200
DISPLAY:    *MOVE* EOJ

ENTER:      > DEB

                > BIF CLR

                > BIF 2

                > EV 2 A=300

                > EV
DISPLAY:    EVT1 B=ALL
            EVT2 A=0300 B=ALL
            BIF-2 S

ENTER:      > G 200

DISPLAY:    LOC  INST  MNEM  OPER      SP   RF   RA  RB  RC  RD  RE  RH  RL
            0300  00    NOP           0000  56   25  03  25  02  75  03  25
            0300  BREAK

                >

```

The display shows that the program did execute properly and the program stopped at address 0300. When you dump the last ten lines of the RTT buffer, you can observe the binary states of the eight data lines connected to the probe test clips.

Example:

ENTER: > DR 10

DISPLAY:	ADDR	DATA	MNEMONIC	EXTERNAL	BUS
	0325	00		00000000	M W
	037D	7D	MOV A,L	01111101	M R F
	037E	B9	CMP C	10111001	M R F
	037F	C2	JNZ	11000010	M R F
	0380	85		10000101	M R
	0381	03		00000011	M R
	0382	C3	JMP	11000011	M R F
	0383	00		00000000	M R
	0384	03		00000011	M R
	0300	00	NOP	00000000	M R F

>

Note that in the display of the RTT buffer, the binary values of the test clips are equal to the hexadecimal values listed in the DATA column, since the probe test clips are attached to the data lines.

Set EVT1 to break on a test clip value of B9 (10111001). Execute the program from its starting address and dump the last ten lines in the RTT buffer.

Example:

ENTER: > BIF CLR

> BIF 1

> EV 1 T=10111001

> EV

DISPLAY: EVT1 T=10111001 B=ALL
EVT2 A=0300 B=ALL
BIF-1 S

ENTER: > G 200

DISPLAY: 037E B9 CMP C 0000 06 50 03 25 02 75 03 50
037E BREAK

>

Example:

```

ENTER:      > DR 10

DISPLAY:    ADDR DATA MNEMONIC  EXTERNAL  BUS
            0376 25          00100101  M R
            0377 03          00000011  M R
            0378 21 LXI     H    00100001  M R F
            0379 50          01010000  M R
            037A 03          00000011  M R
            037B 36 MVI     M    00110110  M R F
            037C 00          00000000  M R
            0350 00          00000000  M W
            037D 7D MOV     A,L  01111101  M R F
            037E B9 CMP     C    10111001  M R F

            >

```

The EVT1 comparison is satisfied when the EVT1 option T parameters match the value of the test clips. Since BIF 1 is set, this EVT1 match generates an EVT1 trigger, which stops program execution and freezes the RTT buffer at the test clip binary value of 10111001.

Disconnect the eight probe test clips; do not disconnect the ground or clock probe test clips. Connect the probe test clip of channel 0 to any TTL test point in the prototype system. Set the EVT1 option T parameters to X (don't care) for all of the channels except channel 0. Set channel 0 to either "0" or "1". Execute the program again and display the contents of the RTT buffer.

You can experiment, by connecting other test probe channels to TTL test points in the prototype, until you are familiar with the operation of the probe test clips and the EVT command option T. Remember that the test clips can be connected to any 8-bit data source.

USING THE LOGIC ANALYZER

The equipment list at the beginning of this section includes the 7D01/DF2 Logic Analyzer and Display Formatter. The DF2 Display Formatter is a companion unit to the 7D01 Logic Analyzer, and provides the following additional features:

- State table and map displays for up to 16 data channels.
- Hex, octal, and binary readouts of data, cursor, and trigger.
- Data comparison features.

In the following discussions and demonstration, using the logic analyzer could (with some modifications) be used without the display formatter. However, when the term "logic analyzer" is used in this section, you can generally assume that the display formatter is also included.

When the logic analyzer is connected to the prototype system, it provides timing and state table displays of up to 16 binary bits. These bits are in addition to the eight binary bits provided by the probe test clips.

The logic analyzer can be triggered by the RTPA EVT1 or EVT2 triggers. The two BNC connectors on the Data Acquisition Interface unit are used to transfer the trigger pulse from the RTPA to the logic analyzer. The BNC connectors are located on the 8001/8002A back panel. (Refer to Fig. 5-2.) A cable with BNC connectors on each end is connected from the desired EVENT TRIG OUT connector to the EXTERNAL TRIG INPUT on the logic analyzer.

Fig. 5-3 is a typical interconnection, showing the 8001/8002A and logic analyzer connected to the prototype system. Note the interconnecting BNC cable, which provides the trigger pulse to the logic analyzer.

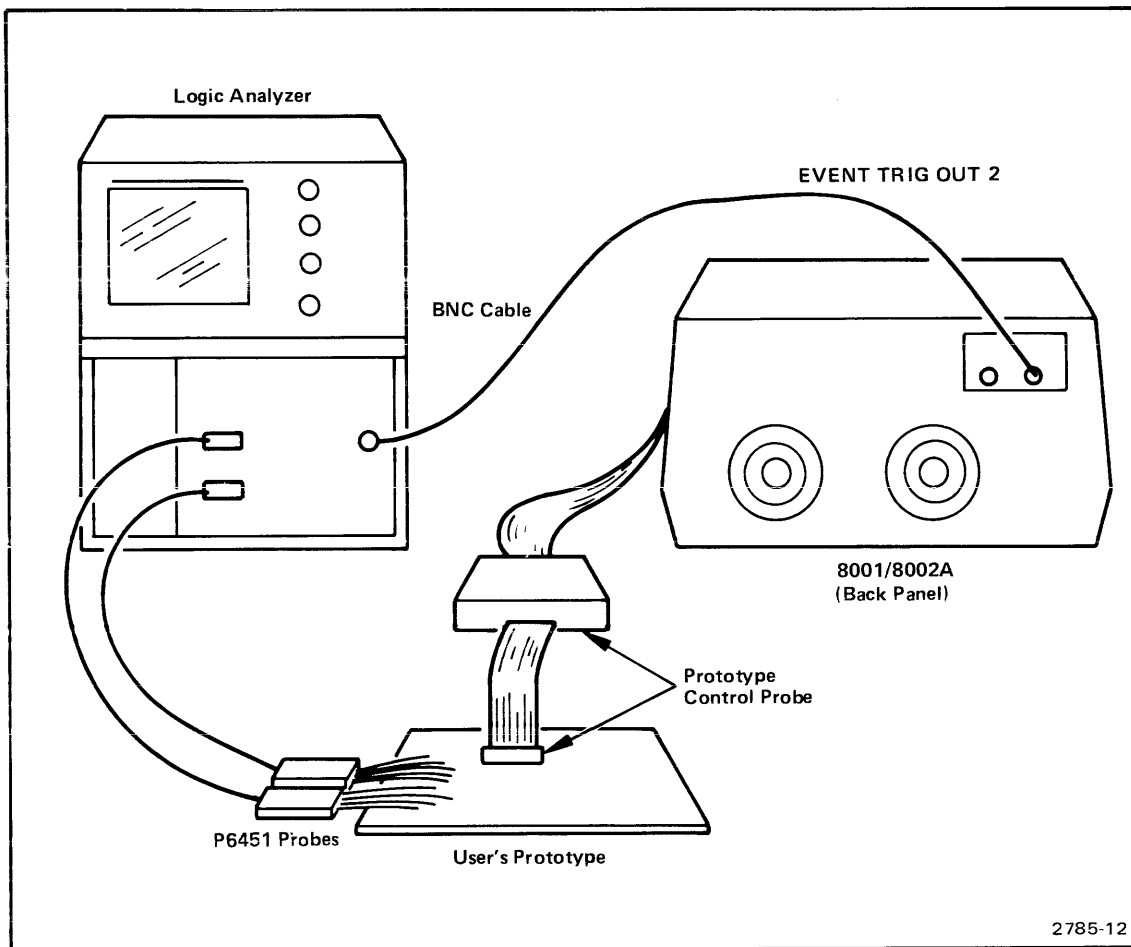


Fig. 5-3. Typical interconnection showing the RTPA triggering the logic analyzer.

CAUTION

When using the logic analyzer with the RTPA and prototype system, all units must be properly grounded to eliminate ground loops and reduce the equipments susceptibility to static discharges. See Appendix C for proper grounding instructions.

As stated previously, some of the features contained in the RTPA are also available in the logic analyzer. To refresh your memory, these features are listed as follows:

- **Storage buffer.** The logic analyzer captures and stores up to 16 bits of data from the two P6451 Data Acquisition Probes. The data is stored in one of three formats: 4 channels at 1016 bits/channel; 8 channels at 508 bits/channel; or 16 channels at 254 bits/channel.
- **Triggering Sources.** The logic analyzer can be triggered either internally or externally. Internal triggering is available in either synchronous or asynchronous mode. In asynchronous mode, the logic analyzer uses the variable internal clock as its triggering source; in synchronous mode, the logic analyzer uses the clock and qualifier channels from the P6451 probes. A word recognizer is available in either synchronous or asynchronous mode. An external trigger source, such as the RTPA EVT1 or EVT2 trigger, can also be used to trigger the logic analyzer.
- **Variable data window.** The data window in the logic analyzer can be positioned around the trigger that corresponds to a desired event. Pre-, center, or post-triggering may be selected. The data within the window can be analyzed by means of a moveable cursor.

You may ask yourself: why are both the RTPA and logic analyzer required to debug a microprocessor-based system, when they both contain similar features? The RTPA, as previously stated, is used primarily in the software development phases; its hardware debugging capabilities are limited to the 8-bit data probe. The logic analyzer is used primarily throughout the hardware design. (With some limitations, the logic analyzer can also be used in software debugging.) When the RTPA and logic analyzer are teamed together in the final integration phase, they act as a very powerful software and hardware debugging tool. The flexibility of the RTPA under the control of the 8001/8002A system terminal can provide the following:

- External triggering of the logic analyzer. The triggers can be positioned to any bus transaction in the program.
- Generation of program breakpoints.
- Comparison of the data obtained from the logic analyzer displays with the RTPA displays (such as the RTT buffer contents).
- Moving the prototype program between the prototype memory and the known environment of the program memory.
- Timing and counting measurements between any two program instructions.

Logic Analyzer Demonstration No. 1

This logic analyzer demonstration defines the following procedures:

- Interconnecting the RTPA, the logic analyzer, and the prototype.
- Setting up the 7D01 Logic Analyzer front panel controls.
- Entering the RTPA commands at the 8001/8002A system terminal.
- Interrupting the logic analyzer displays.
- Comparing the logic analyzer displays and the RTPA displays.

Connections Between the RTPA, Logic Analyzer, and Prototype

Refer back to Fig. 5-3. To interconnect the RTPA, the logic analyzer, and the prototype, perform the following steps:

1. Connect the prototype control probe between the 8001/8002A and the prototype system. See Appendix C for the installation procedures.
2. Connect a cable with BNC connectors on each end from the 7D01's EXTERNAL TRIG INPUT connector to the desired EVENT TRIG OUT connector on the 8001/8001A back panel. Whether you connect this cable to the EVENT TRIG OUT 1 or EVENT TRIG OUT 2 connector will depend on whether you want the logic analyzer triggered on the EVT1 or the EVT2 breakpoint.
3. Connect the test clips of the two P6451 Data Acquisition Probes, as follows:

Probe Channels 0—7

Channels 0—7. Connect to the 8-bit prototype system data bus, lines D0—D7.

Clock Channel. Connect to the prototype system clock (Φ_2).

Ground. Connect to the prototype logic ground.

Probe Channels 8—15

Channels 8—15. Connect to the lower eight bits of the prototype system address bus, lines A0—A7.

Qualifier Clock Channel. Connect to the prototype system clock (Φ_2).

Ground. Connect to the prototype logic ground.



NOTE

The test clip leads attached to the probe head of the P6451 are color-coded in accordance with EIA Color Coding. Each channel is color-coded as follows:

Color	CH 0—7 Probe	CH 8—15 Probe
Black	CH 0	CH 8
Brown	CH 1	CH 9
Red	CH 2	CH 10
Orange	CH 3	CH 11
Yellow	CH 4	CH 12
Green	CH 5	CH 13
Blue	CH 6	CH 14
Violet	CH 7	CH 15
White	Ground	Ground
Gray	Clock	Qualifier

Set-Up Procedures for the Logic Analyzer

Set the front panel controls on the 7D01 Logic Analyzer as follows:

SAMPLE INTERVAL	EXT
RECORD-DISPLAY TIME	∞ (full clockwise detent)
DATA CHANNELS	0—15
TRIGGER SOURCE	EXT
EXT TRIG POLARITY	
THRESHOLD VOLTAGE	TTL (+1.4 V)
EXT CLOCK POLARITY	
WORD RECOGNIZER	CH 0—CH 15 to X (don't care)
EXTERNAL QUALIFIER	1
PROBE QUALIFIER	X (don't care)
W.R. MODE	SYNC

Then make the following initial power-up adjustments to the 7D01 Logic Analyzer:

- Turn the oscilloscope power to ON. (Pull the POWER switch)
- Press the red MANUAL TRIGGER push button (above the EXT TRIG POLARITY switch).
- Adjust the oscilloscope Intensity and Focus and the 7D01 VERT and HORIZ controls for a pattern similar to Fig. 5-4.
- Adjust the CURSOR (FINE and COURSE) controls for a cursor-to-trigger reading of —16 (minus 16).
- Press the RECORD-MANUAL RESET push button. The display shown in Fig. 5-4 should disappear

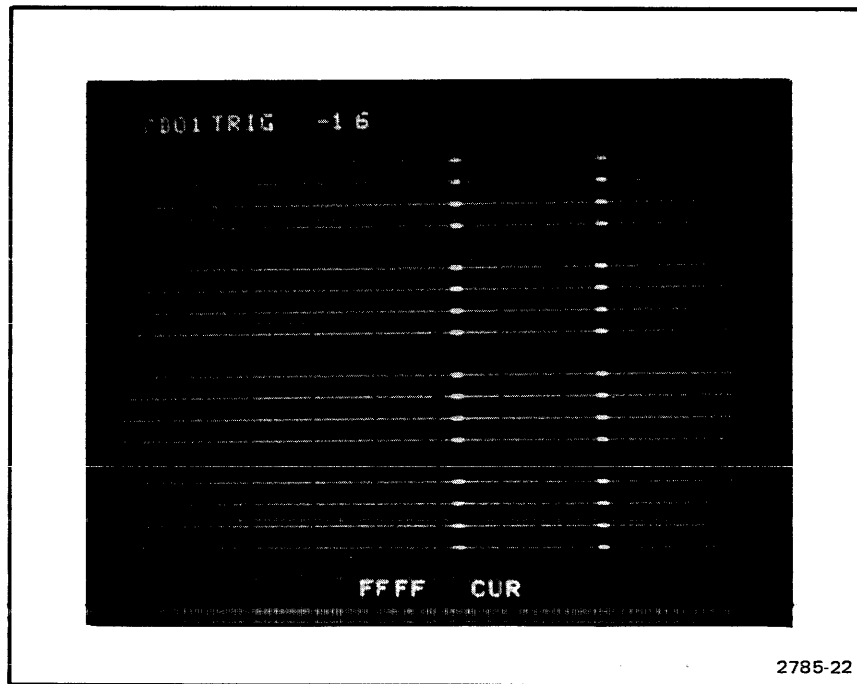


Fig. 5-4. Initial display for adjusting oscilloscope controls.

The DF2 Display Formatter is in the Timing Diagram mode from initial power-on. When the oscilloscope mainframe power is ON, the TIMING DIAGRAM push button should be lit. The DF2 also power-ups with the cursor word in binary.

Make the following adjustments to the DF2 Display Formatter:

- Press the STATE TABLE HEX push button, and then the TIMING DIAGRAM push button (in that order), for a hexadecimal cursor word reading at the bottom of the display.

Set-up Procedures for the 8001/8002A

This demonstration uses the sample program contained in Appendix D. Load the sample program in the program memory. Map and move the program into prototype memory locations 0200—03FF. Set a program breakpoint for EVT2 at address 0300. Execute the program from its starting address (0200) to insure that the program can be executed in the prototype memory.

Example:

```

ENTER:      > LO RTPA4
DISPLAY:    TRANSFER ADDRESS: 0200
            *LOAD* EOJ

ENTER:      > EM 1
DISPLAY:    *EMULATE* EOJ

ENTER:      > MA U 200-3FF

            > MOV PU 200 3FF 200
DISPLAY:    *MOVE* EOJ

ENTER:      > DEB

            > BIF CLR

            > BIF 2

            > EV 2 A=300

            > G 200

DISPLAY:    LOC  INST  MNEM  OPER      SP    RF  RA  RB  RC  RD  RE  RH  RL
            0300  00    NOP           0000  56  25  03  25  02  75  03  25
            0300  BREAK

            >

```

NOTE

When this program is executed, the EVT2 trigger may have triggered the logic analyzer. Before proceeding to the next step in this demonstration, press the RECORD-MANUAL RESET push button on the logic analyzer to remove the display.

Triggering the Logic Analyzer With the RTPA

Once the set-up procedures for the 7D01/DF2 Logic Analyzer and the 8001/8002A have been completed, you are ready to trigger the logic analyzer with the EVT2 trigger from the RTPA.

Timing Diagram Display. Execute the program again from its starting address (0200). When the program reaches the EVT2 address (0300), the EVT2 trigger will stop program execution. This same trigger will trigger the logic analyzer. Note that the green TRIG'D light (to the left of the red MANUAL TRIGGER push button on the 7D01) is lit, and that a display is present on the oscilloscope. Adjust the HORIZ (POS and MAG) controls to obtain a display pattern similar to Fig. 5-5. The intensity control on the oscilloscope may require readjusting.

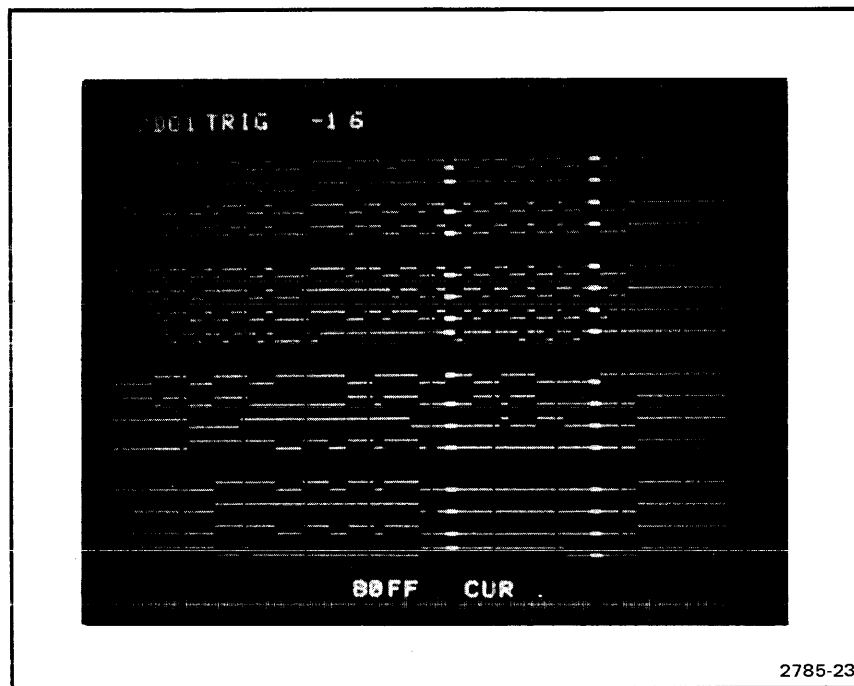


Fig. 5-5. Timing diagram of data bus and lower 8 bits of the address bus.

Fig. 5-5 is a timing diagram, showing the relationship between the 8-bit prototype data bus and the lower eight bits of the prototype address bus. The top line in the display corresponds to D0, the eighth line corresponds to D7; the ninth line to A0, and the sixteenth line to A7. The readout of "—16" intervals, at the top of the display in Fig. 5-5, represents the distance between the cursor and the trigger in clock intervals. The prototype clock is used as the clock source; since each interval is equal to one clock period, the distance between the cursor and the trigger is 16 clock periods. In the 8080A microprocessor, each clock period is equal to a "state"; three to five states constitute a machine cycle; and one to five machine cycles compose an instruction cycle. The display in Fig. 5-5 is thus showing the lower eight address bits and the eight data bits for each state.

As the CURSOR FINE POS control is rotated, the cursor word at the bottom of the display is changed; each state in the instruction can be read in hexadecimal values. The two leftmost digits are the value of the lower address bits, and the two rightmost digits are the value of the data bits.

NOTE

The DF2 displays the cursor word in binary values when it is powered up. To display the cursor word in hexadecimal or octal values press the HEX or OCTAL push button, and then press the TIMING DIAGRAM push button. Each time you press the TIMING DIAGRAM push button the cursor word readout alternates between hexadecimal or octal (whichever one was pressed last) and binary.

State Table Display. Fig. 5-6 is a state table display provided by the DF2 for each of the 16 states between the cursor and the trigger shown in Fig. 5-5. Press the DF2 STATE TABLE HEX push button to obtain the display shown in Fig. 5-6. (Push buttons HEX, 7D01 ONLY and LOGIC POS should become lit when the HEX push button is pressed.)

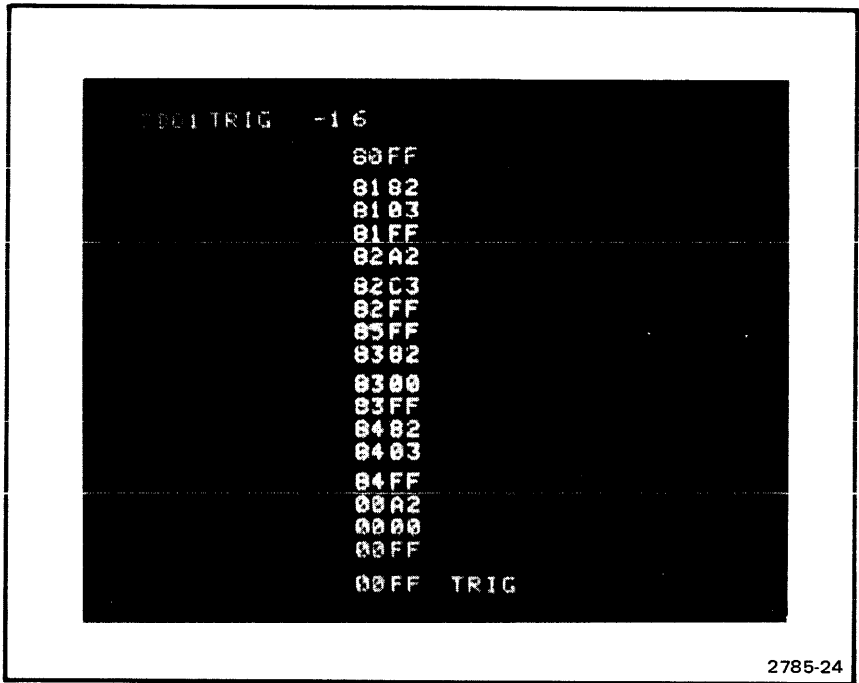


Fig. 5-6. State table display of the 16 states between the cursor and trigger.

The two leftmost columns in the display are the hexadecimal value of the lower eight address bits; the two rightmost columns are the value of the eight data bits. The trigger word is shown at the bottom of the display, and the cursor word is always shown as the uppermost data word in the display, directly under the cursor-to-trigger readout. In Fig. 5-6, the trigger word is 00FF, and the cursor word is 80FF. The cursor word is also shown at the bottom of Fig. 5-5. In the display, the trigger word is continually blinking, providing a means of identification.

In the set-up procedures, the program breakpoint was set for address 0300. This address contains a data byte of "00". The lower eight address bits and the data bits for this breakpoint address, form the word "0000". This value is shown in Fig. 5-6, one state above the trigger word. Recall that the EVT2 trigger, which causes the breakpoint, occurs when the program instruction is completed. Thus, the EVT2 trigger from the RTPA, triggers the logic analyzer on the next clock period after the EVT2 breakpoint address.

Fig. 5-6 shows the states for the program addresses 0381, 0382, 0383, 0384, and 0300. Address 0381 is the last address of the multi-byte JNZ instruction. Addresses 0382—0384 contain the complete jump (JMP) instruction. We'll use this instruction to analyze the various states represented in Fig. 5-6.

The JMP instruction consists of ten states, which constitute three machine cycles. The eight data bits on the data bus for each state make up the status word, which defines the type of machine cycle or data byte.

The first machine cycle in any instruction cycle is always a FETCH cycle. This cycle is identified by the 8-bit status word "A2", during the first state period. The FETCH cycle consists of four states. The first state of this FETCH cycle is shown in Fig. 5-6 as the word "82A2". "82" is the address 0382, and "A2" indicates the status word FETCH. During the second state of the FETCH cycle, the processor fetches the instruction indicated by its program counter. This is shown in Fig. 5-6 as the next word "82C3." "82" is again the address 0382, and "C3" is the JMP instruction.

During the third and fourth states the data bus is floating. This is indicated by the "FF" data byte shown in Fig. 5-6 for the next two states, "82FF" and "85FF". Note that in the fourth state the address byte has changed from 82 to 85. The information on the address bus is not valid after the third state of an instruction.

The first state of the next machine cycle is shown in Fig. 5-6 as "8382". "83" represents the next program address (0383), and "82" is the value of the status word; this word indicates a MEMORY READ machine cycle. The MEMORY READ machine cycle consists of three states. During this machine cycle, the first data byte indicated by the program counter is read from memory and placed in the processor's internal Z register; the program counter is then incremented. This is shown in Fig. 5-6 by the second and third states, "8300" and "83FF".

The next machine cycle is another MEMORY READ machine cycle. This cycle functions similar to the first MEMORY READ machine cycle, except that the data byte is this time placed in the processor's internal W register. The program counter is incremented in anticipation of the next instruction FETCH. This cycle is displayed in Fig. 5-6 as "8482", "8403" and "84FF".

You may also dump the last five lines from the RTT buffer and compare the lower eight address bits and the data bits with the logic analyzer display shown in Fig. 5-6. You must remember that the RTT buffer displays the data byte for each machine cycle, whereas the Fig. 5-6 display shows the individual states in the machine cycle.

Now that you've completed this demonstration, experiment with other program breakpoints and analyze the logic analyzer displays. As you experiment with various breakpoints, you'll develop a high degree of confidence in directing the RTPA to trigger the 7D01 and display the data before, around, and after any designated RTPA event parameters.

This confidence in the RTPA's triggering capabilities is necessary when you want to use the logic analyzer to look within the prototype hardware circuitry. There, the only relationships between the RTPA displays and the logic analyzer displays are the EVT triggers. The logic analyzer and RTT buffer displays are synchronized by the EVT triggers; however, there is no correlation between the data in the displays, as there was in Fig. 5-6 and the RTT buffer display.

NOTE

Remember that when new data is to be displayed on the logic analyzer, you must press the RECORD-MANUAL RESET pushbutton before the logic analyzer can accept the new data. The RESET pushbutton should be pressed before executing a program.

In this demonstration, we've seen how the prototype address bus and data bus can be displayed on the logic analyzer and in the RTPA. We've learned how to correlate these displays. In the following demonstration, the relationship between the address bus and data bus in the displays no longer exists, only the EVT trigger relationship is maintained.

Logic Analyzer Demonstration No. 2

This demonstration builds on the knowledge you have gained in the previous demonstration. The troubleshooting capabilities of the logic analyzer are carried further into the microprocessor control circuitry. You will see how the timing diagrams of the logic analyzer can be used to determine if the microprocessor's control signals are in correct sequence, and whether the timing relationships between the signals are correct.

Connections Between the RTPA, Logic Analyzer, and Prototype

The interconnections between the RTPA, the logic analyzer, and the prototype hardware remain functionally the same as in the previous demonstration. However, the following connections must be changed:

1. Connect the cable with BNC connectors on each end to the EVENT TRIG OUT 1 connector on the 8001/8002A back panel. This connection permits the logic analyzer to be triggered by the EVT1 trigger.

2. Connect the test clips of the two P6451 Data Acquisition Probes as follows:

Probe Channels 8—15

Channels 8—15. In the previous demonstration the data bus in the prototype system is connected to channels 0—7 in the logic analyzer. Reverse both front panel connectors on the logic analyzer. This connects the data bus, lines D0—D7, to channels 8—15 in the logic analyzer.

Qualifier Clock Channel. Connect to the prototype system clock (Φ_2).

Ground. Connect to the prototype logic ground.

Probe Channels 0—7

Channel 0. Connect to the prototype system clock (Φ_1).

Channel 1. Connect to the prototype system clock (Φ_2).

Channel 2. Connect to the SYNC timing output line of the microprocessor.

Channel 3. Connect to the DBIN timing output line of the microprocessor.

Channel 4—7. Not connected.

Clock Channel. Connect to the prototype system clock (Φ_2).

Ground. Connect to the prototype logic ground.

Set-up Procedures for the Logic Analyzer

Retain the same front panel control settings on the 7D01 Logic Analyzer from the previous demonstration, with the following exceptions:

SAMPLE INTERVAL	.1 μ sec
DATA POSITION	CENTER
W.R. MODE	ASync

Repeat the power-up adjustments to the 7D01 Logic Analyzer and DF2 Display Formatter, as listed in the previous demonstration.

Set-up Procedures for the 8001/8002A μ Processor Lab

The sample program contained in Appendix D is also used in this demonstration. If you have the sample program, already loaded in prototype memory, enter the following commands:

Example:

```
ENTER:      > EV 1 A=37D
            > EV 2 A=300
            > BIF ARM
            > EV
DISPLAY:    EVT1 A=037D B=ALL
            EVT2 A=0300 B=ALL
            BIF-ARM S
            >
```

If the sample program is not loaded in the prototype's memory, perform the following steps:

- Load the sample program.
- Invoke emulation Mode 1.
- Map and move the program between addresses 0200—03FF to the prototype's memory.
- Invoke the DEBUG command.
- Then enter the commands listed in the preceding paragraph.

Triggering the Logic Analyzer With the RTPA

Once you've completed the set-up procedures for the logic analyzer and the 8001/8002A, you are ready to trigger the logic analyzer with the EVT1 trigger from the RTPA. Note that in this demonstration, we are using the EVT1 trigger and the BIF ARM mode. This means that the program will not stop when the logic analyzer is triggered by the EVT1 trigger; it will continue to run until it is stopped at the EVT2 address. The logic analyzer is in the center trigger position. Data will be present before and after the trigger; the EVT1 parameters will appear in the center of the display.

Timing Display No. 1. Execute the program from its starting address, 0200. When the program reaches the EVT1 address (037D), the EVT1 trigger will trigger the logic analyzer. The program will continue to run until the EVT2 address (0300) is reached.

Adjust the 7D01 HORIZ and VERT position (POS) and magnitude (MAG) controls, for a display pattern similar to Fig. 5-7.

The SAMPLE INTERVAL switch on the logic analyzer may require some adjustments, depending on the speed of your prototype clock and the type of emulator processor you're using. Increase or decrease the SAMPLE INTERVAL switch until you obtain a pattern similar to Fig. 5-7. Remember that each time the switch is changed, the RECORD-MANUAL RESET push button on the logic analyzer must be pressed and the program executed again from its starting address.

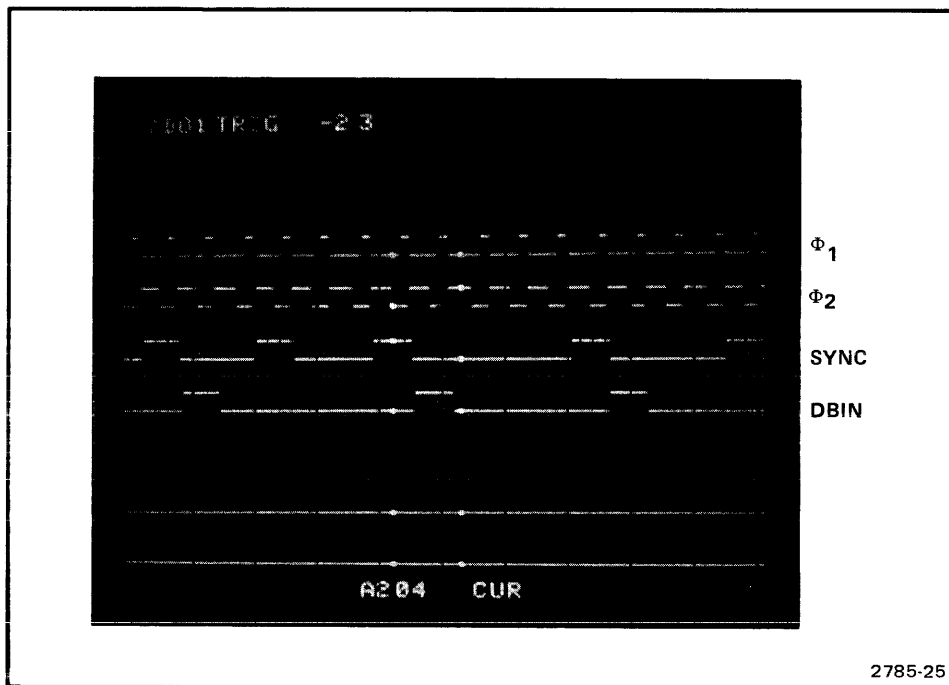


Fig. 5-7. Timing diagram showing relationships between timing lines from the microprocessor.

Fig. 5-7 is a timing diagram of the first four channels of the logic analyzer. This diagram shows the relationship between the Φ_1 and Φ_2 clocks, and the SYNC and DBIN timing lines. This timing diagram is similar to the timing diagrams in any 8080A microprocessor data book. Similar timing diagrams can be obtained for any of the microprocessors supporting the 8001/8002A μ Processor Lab. You can also add timing and control lines to channels 4—7; however, in this demonstration, only four timing lines are displayed.

The SYNC pulse (third line from the top) identifies the first state (T_1) in every machine cycle. During the SYNC pulse, the data bus contents indicate the type of machine cycle. The DBIN pulse (fourth line from the top) is present during the T_2 state of every FETCH, MEMORY READ, STACK READ, and INTERRUPT machine cycle. During the DBIN pulse, the contents on the data bus identifies the program instruction.

In Fig. 5-7 even though the data bus lines (channels 8—15) are not displayed, the two leftmost digits in the cursor word (at the bottom of the display) represents the hexadecimal value of the data bus.

Rotate the CURSOR control on the logic analyzer counterclockwise until the cursor is in the middle of the SYNC pulse, as shown in Fig. 5-7. The data bus reading of "A2" (instruction FETCH) indicates that this SYNC pulse is not only the start of a machine cycle but also the start of an instruction cycle.

NOTE

The timing diagram in Fig. 5-7 shows a cursor-to-trigger readout of -23 (from the middle of the SYNC pulse). The cursor-to-trigger reading may be different if you are using a different microprocessor, another program, or if the SAMPLE INTERVAL switch is in another setting. However, if you are using an 8080A microprocessor, the relationship between the Φ_1 , Φ_2 , SYNC, and DBIN signals should be similar.

Rotate the CURSOR control clockwise, until the cursor is in the middle of the DBIN pulse, as shown in Fig. 5-8. The data bus reading of 7D represents the program instruction MOV A,L. This is the data byte from the program address 037D. This address was entered into the EVT1 command line at the beginning of this demonstration.

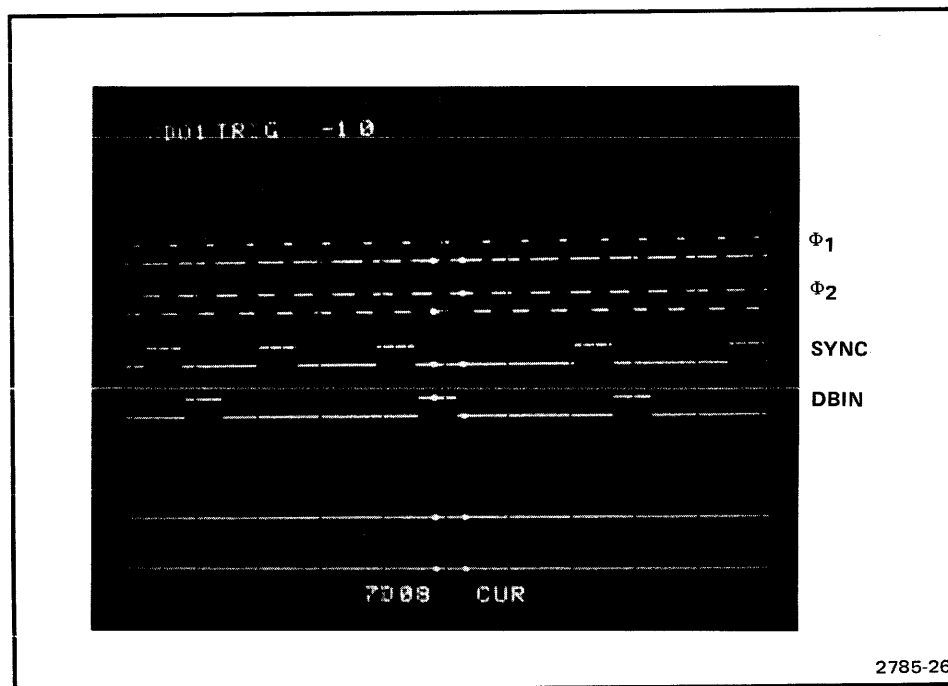


Fig. 5-8. Timing diagram showing cursor and cursor word on the data byte of 7D. This is the second state of the program instructions MOV A,L. (See text.)

The program instruction MOV A,L has one machine cycle and five states. As the CURSOR control is rotated further clockwise, the data bytes of the other three states can be read. All three are FF. This indicates that the data bus is floating during these states. The positive-to-negative edge of the Φ_2 clock pulse provides the best reading of the various states.

Continue to rotate the CURSOR control clockwise and observe the next program instruction, a CMP instruction. This instruction has one machine cycle and four states. The data bus readings should be A2, B9, FF, and FF.

Timing Display No. 2. Now let's examine an instruction that has more than one machine cycle. The MVI M,O instruction at program address 037B has three machine cycles and ten states. Since this instruction occupies over half of the display, you'll get a better display if you specify the address (037C), the second data byte, for the EVT1 trigger. This shifts the data display on the scope one machine cycle to the left of center trigger, and allows the complete instruction to be seen.

If you have not performed Timing Display No. 1, first go back to the beginning of that demonstration and complete the set-up procedures for the logic analyzer and the 8001/8002A.

To change the EVT1 trigger, enter the following command.

```
Example:
ENTER:    > EV 1 A=37C
          >
```

Press the RECORD-MANUAL RESET push button and execute the program from its starting address 0200.

Adjust the HORIZ and VERT position (POS) and magnitude (MAG) controls, to obtain a display pattern similar to Fig. 5-9. Rotate the CURSOR controls to the start of the instruction cycle (a FETCH, hexadecimal value of A2). Table 5-2 lists the data bytes for each T state as the CURSOR controls are rotated clockwise.

Note that in Fig. 5-9, the SYNC pulse for the last machine cycle in this instruction does not have a corresponding DBIN pulse. As previously stated, the DBIN pulse is present only in a FETCH, MEMORY READ, STACK READ or INTERRUPT machine cycle.

As mentioned earlier in this section, a wait state is added to each machine cycle when operating in Mode 1 and the program is executing in the program memory.

This wait state (T_w) is added between states T_2 and T_3 of each machine cycle. However, when the program is executing in the prototype's memory, the wait state is not added. In Fig. 5-9, wait states are not added.

Table 5-2
Breakdown of T States for Program Instruction MVI M,0

	T State	Data Byte
Designates FETCH machine cycle.	T ₁	A2
	T ₂	36
	T ₃	FF
	T ₄	FF
Designates MEMORY READ machine cycle.	T ₁	82
	T ₂	00
	T ₃	FF
Designates MEMORY WRITE machine cycle.	T ₁	00
	T ₂	00
	T ₃	00

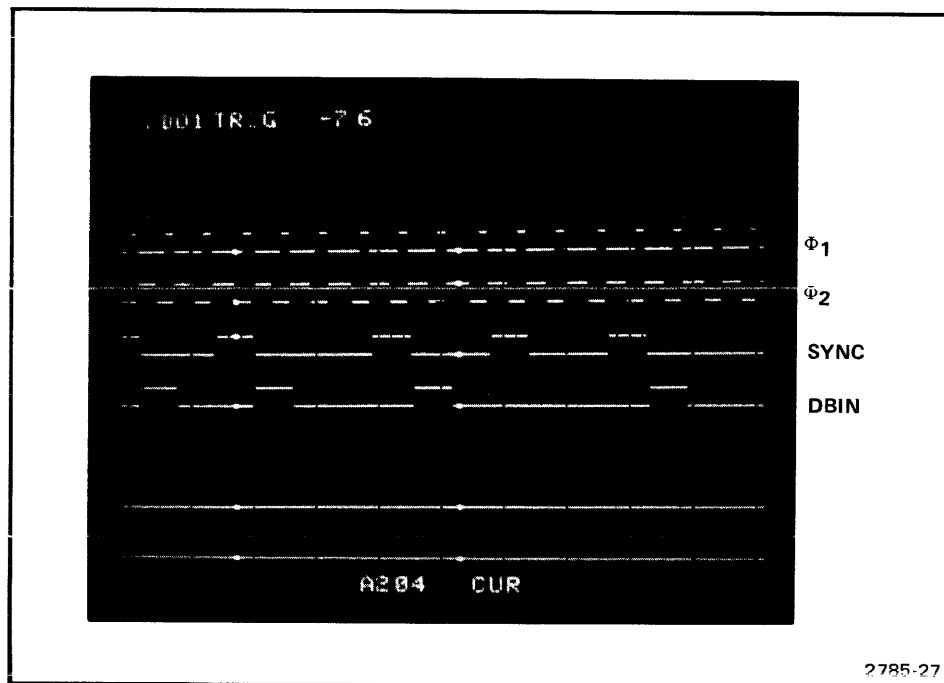


Fig. 5-9. Timing diagram showing the cursor and cursor word at the start of the instruction cycle MVI M,0. This is a two byte instruction with three machine cycles and 10 states.

To observe how wait states are added, map the program addresses back to the 8001/8002A. Enter the following command:

Example:

```
ENTER:    > MA P 200-3FF
          >
```

Now press the RECORD-MANUAL RESET push button and execute the program from its starting address, 0200.

Adjust the HORIZ MAG control to obtain a display pattern similar to Fig. 5-10. Rotate the CURSOR controls to the start of the instruction cycle. Note that the DBIN pulses are twice as wide as they were in Fig. 5-9. Rotate the CURSOR controls through the complete instruction cycle, and observe the data byte reading at each T state. Observe for each machine cycle that the T_2 data byte reading (from Table 5-2) also appears in the next T state. This verifies that the wait state (T_w) is added between states T_2 and T_3 . The total number of T states in the instruction cycle is now 13, instead of 10, this indicates that a wait state is added to each machine cycle.

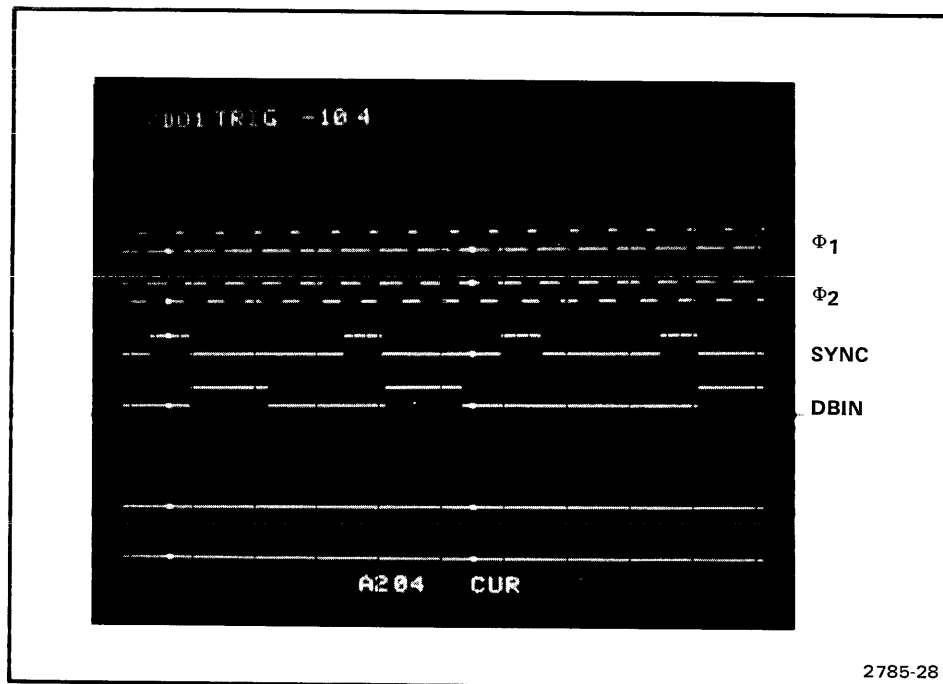


Fig. 5-10. Timing diagram showing the added wait state to each machine cycle when the program runs in the program memory in emulation Mode 1.

Now map the program addresses back to the prototype memory, by entering the following command:

Example:

```
ENTER:    > MA U 200-3FF
          >
```

Timing Display No. 3. The four timing lines in Fig. 5-9 and 5-10 can be further analyzed by making the following changes to the 7D01 front panel controls:

SAMPLE INTERVAL	10 nsec
DATA CHANNELS	0—3

Press the RECORD-MANUAL RESET push button and execute the program from its starting address, 0200. Adjust the HORIZ and VERT position (POS) and magnitude (MAG) controls for a display pattern similar to Fig. 5-11.

Time synchronization measurements can be made by rotating the CURSOR controls between the various points of interest in the display. Each step of the FINE CURSOR control is equal to 10 nsec. You can measure timing delays between the falling and rising edges of pulses; pulse width measurements; and data set-up intervals.

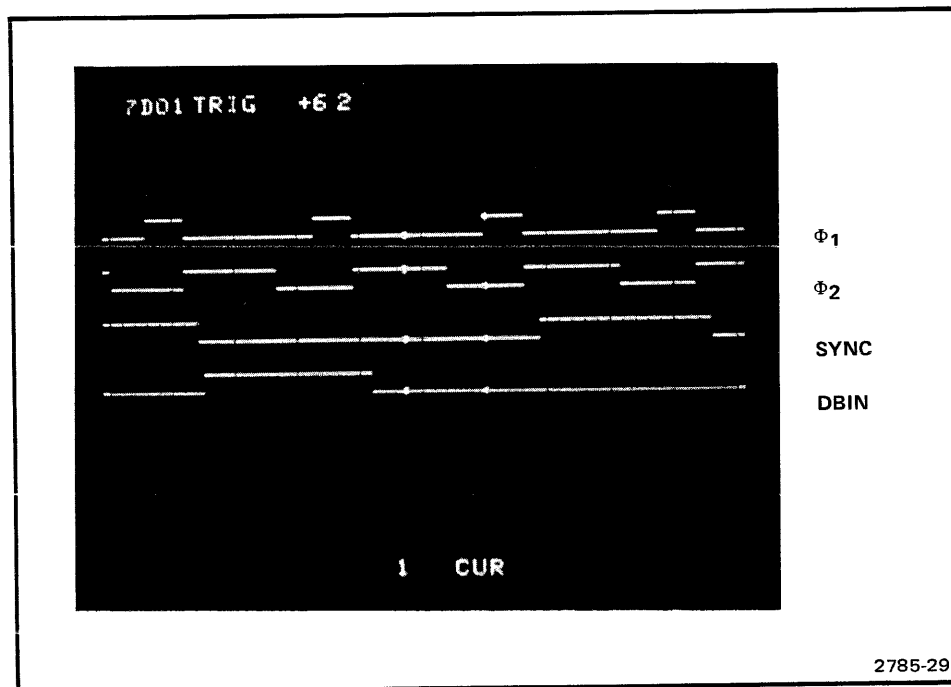


Fig. 5-11. An expanded timing diagram of channels 0—3. Each step of the FINE CURSOR control is equal to 10 nsec permitting timing measurements within the display.

SUMMARY

In these demonstrations, we've attempted to acquaint you with the basic capabilities and possibilities of the RTPA and logic analyzer. We realize that these demonstrations have uncovered only the tip of the iceberg. We hope that your knowledge of the RTPA, how it works, and how to use the various commands, will now allow you to use this tool effectively in your everyday work. We anticipate that your skill with the RTPA coupled with your understanding of microprocessor-based systems, will stimulate your imagination towards creative applications and future uses of the RTPA in your own work environment.

As previously stated, the RTPA coupled with the logic analyzer is a powerful debugging tool, with many applications in the software/hardware design and integration phases of a microprocessor-based system. When you use a tool with these capabilities, there is always the possibility of ending up in a stalemate, or an "operator trap." It is only fair to warn you that you'll encounter "operator traps" quite frequently at first. However, as you become more proficient with the RTPA commands and the logic analyzer, these traps will gradually diminish.

To this point, we have tried to give you all the necessary steps, examples and notes so that you could perform the procedures and demonstrations. From now on, you are on your own. If the RTPA helps you to find software and hardware bugs and is making your job easier, then these demonstrations have succeeded.

Wishing you smooth RTPAing and many eventful EVT's.

Section 6

GLOSSARY

This glossary defines the terms used in this manual that are associated with RTPA operations in the 8001/8002A system.

ARM	An EVT or BIF command mode. ARM is a sequential mode: the EVT1 trigger arms or enables the EVT2 comparator. The EVT2 trigger then arms or enables the EVT1 comparator.
BIF	The acronym for the RTPA command — Break if. BIF sets or clears the event break options. Program execution is broken if the EVT options match the RTPA input data and an EVT trigger is generated.
Breakpoint	A point at which program execution is suspended. The BIF command sets the break option and generates a breakpoint, which stops program execution.
Bus operations	Refers to the three control signal lines to the RTPA: M/IO, R/W, and FETCH.
CNT	The acronym for the RTPA command — Count. The CNT command is used to set the count units for the general purpose counter and delay units for the delay counter.
Command	The initial portion of a command line, usually a symbol or word specifying an action or a definition for the system.
Command line	A command and its parameters.
Comparison options	Those EVT options (options A, D, B, and T) that are entered in the EVT comparison files and compared to the input data.
Counting options	Those EVT options (options P and C) that are entered in the pass and delay counters. When specified, counting options delay the comparison options.
Data Acquisition Probe	An active nine-channel probe. Eight channels are used for data acquisition. The ninth channel is used for a clock signal. The probe head contains leads with test clips attached for connecting to a data source and associated clock.

Debugging	The search for sources of logic error in a program or hardware. The RTPA commands (BIF, CNT, DRT, EVT, and RTT) and system commands (BKPT, CLBP, DISM, DSTAT, GO, SET, and TRACE) facilitate debugging.
Delay counter	Delays the EVT trigger by the number of units designated in the EVT option C parameter. The delay units are specified with the CNT (count) command.
DRT	The acronym for the RTPA command — Display real-time trace. The DRT command displays the RTT (real-time trace) buffer contents.
Emulation	Various techniques where one system acts like another system. With the 8001/8002A, emulation refers to the ability of the 8001/8002A system to execute software programs written for the prototype system. Emulation minimizes the impact of conversion from one system to another.
Emulation Mode 0	One of three modes of emulator processor operation. The prototype program executes entirely within the 8001/8002A program memory. Program execution is controlled by the 8001/8002A system clock.
Emulation Mode 1	One of three modes of emulator processor operation. Any portion of the prototype program may be executed in either the 8001/8002A program memory or in the prototype memory. Program execution is controlled by the prototype clock.
Emulation Mode 2	One of three modes of emulator processor operation. The prototype program executes entirely within the prototype memory, with overall control being retained in the 8001/8002A system. Program execution is controlled by the prototype clock.
Emulator clocks	A CNT command option (CN C) that counts the emulator clock cycles during the program run time. This CNT option also sets emulator clock cycles as the delay units for the EVT option C command.
Emulator processor	The hardware module that permits program execution and debugging; the emulator processor uses the same microprocessor type as in the prototype. With the prototype control probe attached in emulation Mode 1 or 2, the emulator processor emulates the microprocessor in the prototype under development. A program can then drive the prototype hardware under the supervision of the 8001/8002A system.

Event comparators	Hardware within the RTPA that compares the conditions set with the EVT option commands to the data entering the RTPA during program execution.
EVT	The acronym for the RTPA command — Event. The EVT command sets or displays the EVT comparison and counting options. There are two EVT command lines, EVT1 and EVT2.
Execution time	The actual time required to run a specific software program in a specific microprocessor. The RTPA can measure execution time in microseconds or in milliseconds. Adding wait states to a microprocessor operation lengthens the program execution time.
Freeze	When the RTT buffer freezes, it stops storing data. One of the EVT triggers freezes the RTT buffer; the specific trigger is dependent on the BIF command specified.
FRZ	A BIF command mode. FRZ is a sequential mode, similar to ARM, except that in FRZ mode, the RTT buffer is frozen with the EVT1 trigger.
Hardware	The physical components of the 8001/8002A or prototype system.
Hardware-generated triggers	The RTPA hardware can generate two triggers, EVT1 and EVT2. The generation and relationship of these triggers is established with the BIF or EVT command mode.
IND	An EVT or BIF command mode. In this mode, the EVT1 and EVT2 triggers are independent; that is, the trigger is generated when the EVT comparison and counting options are satisfied for each command line.
LIM	An EVT or BIF command mode. LIM is a conditional mode: EVT1 options A, D, T, and B, plus EVT2 options A and B, must simultaneously match the data entering the RTPA before an EVT1 trigger can be generated.
Pass counter	Delays the EVT trigger by a number of EVT comparison occurrences; that number is designated with the EVT option P parameter.
Program memory	The memory area in the 8001/8002A where programs are stored and executed in emulation Mode 0.
Prototype	The actual microprocessor-based software or hardware system under design development.

Prototype Control Probe	A probe that links the user prototype via an interconnecting cable to the active 8001/8002A emulator processor module. The probe enables the emulator processor to act in place of the prototype microprocessor. With the probe installed, programs and prototype hardware can be debugged under 8001/8002A control. Each emulator processor type requires its own corresponding probe.
Prototype memory	The memory area in the prototype device where the executing program is stored during emulation Mode 2 or Mode 1 (if the program is mapped and moved to the prototype).
Prototype program	The actual microprocessor-based software program under development.
RTPA	The acronym for the Real-Time Prototype Analyzer. Refer to Section 1 of this manual for an explanation of the RTPA.
RTT	The acronym for the RTPA command — Real-Time Trace buffer. Selects the type of bus transaction that is stored in the RTT buffer.
RTT buffer	Stores the data entering the RTPA during program execution. Up to 128 selected bus transactions can be stored.
System terminal	The console connected to the 8001/8002A; used for entering system commands or information, and for monitoring program execution.
TEKDOS	Tektronix Disc Operating System; the operating system of the 8002A μ Processor Lab. TEKDOS performs text editing, controls and monitors prototype program execution, loads data from external sources, and provides system utility functions.
TEKOPS	Tektronix Operating System; the operating system of the 8001 μ Processor Lab. TEKOPS loads data from external sources, controls and monitors prototype program execution, and provides system utility functions.
True comparison	A match between EVT comparison options (entered in the EVT comparison files) and the data entering the RTPA during program execution. A true comparison enables the EVT counting options (if set) or generates an EVT trigger.
User memory	Prototype memory; can be accessed in emulation Modes 1 and 2.

Appendix A EMULATOR CHARACTERISTICS

	Page
Introduction	A-1
RTPA Timing Considerations	A-1
Addition of Wait States	A-1
Effects on the Program Run Time	A-4
Effects on the Total Emulator Clock Cycles	A-4
Program Calculations	A-5
Model Program	A-5
RTPA Measurements	A-6
Measurements for Mode 0	A-7
Measurements for Mode 1 Using 8001/8002A Program Memory	A-9
Measurements for Modes Using Prototype Memory	A-9
Measurements for Other Emulator Processors	A-9
Measurements for Complex Programs	A-10
Individual Emulator Characteristics	A-10
3870 Emulator Processor	A-11
6800 Emulator Processor	A-13
8080A Emulator Processor	A-15
8085A Emulator Processor	A-17
9900 Emulator Processor	A-19
Z80 Emulator Processor	A-21

TABLES

Table No.		
A-1	8002A μ Processor Lab/TEKDOS Addition of Wait States	A-2
A-2	8001 μ Processor Lab/TEKOPS Addition of Wait States	A-3
A-3	Total Machine and Emulator Clock Cycles for Model Program	A-5
A-4	Comparison of Program Run Times and Emulator Clock Cycles	A-6

Appendix A

EMULATOR CHARACTERISTICS

INTRODUCTION

This appendix describes those characteristics of each emulator processor that are associated with RTPA operations. Each emulator processor has distinctive features, related to the design of the associated microprocessor device. Those features associated with the RTPA's operation are explained first in this appendix. Then, this appendix describes the RTPA-related features of each emulator processor supporting the 8001/8002 systems.

For additional information about any of the emulator processors that support the 8001/8002A, refer to the Emulator Specifics section in 8001 or 8002A μ Processor Lab System User's Manual, or to the appropriate emulator processor service manual.

RTPA TIMING CONSIDERATIONS

Earlier in this manual we discussed how the RTPA can be used to determine the program run time. In Section 5, we discovered that the program timing as measured, with the RTPA commands, changed when the emulator mode was changed from 0 to 1 or 2. Also, in emulation mode 1, when the program was mapped and moved from program memory to the prototype memory, the program run time was not the same. In Section 5, we also mentioned that some emulator processors add wait states to every machine cycle when operating in emulation Modes 0 and 1. The addition of these wait states (T_w) was shown in Fig. 5-10. These wait states have a considerable affect on the program run time as measured with the RTPA.

In our previous encounters with program timing, we used only the 8080A Emulator Processor in all the discussions and demonstrations. The following paragraphs discuss the problems of determining the correct program run times for all the emulators supporting the 8001/8002A systems.

ADDITION OF WAIT STATES

Tables A-1 and A-2, with their associated notes, define the addition of wait states for each emulator processor supporting the 8001/8002A systems. Review these tables and notes carefully before you proceed further in this appendix. They will be referred to continually throughout this appendix.

When the Real-Time Trace module is installed in the 8001/8002A, the figures in both Tables A-1 and A-2 are identical. Wait states are added whenever the Debug TRACE or BKPT is ON. When the Debug TRACE or BKPT commands are OFF, no wait states are added. When the Real-Time Trace module is not installed, the DEBUG command by itself adds wait states. The only exception to this is when the 9900 Emulator Processor is operating in Mode 1 and using the prototype memory.

Table A-1
8002A μ Processor Lab/TEKDOS
Addition of Wait States

8002A μ Processor Lab System Operation Status	Emulator Processor			
	3870 6800 ^a	8080A 8085A	Z80	9900
	Number of wait states added to each machine cycle.			
Real-Time Module NOT Installed in 8002A ^b				
Mode 0— All operations.	0	1	2	1
Mode 1— DEBUG command is ON or program memory is used.	0	1	2	1
DEBUG command is OFF and prototype memory is used.	0	0	0	1
Mode 2— DEBUG command is ON.	0	1	2	1
DEBUG command is OFF.	0	0	0	0
Real-Time Trace Module Installed in 8002A ^c				
Mode 0— All operations.	0	1	2	1
Mode 1— Debug TRACE or BKPT command is ON or program memory is used.	0	1	2	1
Debug TRACE and BKPT commands are OFF and prototype memory is used.	0	0	0	1
Mode 2— Debug TRACE or BKPT command is ON.	0	1	2	1
All other operations.	0	0	0	0

^a Emulator Processors 3870 and 6800 do not add wait states in any emulation mode.

^b The wait states shown are added when the Real-Time Trace module is not installed in the 8002A.

^c The wait states shown are added when the Real-Time Trace module is installed in the 8002A.

Table A-2
8001 μ Processor Lab/TEKOPS
Addition of Wait States

8001 μ Processor Lab System Operation Status	Emulator Processor			
	3870 6800 ^a	8080A 8085A	Z80	9900
	Number of wait states added to each machine cycle.			
Real-Time Trace Module NOT Installed in 8001 ^b				
Mode 0— All operations.	0	1	2	1
Mode 1— All operations.	0	1	2	1
Mode 2— All operations.	0	1	2	1
Real-Time Trace Module Installed in 8001 ^c				
Mode 0— All operations.	0	1	2	1
Mode 1— Debug TRACE or BKPT command is ON or program memory is used.	0	1	2	1
Debug TRACE and BKPT commands are OFF and prototype memory is used.	0	0	0	1
Mode 2— Debug TRACE or BKPT command is ON.	0	1	2	1
All other operations.	0	0	0	0

^a 3870 and 6800 Emulator Processors do not add wait states in any emulation mode.

^b The wait states shown are added when the Real-Time Trace module is not installed in the 8001. The same wait states are added for all emulation modes. This is because the 8001 is always under Debug control.

^c The wait states shown are added when the Real-Time Trace module is installed in the 8001.

Effect on the Program Run Time

From the preceding paragraphs, you can envision how the addition of wait states to each machine cycle will affect RTPA measurements of program run time. These wait states increase the RTPA readings by a measurement of time equal to the sum of the wait states added. The RTPA measurements of program run time can be represented by the following formula:

$$(A.1) \quad \begin{array}{l} \text{Program Run Time as} \\ \text{Measured With the RTPA} \end{array} - \begin{array}{l} \text{Time of Added} \\ \text{Wait States} \end{array} = \begin{array}{l} \text{Actual Program} \\ \text{Run Time} \end{array}$$

In the above formula, the time of the added wait states is dependent on the following variables:

- The type of emulator processor used.
- The emulation mode used.
- The length of the executed program.

Effects on the Total Emulator Clock Cycles

In addition to the effect on program run time, the addition of wait states to each machine cycles has an effect on the total number of emulator clock cycles, when a program is executed. The RTPA measurement of the total number of emulator clock cycles is governed by the following formula:

$$(A.2) \quad \begin{array}{l} \text{Emulator Clock Cycle as} \\ \text{Measured with the RTPA} \end{array} - \begin{array}{l} \text{Number of Wait} \\ \text{States Added} \end{array} = \begin{array}{l} \text{Actual Emulator} \\ \text{Clock Cycles} \end{array}$$

In the above formula, the number of wait states added is dependent on the following variables:

- The type of emulator processor used.
- The emulation mode used.
- The length of the executed program.

PROGRAM CALCULATIONS

To demonstrate how the RTPA measures program run time and emulator clock cycles, we will use a short program as a model. (Note: This is NOT the same sample program used throughout this manual.) We will simulate execution of the model program, calculating the machine cycles, emulator clock cycles, and program run time. We will then load the model program into the 8001/8002A and compare the RTPA measured values to our calculated values.

Model Program

The model program is a simple program that loads a register with "00", increments the register to "FF", and jumps to a halt or quit instruction. (Note: The register could be loaded with "FF" and decremented to "00".) The object file for this model program, in 8080A assembly language, is shown in Table A-3. The number of machine cycles per instruction and the number of emulator clock cycles per instruction were obtained from an 8080A programming manual.

Table A-3
Total Machine and Emulator Clock Cycles for Model Program

Model 8080A Program			Machine Cyc/Inst.	Emulator Clock Cyc/Inst.	No. Times Executed	Total Machine Cycles	Total Emulator Clock Cyc
START	LXI	B,00	3	10	1	3	10
CNT	INR	B	1	5	256	256	1280
	JZ	QUIT	3	10	256	768	2560
	JMP	CNT	3	10	255	765	2550
QUIT	NOP		1	4	1	1	4
	HLT						
	END	SRART				1793	6404

Note: The NOP program instruction is the last RTPA breakpoint in this 8080A program.

The number of times each instruction is executed can be derived by mentally simulating program execution. The total machine and emulator clock cycles for each program instruction are then obtained by multiplying the number of machine and clock cycles per instruction by the number of times the instruction is executed. The overall total machine and emulator clock cycles for the model program is the sum of the individual instruction totals.

The program run time for the model program is dependent on the emulator clock frequency. If we executed the model program in the 8001/8002A, which has a 3 MHz clock frequency in emulation Mode 0, we should expect to measure a program run time as follows:

$$\begin{array}{rclcl}
 \text{Time for Each} & & \text{Total Number} & = & \text{Program Run} \\
 \text{Clock Cycle} & \times & \text{of Clock Cycles} & & \text{Time} \\
 .333 \mu\text{sec} & \times & 6404 \text{ cycles} & = & 2133 \mu\text{sec} \\
 (T = 1/\text{freq.}) & & & &
 \end{array}$$

The total machine cycles (1793) and the total emulator clock cycles (6404) from Table A-3, and the program run time (2133 usec) from the above formula, are referred to in the following paragraphs as the expected (calculated) values of the model program in emulation Mode 0.

When the 8001/8002A executes the model program in emulation Modes 1 and 2, the frequency of the prototype clock determines the program run time. The 8080A prototype used for these demonstrations has a clock frequency of .75 MHz and a 1.333 μsec time for each clock cycle. If we executed the model program in the 8001/8002A in emulation Modes 1 and 2, we should expect to measure a program run time as follows:

$$\begin{array}{rclcl}
 \text{Time for Each} & & \text{Total Number of} & = & \text{Program Run} \\
 \text{Clock Cycle} & \times & \text{Clock Cycles} & & \text{Time} \\
 1.333 \mu\text{sec} & \times & 6404 \text{ cycles} & = & 8537 \mu\text{sec}
 \end{array}$$

The total machine cycles (1793) and the total emulator clock cycles (6404) from Table A-3 have not changed. Only the program run time is changed (to 8537 μsec) due to the different clock frequency. These values are referred to in the following paragraphs as the expected (calculated) values of the model program in emulation Modes 1 and 2.

NOTE

These are expected (calculated) values. They do not account for the wait states that are added when the program is executed in 8001/8002A program memory for emulation Modes 0 and 1.

RTPA MEASUREMENTS

We will now load the model program into the 8001/8002A. We'll execute the program in each of the three emulation modes, and measure the program run time and emulator clock cycles for each mode. Remember our earlier discussions regarding the effects of adding wait states. (Refer back to the formulas A.1 and A.2.) Before the RTPA measured value can be compared to the expected (calculated) value, the effects of the wait states must be subtracted.

Table A-4 compares the expected, measured, and actual values for both the program run times and the emulator clock cycles. This table lists the values for each emulation mode. Each of these values is explained here:

- Expected (calculated). These values are obtained by calculations from the program. Table A-3 shows the calculations of the model program. These are the run times and clock cycles you expect the program to possess.
- Measured. These are the values that are measured with the RTPA commands.
- Actual (calculated). These values are obtained by subtracting the effects of the added wait states (Tables A-1 and A-2) from the RTPA measured values.

The expected and actual values for emulator clock cycles are usually close, within a cycle or two. Whereas, the expected and actual values for program run times may vary several microseconds. Note in Table A-4 that there is a 15-microsecond difference in Mode 1, and an 11-microsecond difference in Mode 2. These differences are due to the number of variables used in obtaining the expected values.

The following paragraphs explain the expected, measured, and actual values listed in Table A-4 for each emulation mode.

Table A-4
Comparison of Program Run Times and Emulator Clock Cycles

Mode of Operation		Expected (Calculated)	Measured With RTPA	Actual (Calculated)
Mode 0	Program Run Time	2133 μ sec	2728 μ sec	2131 μ sec
	Emulator Clock Cycles	6404 cycles	8197 cycles	6404 cycles
Mode 1 using 8001/8002A Program Memory	Program Run Time	8537 μ sec	10,912 μ sec	8522 μ sec
	Emulator Clock Cycles	6404 cycles	8196 cycles	6403 cycles
Mode 1 using the Prototype Memory	Program Run Time	8537 μ sec	8526 μ sec	8526 μ sec
	Emulator Clock Cycles	6404 cycles	6403 cycles	6403 cycles
Mode 2	Program Run Time	8537 μ sec	8526 μ sec	8526 μ sec
	Emulator Clock Cycles	6404 cycles	6403 cycles	6403 cycles

Note: The model program contained in Table A-3 was used to obtain the above values.

Measurements for Mode 0

To measure the program run time and emulator clock cycles, load the sample program from Table A-3 into 8001/8002A program memory. Set the RTPA commands to measure the program run time and the total emulator clock cycles. You will note there are considerable differences between the measured values and the expected values. These values are shown in Table A-4 for Mode 0 operation.

When operating in emulation Mode 0 or 1, the actual (calculated) program run time and emulator clock cycles can be calculated from the RTPA measured values if the following parameters are known:

- The number of wait states added for each machine cycle. (Refer to Tables A-1 or A-2 to obtain the number of wait states added for the associated emulator processor and mode of operation.)
- The number of machine cycles in the program. (Refer to Table A-3, and the accompanying discussion. The machine cycles can also be determined with the CNT C command, if the total does not exceed 65,534 cycles.)
- The clock frequency in the associated emulator processor. (See the separate description of each emulator processor's characteristics later in this appendix.)

Recall that earlier in this appendix, two formulas were listed that show the effects of adding wait states to each machine cycle. (Refer back to the formulas A.1 and A.2.) These formulas are used to find the actual run times and clock cycles of a given program. Apply these formulas to the RTPA measured values in Mode 0 to obtain the actual (calculated) values. These values are then compared to the expected (calculated) values in Table A-4.

$$\begin{array}{rcl}
 \text{Program Run Time as} & & \text{Time of the Added} & & \text{Actual (Calculated)} \\
 \text{Measured with the RTPA} & - & \text{Wait States} & = & \text{Program Run Time} \\
 \\
 2728 & - & 597 & = & 2131 \mu\text{sec} \\
 & & (1793 \times .333 \mu\text{sec}) & &
 \end{array}$$

$$\begin{array}{rcl}
 \text{Emulator Clock Cycle as} & & \text{Number of Wait} & & \text{Actual (Calculated)} \\
 \text{Measured with the RTPA} & - & \text{States Added} & = & \text{Emulator Clock Cycles} \\
 \\
 8197 & - & 1793 & = & 6404
 \end{array}$$

Measurements for Mode 1 Using 8001/002A Program Memory

In Mode 1, the prototype control probe must be connected between the prototype and the 8001/8002A system. In this mode, the clock is supplied by the prototype. With the model program executing in program memory, the RTPA measured values for program run time and emulator clock cycles are recorded in Table A-4.

The actual (calculated) values for program run time and emulator clock cycles are obtained by applying the same formulas used for Mode 0. These values are then compared to the expected (calculated) values in Table A-4.

Program Run Time as Measured with the RTPA	—	Time of the Added Wait States	=	Actual (Calculated) Program Run Time
10,912	—	2390 (1793 x 1.333 μ sec)	=	8522 μ sec
Emulator Clock Cycles as Measured with the RTPA	—	Number of Wait States Added	=	Actual (Calculated) Emulator Clock Cycles
8163	—	1793	=	6403 cycles

Measurements for Modes Using Prototype Memory

In emulation Mode 1 with the program mapped and moved to the prototype memory, and in emulation Mode 2, no wait states are added to the machine cycles, as noted in Tables A-1 and A-2 for the 8080A emulator processor. Therefore, the RTPA measured values are also the actual values, since wait states are not added.

Measurements for Other Emulator Processors

In the preceding calculations, the 8080A emulator processor is used to demonstrate the variances in the program run times and emulator clock cycles for the different emulation modes of operation. You can compare the expected, measured, and actual values of programs executed with the other emulator processors supporting the 8001/8002A systems. To do this, use a model program similar to the program in Table A-3, and perform the same calculations conducted in the preceding paragraphs for the 8080A emulator processor.

NOTE

When calculating the actual emulator clock cycles, the 9900 and Z80 emulator processors use a different formula. Refer to the individual characteristics of these emulator processors, listed in this appendix.

Measurements for Complex Programs

The actual program run time or number of emulator clock cycles for a complex program can be determined with the RTPA commands. The following paragraphs tell you how to make these calculations.

The RTPA general purpose counter can count to a maximum of 65,534. So long as the program run time or number of emulator clock cycles do not exceed 65,534 units of measure, the program timing and clock cycles can be measured as described in the previous paragraphs.

However, if the run time or clock cycles of a program exceeds the maximum count of the RTPA general purpose counter, the following steps should be taken:

- Divide the program into modules or subroutines that do not exceed 65,534 time units or emulator clock cycles. (Time may be measured in milliseconds or microseconds.)
- Then add the various times or cycles together to obtain the overall total.

Use Tables A-1 and A-2 to determine the number of wait states added for each machine cycle. You can then use the same formulas used in the previous paragraphs to determine the actual run time or actual clock cycles of the sample program.

NOTE

Remember that the number of machine cycles in a program can be determined by counting the number of bus cycles or bus transactions with the CNT C command.

INDIVIDUAL EMULATOR CHARACTERISTICS

The emulator characteristics associated with RTPA operation are provided in the following data sheets. Each emulator processor's characteristics are described on a separate data sheet. Additional data sheets will be added as new emulator processors support the 8001/8002A systems. At this printing, the following data sheets are included:

- 3870 Emulator Processor
- 6800 Emulator Processor
- 8080A Emulator Processor
- 8085A Emulator Processor
- 9900 Emulator Processor
- Z80 Emulator Processor

3870 EMULATOR PROCESSOR

This data sheet describes the characteristics of the 3870 Emulator Processor that are associated with RTPA operation.

Mode 0 System Clock

The frequency of the Mode 0 system clock, provided by the 8001/8002A. 4 MHz

Mode 1 and 2 Prototype Clock

The maximum allowable frequency for Mode 1 and Mode 2 clock, provided by the prototype. 4 MHz

Compatibility With RTPA Commands

All of the RTPA commands are compatible with the 3870 Emulator Processor.

Addition of Wait States

No wait states are added in any emulation mode. See Tables A-1 and A-2 for further details.

6800 EMULATOR PROCESSOR

This data sheet describes the characteristics of the 6800 Emulator Processor that are associated with RTPA operation.

Mode 0 System Clock

The frequency of the Mode 0 system clock, provided by the 8001/8002A. 1 MHz

Mode 1 and 2 Prototype Clock

The maximum allowable frequency for Mode 1 and Mode 2 clock, provided by the prototype. 1 MHz

Compatibility With RTPA Commands

All of the RTPA commands are compatible with the 6800 Emulator Processor.

Addition of Wait States

No wait states are added in any emulation mode. See Tables A-1 and A-2 for further details.

8080A EMULATOR PROCESSOR

This data sheet describes the characteristics of the 8080A Emulator Processor that are associated with RTPA operation.

Mode 0 System Clock

The frequency of the Mode 0 system clock, provided by the 8001/8002A. 3 MHz

Mode 1 and 2 Prototype Clock

The maximum allowable frequency for Mode 1 and Mode 2 clock, provided by the prototype. 2.08 MHz

Compatibility With RTPA Commands

All of the RTPA commands are compatible with the 8080A emulator processor.

Addition of Wait States

Wait states are added in Modes 0, 1, and 2, depending on the setting of Debug mode and DEBUG commands. See Tables A-1 and A-2 for further details.

8085A EMULATOR PROCESSOR

This data sheet describes the characteristics of the 8085A Emulator Processor that are associated with RTPA operation.

Mode 0 System Clock

The frequency of the Mode 0 system clock, provided by the 8001/8002A. 6.25 MHz

Mode 1 and 2 Prototype Clock

The maximum allowable frequency for Mode 1 and Mode 2 clock, provided by the prototype. 6.25 MHz

Compatibility With RTPA Commands

All of the RTPA commands are compatible with the 8085A Emulator Processor.

Addition of Wait States

Wait states are added in Modes 0, 1, and 2, depending on the setting of Debug mode and DEBUG commands. See Tables A-1 and A-2 for further details.

9900 EMULATOR PROCESSOR

This data sheet describes the characteristics of the 9900 Emulator Processor that are associated with RTPA operation.

Mode 0 System Clock

The frequency of the Mode 0 system clock, provided by the 8001/8002A. 3.0 MHz

Mode 1 and 2 Prototype Clock

The maximum allowable frequency for Mode 1 and Mode 2 clock, provided by the prototype. 3.3 MHz

Compatibility With RTPA Commands

On the 9900 Emulator Processor, I/O functions are performed within the microprocessor device. The 9900 Emulator Processor therefore does not use the EVT option B and RTT commands to specify the I/O characteristics of the various bus operations. The following parameters of the EVT option B and RTT commands do not yield meaningful results with the 9900 Emulator Processor.

- I I/O address only.
- IR I/O reads only.
- IW I/O writes only.

Do not use these command options when operating the RTPA with the 9900 Emulator Processor.

With the 9900, the measurement of total emulator clock cycles with the RTPA commands differs from the other emulator processors. See the following paragraphs on the addition of wait states.

Addition of Wait States

Wait states are added in Modes 0, 1, and 2, depending on the settings of the Debug mode and DEBUG commands. See Tables A-1 and A-2 for details on when the wait states are added. Note that the 9900 Emulator Processor adds one wait state in Mode 1, regardless of whether the program is executed in program memory or the prototype memory.

The formula used earlier in this appendix to calculate the actual emulator clock cycles must be modified when using the 9900 Emulator Processor. The 9900 emulator clock is divided by two in the 9900 Emulator Processor circuitry, before it is sent to the RTPA. For the 9900, the formula for finding the actual emulator clock cycles is modified, as follows:

$$\begin{array}{l} \text{Emulator Clock Cycles as} \\ \text{Measured with the RTPA} \end{array} \times 2 - \begin{array}{l} \text{Number of Wait} \\ \text{States Added} \end{array} = \begin{array}{l} \text{Actual (Calculated)} \\ \text{Emulator Clock Cycles} \end{array}$$

In this formula, the number of wait states is subtracted from the results of multiplying the measured emulator clock cycles by two. Remember that the number of wait states can be determined by counting the number of bus transactions with the CNT C command.

To find the actual execution time of a 9900 program, you can use the same formula (A.1) stated earlier in this appendix.

Z80 EMULATOR PROCESSOR

This data sheet describes the characteristics of the Z80 Emulator Processor that are associated with RTPA operation.

Mode 0 System Clock

The frequency of the Mode 0 system clock, provided by the 8001/8002A. 4 MHz

Mode 1 and 2 Prototype Clock

The maximum allowable frequency for Modes 1 and Mode 2 clock, provided by the prototype. 4 MHz

Compatibility With RTPA Commands

All the RTPA commands are compatible with the Z80 Emulator Processor with the exception of measuring the total emulator clock cycles. See the following paragraphs on the addition of wait states.

Addition of Wait States

Wait states are added in Modes 0, 1, and 2, depending on the settings of the Debug mode and DEBUG commands. See Tables A-1 and A-2 for details on when the wait states are added. The Z80 Emulator Processor adds two wait states in Modes 0 and 1, when the program is executed in program memory.

The formula used earlier in this appendix to calculate the actual emulator clock cycles must be modified when using the Z80 Emulator Processor. The Z80 emulator clock is divided by two in the Z80 Emulator Processor circuitry, before it is sent to the RTPA. For the Z80, the formula for finding the actual emulator clock cycles is modified, as follows:

$$\begin{array}{l} \text{Emulator Clock Cycles as} \\ \text{Measured with the RTPA} \end{array} \times 2 - \begin{array}{l} \text{Number of Wait} \\ \text{States Added} \end{array} = \begin{array}{l} \text{Actual (Calculated)} \\ \text{Emulator Clock Cycles} \end{array}$$

In this formula, the number of wait states is subtracted from the results of multiplying the measured emulator clock cycles by two. Remember that the number of wait states for the Z80 can be determined by counting the number of bus transactions with the CNT C command and multiplying by two.

To find the actual execution time of a Z80 program, you can use the same formula (A.1) stated earlier in this appendix. The time of the added wait states can be calculated as follows:

$$\begin{array}{r} \text{Number of} \\ \text{Machine Cycles} \end{array} \times 2 \times \begin{array}{r} \text{Time for} \\ \text{Each Cycle} \end{array} = \begin{array}{r} \text{Time of the} \\ \text{Added Wait States} \end{array}$$

$$\begin{array}{r} \text{Measured With} \\ \text{CNT C command} \end{array} \quad \frac{1}{\text{Clock Frequency}}$$

Appendix B COMMAND DICTIONARY

	Page
COMMAND INDEX	
Real-Time Prototype Analyzer Commands	
BIF—Sets, clears, or displays BIF command line break options	B-4
CNT—Sets or displays the general purpose count units	B-7
DRT—Displays the RTT buffer contents	B-10
EVT—Sets, clears, or displays EVT command line options	B-15
RTT—Sets or displays the selection of currently stored RTT buffer transactions	B-25
System Commands	
ABORT—Terminates user or system program execution (8002A only)	B-2
EMULATE—Selects the emulation mode	B-13
LOAD—Reads Assembler or Linker object code files into program memory (8002A only)	B-19
MAP—Displays or sets memory map assignments	B-20
Data Transfer Commands	
DUMP—Transfers memory contents to a file or device	B-11
MOVE—Copies data from one memory location to another	B-23
Debug Commands	
DEBUG—Loads the debug system into system memory (8002A only)	B-9
GO—Passes execution control to the emulator processor	B-18
TRACE—Allows program execution monitoring	B-26

Appendix B

COMMAND DICTIONARY

INTRODUCTION

This Command Dictionary alphabetically lists and describes all the 8001/8002A RTPA commands and those 8001/8002A commands associated with the RTPA operations. The Command Index on the preceding page lists all commands according to their function.

Each entry in this command dictionary contains a syntax block, parameter definitions, command requirements, and command line examples.

The command requirements for the RTPA commands contain a brief description of each command. Each RTPA command is described in detail in Section 2 of this manual. Refer to Section 3 of this manual for more detailed examples and usage of the RTPA commands.

For more information about the 8001 or 8002A system commands, refer to your 8001 or 8002A μ Processor Lab System User's Manual.

NOTE

The ABORT, DEBUG, and LOAD commands are used only with the 8002A μ Processor Lab.

<i>Syntax</i>	
<u>ABORT</u>	$\left. \begin{array}{l} \text{command file name} \\ \text{system command name} \\ * \\ / \end{array} \right\}$

8002A μ Processor Lab Only

Parameters

Function

command file name	Represents a user-created file that contains TEKDOS commands.
system program name	Represents a TEKDOS command name.
*	Designates all active programs, either user-created or TEKDOS.
/	Designates all active user-created programs.

Requirements

The ABORT command is used with the 8002A only.

This command terminates either user or system program execution. To terminate a particular system or user program, enter the ABORT command with the name of the system or user program to be terminated. To terminate all active programs, enter the ABORT command with the asterisk parameter. To terminate all active user programs, enter the ABORT command with the slash parameter. This also closes all assigned channels. After the specified program execution terminates, control returns to TEKDOS

The ABORT command is generally used after the system displays the TEKDOS double prompt sequence. This sequence appears after the ESC key has been pressed once or twice.

To suspend, rather than terminate, an active program, enter the SUSPEND command or use the ESC key.

Examples> **ABORT EDIT**

Terminates the TEKDOS Editor.

> **ABORT**

Generates error code 31 because a required parameter is missing.

> **A ***

Terminates all program execution.

> **A CMDFILE**

Terminates the user-created command file named CMDFILE.

> **A /**

Terminates your program. If you are executing one of your programs under debug system control, this command also terminates the debug system and returns control to TEKDOS. Closes all assigned channels.

> **A PROGNM;S**

Same effect as previous example.

Syntax

BIF
 or
BIF {mode}

C
S

 or
BIF

1
2

 {CLR}

Parameters

Function

mode	BIF modes establish relationships between the two EVT hardware-generated triggers and, under software control, break program execution. The BIF mode consists of one of the following:
1	EVT1 trigger breaks program execution when EVT1 options are satisfied.
2	EVT2 trigger breaks program execution when EVT2 options are satisfied.
ARM	EVT1 trigger arms EVT2 comparator. EVT2 trigger arms EVT1 comparator and breaks program execution.
LIM	When EVT1 and EVT2 options are satisfied simultaneously, EVT1 trigger breaks program execution.
IND	When either EVT1 or EVT2 options are satisfied, the corresponding EVT trigger breaks program execution.
FRZ	EVT1 trigger arms EVT2 comparator and freezes the RTT buffer. EVT2 trigger arms EVT1 comparator and breaks program execution.
C (continue)	Returns program control back to the program for continued program execution after a break in program execution.
S (step)	Returns program control back to the system terminal after a break in program execution.

- 1**
2 CLR Clears the specified EVT trigger from breaking program execution.
- CLR** Clears the previous assigned BIF or EVT mode. Defaults to the EVT IND mode.
- no parameter** Displays the currently assigned BIF command.

Requirements

The BIF modes establish relationships between the two EVT hardware-generated triggers. Under software control, the BIF mode breaks or stops the program execution, freezes the RTT buffer, and returns the program execution control back to the system terminal or back to the program for continued execution. Refer to Table B-1.

The BIF mode parameter overrides the previous set EVT or BIF mode except for BIF 1 or BIF 2. BIF CLR should be entered before you enter BIF1 or BIF2.

Table B-1
BIF Mode Parameters

BIF Mode	Interaction Between Command Lines	General Purpose Counter		RTT Buffer Frozen	Breakpoint Generated	Command Entry ^d	Display ^d
		Enabled	Disabled				
ARM	EVT1 arms EVT2. EVT 2 arms EVT1.	EVT1 Trigger	EVT2 Trigger	EVT2 Trigger	EVT2 Trigger	BIF ARM BIF ARM C	BIF-ARM S BIF-ARM C
IND	None: EVT1 and EVT2 are independent.	Start of program execution. ^a	EVT1 or EVT2 Trigger ^b	EVT1 or EVT2 Trigger ^b	EVT1 or EVT2 Trigger ^b	BIF IND BIF IND C	BIF-1 S 2 S BIF-1 C 2 C
LIM	Both EVT1 and EVT2 must be satisfied at the same time.	Start of program execution. ^a	EVT1 Trigger ^c	EVT1 Trigger ^c	EVT1 Trigger ^c	BIF LIM BIF LIM C	BIF-LIM S BIF-LIM C
FRZ	EVT1 arms EVT2. EVT2 arms EVT1.	EVT1 Trigger	EVT2 Trigger	EVT1 Trigger	EVT2 Trigger	BIF FRZ C BIF FRZ C	BIF-FRZ S BIF-FRZ C
1	None: EVT2 is ignored.	Start of program execution. ^a	EVT1 Trigger	EVT1 Trigger	EVT1 Trigger	BIF 1 BIF 1 C	BIF-1 S BIF-1 C
2	None: EVT1 is ignored.	Start of program execution. ^a	EVT2 Trigger	EVT2 Trigger	EVT2 Trigger	BIF 2 BIF 2 C	BIF-2 S BIF-2 C

^a The general purpose counter is reset and enabled at the start of program execution, when the go command is entered with an address. When the go command is entered without an address, the count on the counter is cumulative.

^b The general purpose counter is disabled, the RTT buffer is frozen, and a breakpoint is generated if either the EVT1 or EVT2 trigger is generated.

^c The general purpose counter is disabled, the RTT buffer is frozen, and a breakpoint is generated, if the EVT1 and EVT2 options are satisfied at the same time and an EVT1 trigger is generated.

^d The entry commands and displays are shown for both return-options S and C. Return-option S is the default value and is displayed if the return-option C is not specified.

Examples> **BIF CLR**

Clears the assigned BIF or EVT mode, and defaults to the EVT IND mode.

> **BIF**

Displays the currently assigned BIF mode.

> **BIF 2**

Generates a break if the EVT2 options are satisfied, and returns control to the system terminal.

> **BIF IND C**

Generates a break if either the EVT1 or EVT2 options are satisfied, and returns control back to the program for continued execution.

> **BIF LIM**

Generates a break if EVT1 comparison options and EVT2 address and bus operation options are satisfied simultaneously, and returns control to the system terminal.

*Syntax***CNT** [option]*Parameters**Function*

option	Sets the units of count for the general purpose counter and the units of delay for EVT option C (delay count). Consists of one of the following units:
C	Bus cycles. Counts each bus transaction.
E	Emulator clocks. Counts the emulator clock cycles.
F	Bus instruction fetches. Counts each bus fetch cycle.
M	Milliseconds. Counts the program running time in milliseconds. Default option.
T	RTT buffer stores. Counts the number of transactions stored in the RTT buffer.
U	Microseconds. Counts the program running time in microseconds.
1	EVT1 occurrences. Counts how many times the EVT1 options parameters are satisfied.
2	EVT2 occurrences. Counts how many times the EVT2 option parameters are satisfied.
no parameters	Displays the count value and units of count for the last executed program.

Requirements

The CNT command sets the units of count for the general purpose counter and the units of delay for the EVT option C (delay count). The general purpose counter is reset each time a program is executed with a GO command followed by an address. If the program is executed without an address, the count is accumulative.

Entering the CNT command with no parameters displays the count value and unit of count from the start of the last executed program or from EVT1 (if in the ARM or FRZ mode) to the end of program execution; at which time the RTT buffer is frozen (except FRZ mode) and a breakpoint is established.

The CNT option should be specified before executing a program. When a new CNT option is entered, the program must be executed again to obtain a valid count. Only one CNT option can be specified at a time.

Examples

> **CNT**

Displays the count value and count units of the general purpose counter.

> **CNT F**

Counts the number of bus instruction fetches.

> **CNT 1**

Counts the number of times the EVT1 comparison parameters are satisfied.

Syntax

```

DEBUG [device
         file name[/disc drive]]

```

8002A μ Processor Lab Only*Parameters**Function*

device	Names the output device that displays all output produced by the debug system commands. Defaults to the system terminal (CONO).
file name	Names the output file that contains all output produced by the debug system commands.
/disc drive	Identifies the drive containing the disc where the specified file resides. Defaults to the system disc.

Requirements

The DEBUG command is used with the 8002A only.

The DEBUG command loads the debug system from disc to system memory. This action invokes the debug system. After TEKDOS loads the debug system, the system terminal displays the TEKDOS prompt sequence and waits for you to enter debug or TEKDOS commands.

Before invoking the debug system, load your object module into program memory with the LOAD command. You may then enter the DEBUG command and invoke any debug system commands or TEKDOS commands that may execute while the debug system is active.

To initiate program execution while the debug system is active, enter the GO command. To suspend your program while it executes press the ESC key once. This action displays a trace line on the debug display device or file. The trace line indicates that a break occurred at the line displayed. The address of the last instruction executed appears below the line, followed by the word BREAK. To resume program execution from the instruction where the break occurred, enter the GO command without an address parameter.

Example

```
> DEB
```

Loads the debug system from disc into program memory. The debug display device defaults to the system terminal (CONO).

Syntax

DRT [* number of bus transactions]
--

*Parameters**Function*

*	Displays the entire contents of the RTT buffer (128 bus transactions).
number	Displays the last number of transactions in the RTT buffer.
no parameter	Displays the RTT buffer contents from the start of the last executed program.

Requirements

The DRT command displays the entire contents of the RTT buffer, or any specified portion thereof. The transactions are displayed sequentially, from the oldest to the most recent transaction stored in the RTT buffer. Blank lines separate the start of the most recent program execution from the previously executed program.

Examples

> **DRT**

Displays the RTT buffer contents from the start of the last executed program.

> **DRT 60**

Displays the last 60 bus transactions from the RTT buffer.

```

Syntax
  [W
  [X
  [Y
  [Z] {lower address} [upper address] [device
  [file name[/disc drive]]
    
```

Parameters

Function

W, X, Y, or Z	Names the index that was assigned an offset amount with the BIAS command. All addresses are offset by the amount assigned to the specified index. Defaults to zero bias.
lower address	Specifies the program or user prototype memory address where the data display starts.
upper address	Represents the program or user prototype memory address where the data display stops.
device	Names the output device that receives the data display. Defaults to the system terminal (CONO).
file name	Identifies the output file that receives the data display.
/disc drive	Selects the disc drive that contains the disc where the data display is written.

Requirements

This command displays 16-byte data blocks from program or user prototype memory on the system terminal. The display starts with the first multiple of 16 below the lower address parameter. The memory map assignments determine whether program or user prototype memory data is displayed. To display and alter memory map assignments, enter the MAP command.

If you specify an index parameter with this command, delimiters between the index and the specified lower address are not allowed.

When you enter both the lower and upper address parameters, the system terminal displays all data blocks in the address range. The data display ends with the first multiple of 16 above the upper address parameter. The 8001/8002A replaces the least significant digit of the lower address parameter address with 0, and the least significant digit of the upper address parameter with F. If you enter only the lower address parameter, the display device receives only one 16-byte data block.

Each 16-byte data block displayed follows the address and an equals sign. The display line contains a space between each data byte or word. The DUMP command outputs the ASCII character representation of the data bytes on the right side of the display device. Periods represent data bytes with hexadecimal values less than 20 and greater than 60. A space represents the data byte value 20.

Examples

> **DUMP 107**

Displays the data between program memory addresses 100 and 10F.

> **DUMP 11D 202**

Displays the data between program memory addresses 110 and 20F.

*Syntax***EMULATE** {emulation mode}*Parameters**Function***emulation mode**

Sets the emulation mode for the emulator processor. The possible values for this parameter are 0, 1, or 2.

0

System mode. Execution in this mode uses program memory and the 8001/8002A I/O and clock.

1

Partial emulation mode. Execution in mode 1 uses program memory and/or user prototype memory, according to the map assignments. Refer to the MAP command description appearing later in this section. Execution also uses prototype I/O and clock.

2

Full emulation mode. Execution in mode 2 uses prototype memory, I/O and clock.

Requirements

You must select the emulation mode before initiating program execution. When executing programs in emulation modes 1 and 2, your prototype must be connected to the 8001/8002A system via the prototype control probe.

If you wish to map portions of your program to both program memory and user prototype memory with the MAP command, first enter the EMULATE 1 command line.

CAUTION

Use caution when performing memory write operations in program memory when executing programs in emulation mode 1. Memory write operations performed in program memory are also performed in user prototype memory.

Examples

> **EMULATE 0**

Selects emulation mode 0.

> **EMULATE 1**

Selects emulation mode 1.

> **EM 2**

Selects emulation mode 2.

```

Syntax
EVT
or
EVT {mode}
or
EVT [ 1 ] {CLR}
or
EVT [ 2 ] [CLR] {option} {operator} {value} [ {option} {operator} {value} ] ...
    
```

Parameters

Function

mode	The EVT modes establish relationships between the two EVT hardware-generated triggers. The EVT mode consists of the following:
ARM	EVT1 trigger arms EVT2 comparator. EVT2 trigger arms EVT1 comparator.
LIM	When EVT1 and EVT2 options are satisfied, an EVT1 trigger is generated.
IND	Both EVT1 and EVT2 triggers are independent of each other. EVT default mode.
[1] CLR [2] CLR	Clears the specified EVT command line options, except option B.
CLR	Clears all EVT1 and EVT2 command lines options, except option B. Clears the current assigned EVT or BIF mode and defaults to the EVT IND mode.
no parameters	Displays the current status of both EVT1 and EVT2 in separate display lines. Displays the assigned EVT or BIF mode in the third line.
options	Each option parameter consists of an option identifier, an operator, and an option value. Table B-2 lists the EVT options and their associated operator signs, values, and functions.

Table B-2
EVT Option Parameters

Option Identifier	Operators	Option Value	Function
A	= ; < >	0000 to FFFF	Bus address (hexadecimal).
D	= ; < >	0000 to FFFF	Bus data (hexadecimal).
T	=	00000000 to 11111111	Test clip bits (binary). "X" may be used to indicate "dont care."
B	=	ALL F I IR IW M MR MW R W	Bus options All bus activity. Default value. Instruction fetches only. I/O address only. I/O reads only. I/O writes only. Memory accesses only Memory reads only. Memory writes only. Read operations only. Write operations only.
P	=	0 to 65535	Pass count (decimal).
C	=	0 to 65535	Counter delay units (decimal), as specified by the CNT command.

Operator Definition

- = 16-bit equality > 16-bit greater than or equal
- 8-bit equality < 16-bit less than or equal

Requirements

The EVT command sets, clears, or displays the current EVT1 and EVT2 command lines. EVT option parameters set the event comparators and the pass and delay counters. When the comparators and counters are satisfied, hardware triggers generate a pulse for triggering external equipment, such as an oscilloscope or logic analyzer.

The EVT modes establish a relationship between the two hardware-generated triggers. The EVT modes do not break program execution or freeze the RTT buffer. Refer to Table B-3.

Table B-3
EVT Mode Parameters

EVT Mode	Interaction Between Command Lines	General Purpose Counter		RTT Buffer Frozen ^c	Breakpoint Generated ^c	Command Entry	Display
		Enabled	Disabled				
ARM	EVT1 arms EVT2. EVT2 arms EVT1.	EVT1 Trigger	EVT2 Trigger	No	No	EV ARM	ARM
IND	None	Start of program execution. ^a	End of program execution. ^b	No	No	EV IND	IND
LIM	Both EVT1 and EVT2 must be satisfied at the same time.	Start of program execution. ^a	End of program execution. ^b	No	No	EV LIM	LIM

^a The counter is reset and enabled at the start of program execution, when the GO command is entered with an address. When the GO command is entered without an address, the count in the counter is cumulative.

^b The counter is disabled at the end of program execution. There is no breakpoint set with the EVT mode commands; therefore, the end of program execution occurs under one of the following conditions:

- the ESC key is pressed.
- a software breakpoint is set with the 8001/8002A system command—BKPT
- a HALT type command is executed.

^cThe RTT buffer is not frozen and a breakpoint is not generated with the EVT mode commands.

Examples

> EV 1 CLR

Clears only the EVT1 command line options, except option B.

> EV 1 A=37D

Sets the EVT1 comparator for program address 037D.

> EV CLR

Clears the EVT1 and EVT2 command line options, except option B. Also clears the assigned EVT or BIF mode, and defaults to the EVT IND mode.

> EV ARM

Sets the EVT mode to ARM.

*Syntax***GO** [address]*Parameters***address***Function*

Indicates the program memory address where execution begins.

Requirements

This command passes execution control to the emulator processor. The first time you enter the GO command without an address parameter, execution begins at address 0. Thereafter, if you enter the GO command without an address parameter, execution begins at the address following the last instruction executed. When you enter the GO command with an address parameter, execution begins at the specified address in program memory.

If you specify a transfer address with the Assembler directive END, the LOAD command passes this transfer address to the GO command. Execution then begins at the transfer address when you specify the GO command without an address parameter. This occurs whether or not you invoke the GO command within the debug system.

To suspend execution, press the ESC key. To interrupt program execution when your program reads from or writes to specified addresses, set breakpoints with the BKPT command.

Examples

> **GO 0**

Begins program execution at address 0.

> **GO**

Resumes execution at the address following the last instruction executed. If your program is executing for the first time, execution proceeds from address 0, or from the transfer address passed by the LOAD command.

Syntax

```
LOAD [/offset amount] {file name[/disc drive]} [file name[/disc drive]] ...
```

8002A μ Processor Lab Only*Parameters**Function*

/offset amount	Alters the absolute load address by the specified hexadecimal amount for each data block transferred. Defaults to 0.
file name	Specifies the name of the assembled or linked object file to be loaded into program memory.
/disc drive	Identifies the number of the disc drive containing the disc where the file to be loaded resides. Defaults to 0.

Requirements

This command reads the specified hexadecimal object file into program memory. The file must have been previously created by the Assembler or Linker.

The file is loaded into program memory, starting at the location specified in the source code. If you specify an offset amount, the file starts at the location that equals the offset amount plus the location specified in the source code.

Object code loaded into program memory with the LOAD command line can be executed with the GO command. You can also trace program execution by loading an object file into program memory with the LOAD command; entering the DEBUG command, the appropriate TRACE command, and the GO command. Always enter the LOAD command before entering the debug system.

Examples

```
> LOAD PROOBJ;0/1
```

Loads the assembled object file named PROOBJ;0, residing on the disc in drive 1, into program memory, starting at the location specified in the source code.

```
> LO/100 PROOBJ;0
```

Loads PROOBJ;0 into program memory, starting at the address equal to the starting location specified in the source code plus 100. PROOBJ must reside on the disc in drive 0.

Syntax

```

MAP      [ M ]
or
MAP      { U }
         { P } { {lower address [-upper address]} } [ {lower address} [-upper address] ] ...
    
```

Display Memory Map Assignments

<i>Parameters</i>	<i>Function</i>
R	Displays the current memory map assignments in tabular form. Default value.
M	Displays the memory map assignments in matrix form.

Setting Memory Map Assignments

<i>Parameters</i>	<i>Function</i>
P	Assigns the memory contents, located at the specified address or address range, to program memory.
U	Assigns the memory contents, located at the specified address or address range, to user prototype memory.
lower address	Marks the lower address of the range assigned to program or user prototype memory.
upper address	Marks the upper address of the range assigned to program or user prototype memory. The hyphen must be entered as part of this parameter.

Requirements

Displaying Memory Map Assignments:

The R parameter outputs a tabular display that includes addresses or address ranges. Each address or address range is followed by either the letter U, indicating address assignments to user prototype memory, or the letter P, indicating address assignment to program memory.

The M parameter outputs a matrix display, with each element representing 128 bytes of user prototype or program memory. An asterisk represents 128 bytes of user prototype memory assignment; a hyphen represents 128 bytes of program memory assignment.

Each double column in the matrix has a number beside and above it to respectively identify its line and column. Each double column consists of two 128-byte blocks, comprising a total of 256 bytes. The address range of the first two blocks is 0000 through 00FF. The range of the second two blocks is 0100 through 01FF. One line of the matrix represents 4K bytes of memory. The address range of the first 16K byte memory module in the matrix is 0000 through 3FFF. The second 16K-byte memory module ranges from 4000 to 7FFF, the third from 8000 to BFFF, and the fourth from C000 to FFFF. The entire matrix consists of 64K bytes of memory.

The system ignores parameters following M or R.

Setting Memory Map Assignments:

You may set memory map assignments to program memory (P) and to user prototype memory (U) in 128-byte increments. You may specify either a single address or an address range. When you enter a single address, 128 bytes are mapped according to the specified U or P parameter. When you specify an address range, you must not enter a space between the addresses and the hyphen that delimits them. You may include as many addresses or address ranges as will fit on one line. These addresses or ranges need not appear in ascending numeric order. The second address in a range, however, must occur within the same 256-byte block as the first, or else be greater than the first address. For example, 9402-94FF is valid, but 1000-01FF produces error code 30, because 01FF does not share the same block as 1000.

Place the system in emulation mode 1 before setting memory map assignments. Otherwise, the system terminal will display the message, WARNING EMULATOR NOT IN MODE 1. Enter emulation mode 1 by typing the EMULATE 1 command line.

CAUTION

Use caution when performing memory write operations in program memory when executing programs in emulation mode 1. Memory store operations performed in program memory are also performed in user prototype memory.

Examples

```
> MAP P 0-3DFF 8700-FBFF
```

Assigns memory within the address ranges 0000 through 3DFF and 8700 through FBFF to program memory.

```
> MA U 3E00-86FF FC00-FFFF
```

Assigns memory within address ranges 3E00 through 86FF and FC00 through FFFF to user prototype memory.

> **MAP R**

0000-3DFF=P 3E00-86FF=U

8700-FBFF=P FC00-FFFF=U

Displays memory map assignments in tabular form. The tabular response above indicates that memory locations 0000 through 3DFF and 8700 through FBFF are assigned to program memory. Memory locations 3E00 through 86FF and FC00 through FFFF are assigned to user prototype memory.

> **MA**

Displays memory map assignments in tabular form.

Syntax

<u>MOVE</u>	$\left. \begin{array}{c} \text{PP} \\ \text{PU} \\ \text{UP} \\ \text{UU} \end{array} \right\}$	$\{\text{lower address}\} \{\text{upper address}\} \{\text{starting address}\}$
-------------	---	---

*Parameters**Function*

PP PU UP UU	<p>The first parameter consists of two letters. The first letter identifies the data source; the second letter identifies the data destination. The letter P designates program memory; U designates user prototype memory.</p>
lower address	<p>Marks the lower limit of the data block copied.</p>
upper address	<p>Marks the upper limit of the data block copied.</p>
starting address	<p>Marks the memory location where the data block starts to move.</p>

Requirements

This command copies the specified data block from either program or user prototype memory to a new location in either program or user prototype memory.

You may not enter a delimiter between the two letters (U or P) of the first parameter.

The specified upper address must be greater than or equal to the specified lower address.

You may only copy data to existing RAM with the MOVE command. When you attempt to copy data from program memory to user prototype memory, error code 59 may occur. This error code indicates that you attempted to write to unavailable or non-existing memory. If error code 59 occurs when you attempt to copy data to existing RAM in user prototype memory, a damaged prototype control probe may be causing the trouble.

The MOVE command moves the data between user prototype memory and program memory regardless of the assigned emulation mode.

Examples

> **MOVE PU 300 35E 300**

Copies the program memory data block ranging from address 300 through address 35E to user prototype memory, starting at address 300.

> **MOV UU 300 35E 1000**

Copies the user prototype memory data block ranging from address 300 through address 35E to user prototype memory, starting at address 1000.

*Syntax***RTT** [option]*Parameters**Function***option**

Selects the type of bus transactions that are stored in the RTT buffer. The RTT option consists of one of the following:

ALL	All bus transactions. Default value.
F	Instruction fetches only.
I	I/O accesses only.
IR	I/O reads only.
IW	I/O writes only.
M	Memory accesses only.
MR	Memory reads only.
MW	Memory writes only.
R	Read operations only.
W	Write operations only.

Requirements

The RTT command provides selective bus transaction storage. Transactions are stored in the RTT buffer sequentially from the oldest to the most recent transaction. The last instruction stored in RTT buffer when a breakpoint is reached is the break line (except in the BIF FRZ mode, when the buffer is frozen on the EVT1 trigger).

Examples

> **RTT**

Displays the RTT option in effect.

> **RTT MW**

Selects only memory-write bus transactions for storage in the RTT buffer.


```
Syntax
TRACE
or
TRACE { ALL } [STEP] [[lower address] [upper address]]
      { JMP }
      { OFF }
or
TRACE { LONG }
      { SHORT }
```

Parameters

Function

ALL	Displays a trace line for every instruction, after the emulator processor executes that instruction.
JMP	Displays a trace line for every instruction that changes program execution flow. Includes unconditional jumps, branches, and subroutine calls. Conditional branches, where the specified condition is unsatisfied, do not cause trace line display.
OFF	Disables trace line displays.
STEP	Suspends instruction execution and trace line display after each trace line appears.
lower address	Marks the lower bound of the instruction block affected by the current TRACE command. Must be less than or equal to upper address. Defaults to 0.
upper address	Marks the upper bound of the instruction block affected by the current TRACE command. Defaults to FFFF. Must be greater than or equal to lower address.
LONG	Displays trace lines in their standard form.
SHORT	Displays trace lines for certain microprocessors in an abbreviated form. Default value.

Requirements

The TRACE command allows program execution monitoring. Trace lines are displayed for instructions executed by the emulator processor. This monitoring process is called tracing. All trace lines appear on the DEBUG display device or file. Refer to the DEBUG command discussion for information on assigning a device or file for display.

Each trace line contains the memory address of the last executed instruction, the instruction executed, and the emulator processor register contents. The trace line display format varies for each microprocessor type. For more information, refer to your 8001 or 8002A System User's Manual: Emulator Specifics.

Once you have entered the desired TRACE commands, enter the GO command to begin program execution. During program execution, the trace lines are sent to the DEBUG display device or file. Pressing the ESC key during trace line display suspends program execution and the display. Entering the GO command after suspending execution in this manner resumes execution and trace line display at the point of interruption.

Examples

> **TRACE ALL,,500 2000**

Specifies continuous trace line display of all instructions within the address range 500 through 2000.

> **TRACE ALL,,,2000**

Specifies continuous trace line display of all instructions within the range 0 to 2000. The lower address defaults to 0.

> **TRACE JMP STEP**

Specified trace line display of all instructions that alter execution flow within the default address range 0 through FFFF. Suspends instruction execution and trace line display after each "jump" instruction executes. Clears all other active TRACE commands. To display the trace next line, enter the GO command.

Appendix C INSTALLATION

	Page
Installing the Real-Time Prototype Analyzer	C-1
Installing the Data Acquisition Interface	C-1
Installing the Real-Time Trace Module.	C-2
Grounding	C-5
Additional Grounding Precautions	C-5
Jumpers/Switches	C-5
Packaging for Shipment	C-8

ILLUSTRATIONS

Fig. No.		
C-1	Data Acquisition Interface installation.	C-2
C-2	8001/8002A module assignment; the Real-Time Trace module is in position 15	C-3
C-3	Typical cabling drawing of the 8001/8002A system installation	C-4
C-4	Typical 8001/8002A system grounding	C-6
C-5	Data Acquisition Interface Board	C-7

Appendix C

INSTALLATION

INSTALLING THE REAL-TIME PROTOTYPE ANALYZER

The installation procedure for the Real-Time Prototype Analyzer is divided into two parts: installing the Data Acquisition Interface, and installing the Real-Time Trace module.

Installing the Data Acquisition Interface

The Data Acquisition Interface unit resides inside the 8001/8002A system. The Data Acquisition Interface panel mounts flush with the rear panel of the Lab, as shown in Fig. C-1. Use the following procedure to attach the Data Acquisition Interface to the 8001/8002A.

1. Make sure that the main power to the 8001/8002A is OFF.
2. Remove the three screws along each side of the 8001/8002A top cover.
3. Lift the cover of the 8001/8002A straight up. Remove the cover, then set it aside.
4. As you face the 8001/8002A rear panel, notice the four screws holding a blank plate to the upper right hand corner of the rear panel. Remove the screws and the plate, and set them aside.
5. Place the Data Acquisition Interface unit inside the 8001/8002A chassis, with the component side up and the panel facing into the newly-created opening.
6. Align the four screw holes in the Data Acquisition Interface panel with the four screw holes in the 8001/8002A rear panel.
7. Thread the four screws back into the rear panel of the 8001/8002A, be sure to tighten them securely.

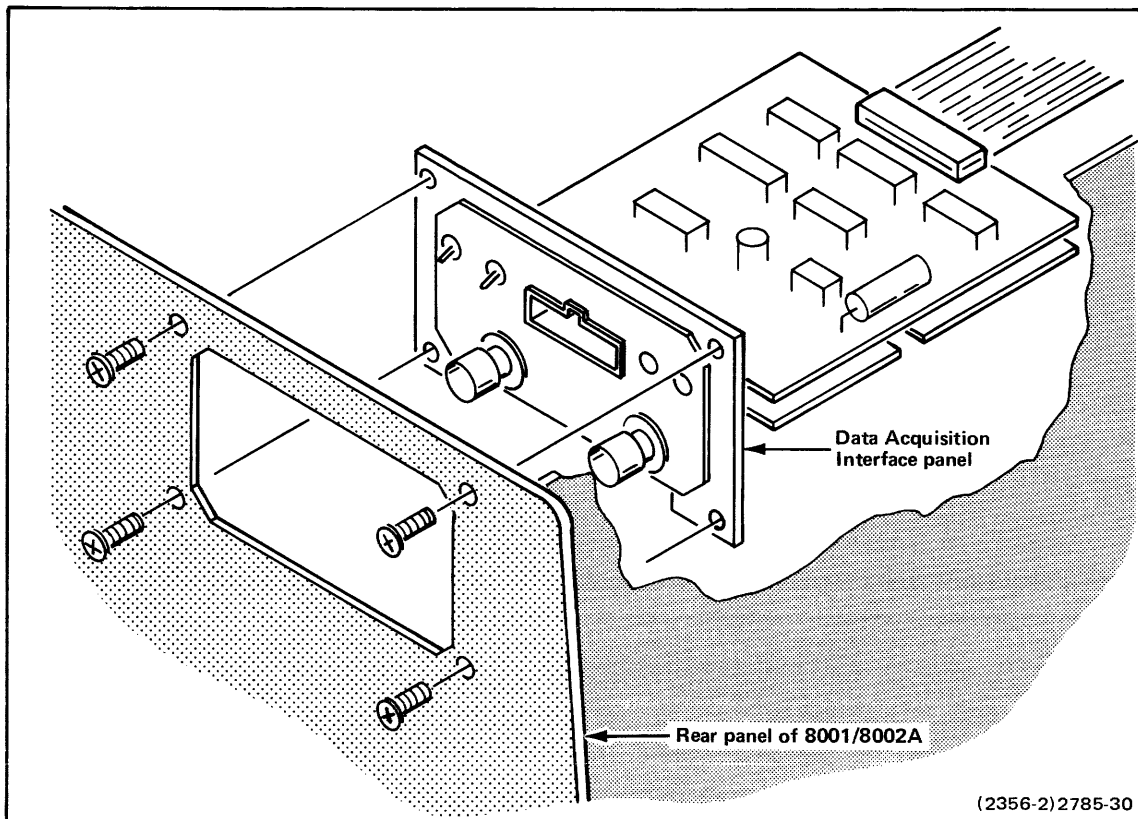


Fig. C-1. Data Acquisition Interface Installation.

Installing the Real-Time Trace Module

1. While facing the front of the 8001/8002A, grasp the Real-Time Trace module by its upper edges. Align the module with other 8001/8002A modules, so that its component side faces left.
2. Slide the module down the vertical guide channels labeled J15. Refer to Fig. C-2.
3. Lower the Real-Time Trace module until it reaches the motherboard connector; then, press down firmly and evenly on the top edge of the module until it snaps into place.
4. Attach the ribbon cable from the Data Acquisition Interface to the Real-Time Trace module edge connector P2. Make sure that pin 1 of the ribbon cable (red stripe) aligns with pin 1 of the edge connector. Refer to Fig. C-3 for an overall 8001/8002A cabling drawing.
5. Place the top cover back onto the 8001/8002A. Thread the six screws (previously removed) back into the sides of the cover, and tighten them securely.

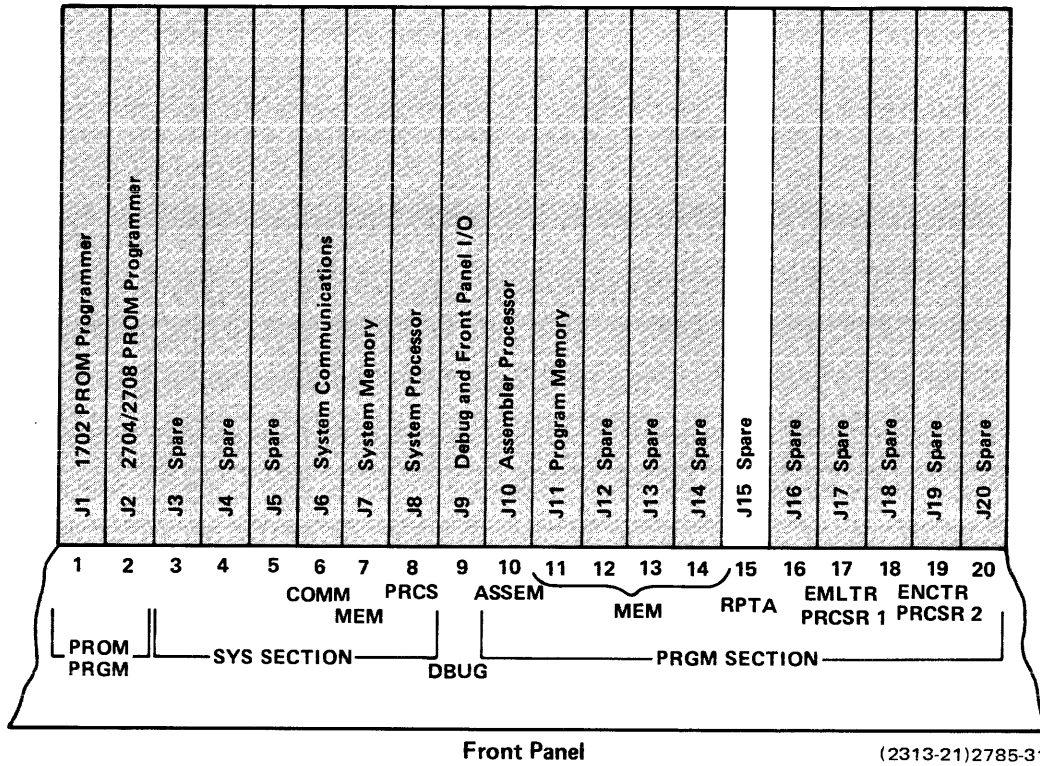
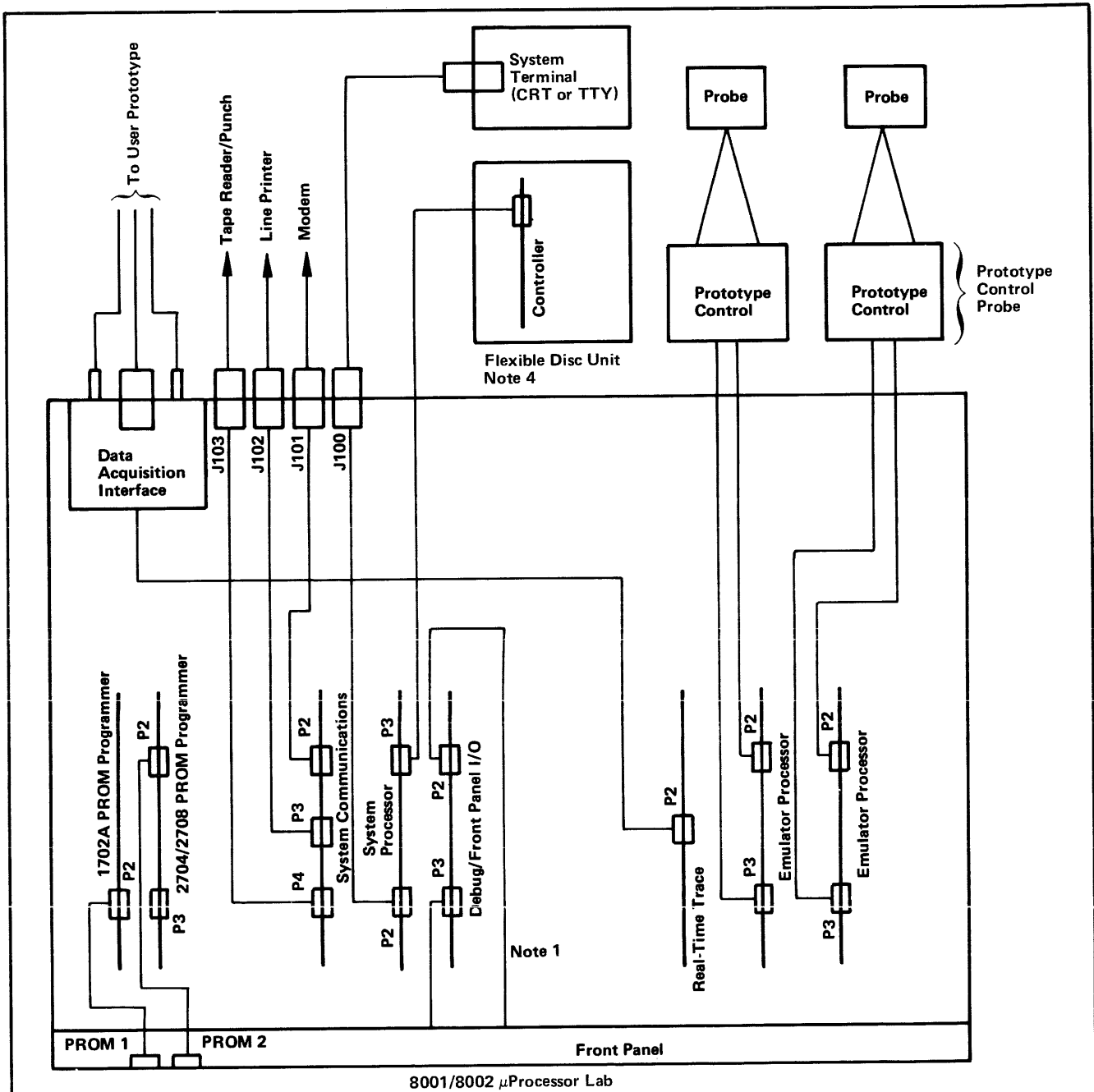


Fig. C-2. 8001/8002A module assignment; the Real-Time Trace Module is in position 15.



NOTES:

1. P2 and cable are connected only when the Maintenance Front Panel is installed on the 8001/8002A.
2. All units shown (except Prototype Control Probe) are connected to primary power source (115 or 230 Vac). Power cables not shown.
3. PROM 1 may be connected to either 1702A or 2704/2708 PROM Programmers. PROM 2 is connected only to 2704/2708 PROM Programmer.
4. Flexible Disc Unit is installed with the 8002A only.

(2312-2)2785-32

Fig. C-3. Typical cabling drawing of the 8001/8002A System Installation.

GROUNDING

A proper ground system for the 8001/8002A is very important; this also includes any optional and peripheral equipment. The following grounding procedures are recommended to provide and maintain a proper ground system. Refer to the 8001/8002A μ Processor Lab Installation Guide or 8001/8002A μ Processor Lab Service Manual for additional grounding information.

1. Make sure the primary power cords of all units (including the user's prototype equipment) are connected to outlets that are on the same ground system.
2. When the 8002A (including the user's prototype equipment) is operating in a high static environment, use a grounding strap that is as short as possible between the 8002A and Flexible Disc Unit. Use copper braid strap or other low-impedance wire, with soldered terminal lugs, for the grounding straps. (The Flexible Disc Unit is installed with the 8002A only.)
3. Attach all grounding strap lugs to the chassis of any unit being grounded. Ensure that the lugs make good contact with the bare metal by removing any paint or protective coating from the metal before attaching the grounding lug.

Additional Grounding Precautions

When the 8001/8002A is being used in conjunction with test equipment (such as an oscilloscope, or a logic analyzer), all units must be properly grounded to eliminate ground loops and reduce susceptibility to static discharge. Fig. C-4 shows a typical interconnection for an 8001/8002A system; a prototype control probe, a P6451 Logic Probe and an oscilloscope probe are connected to the user's prototype system. The grounding strap between the 8001/8002A chassis and the test equipment chassis prevents static discharge. The test equipment, when connected to the trigger from the Data Acquisition Interface unit, establishes a second tie point between the logic ground and chassis ground of the 8001/8002A. To prevent ground loops when test equipment is connected to the 8001/8002A remove the strap between TB2-2 and TB2-3 on the 8001/8002A back panel. This establishes the common tie point between logic ground and chassis ground at the test equipment, not at both units. Any unused leads of the P6451 Logic Probe should be grounded; if the probe is not in use, disconnect it from the Data Acquisition Interface unit.

JUMPERS/SWITCHES

There are no jumpers or switches on the Real-Time Trace module.

There is one jumper on the Data Acquisition Interface circuit board; it is used to select the clock threshold. See Fig. C-5 for the jumper location. In the 0—3 position, the clock threshold is the same as that of data channels 0—3. In the 4—7 position, the clock threshold is the same as that of data channels 4—7.

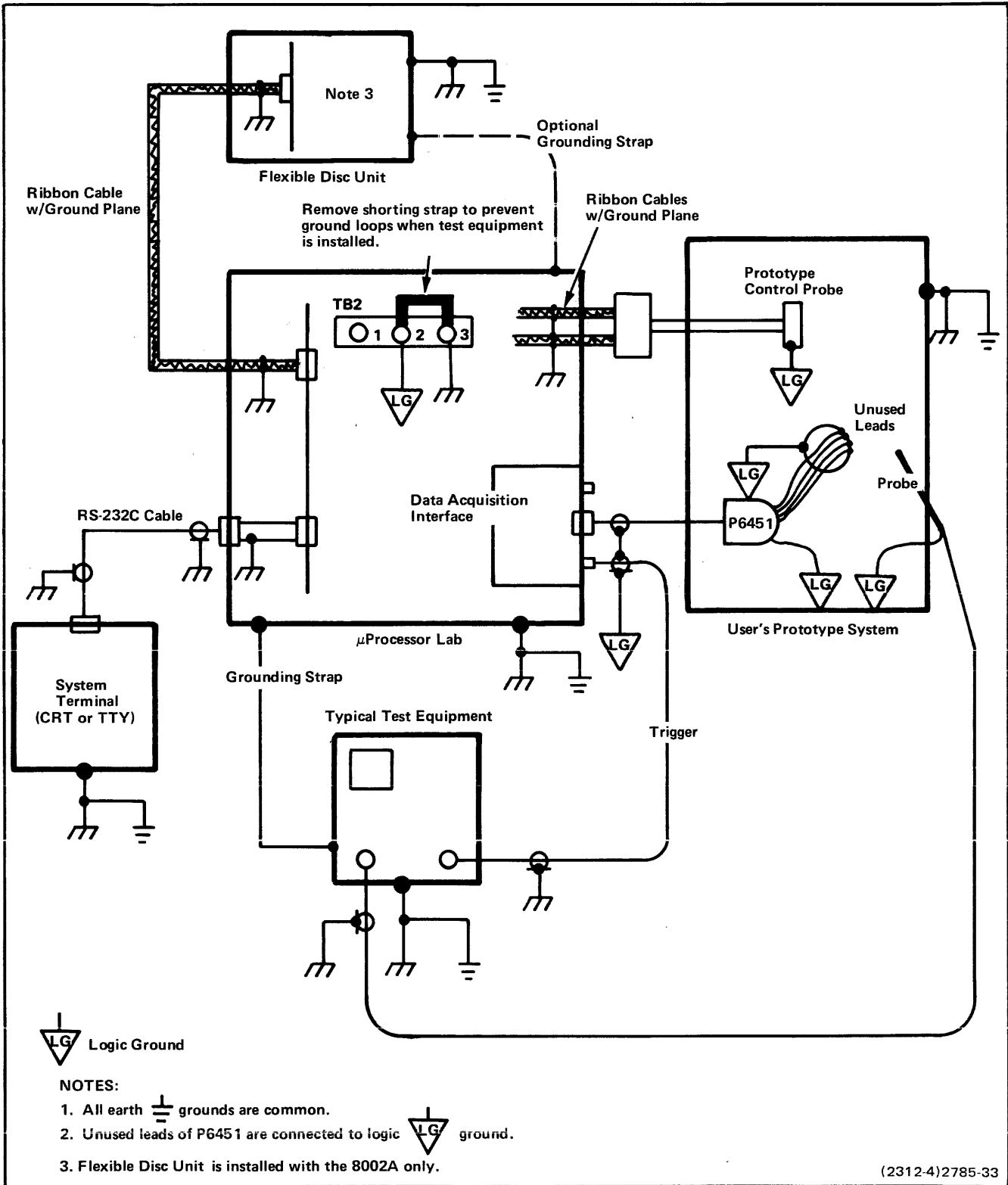


Fig. C-4. Typical 8001/8002A system grounding.

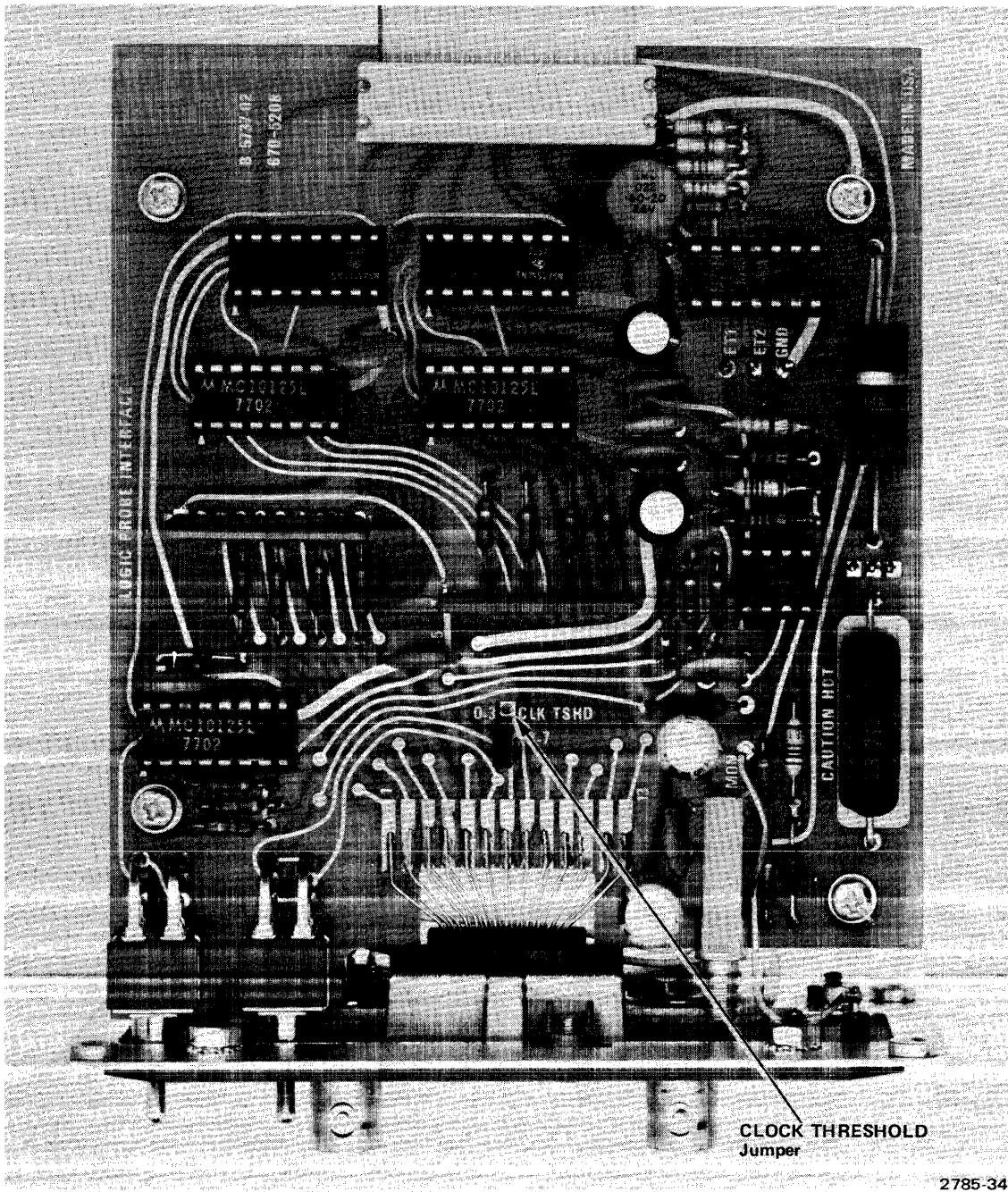


Fig. C-5. Data Acquisition Interface Board.

PACKAGING FOR SHIPMENT

If this RTPA option is to be shipped long distances by commercial transportation, it should be packaged as shipped from the factory for maximum protection. The carton and packing material in which your equipment was shipped should be saved for this purpose.

If the original packaging is unfit for use or not available, package the RTPA option as follows:

1. Obtain a carton of corrugated cardboard with at least a 200 lb. test strength and inside dimensions at least six inches greater than the instrument dimensions. The larger dimensions will allow for cushioning.
2. Surround the instrument with anti-static polyethylene sheeting for protection.
3. Cushion the instrument on all sides by tightly packing dunnage or urethane foam between the carton and the instrument. Allowing three inches on all sides.
4. Seal the carton with shipping tape or an industrial stapler.

Also, if this equipment is to be shipped to a Tektronix Service Center for service or repair, attach a tag to the equipment showing the following: the owner of the equipment (with address), the name of an individual at your firm that can be contacted, complete equipment type and serial number, and a description of the service required. Mark the address of the Tektronix Service Center, and your return address, on the carton in one or more prominent locations.

Appendix D

SAMPLE PROGRAM

```

00001      ;*****
00002      ;*
00003      ;*
00004      ;*          SAMPLE PROGRAM FOR 8080A EMULATOR          ;*
00005      ;*          USED FOR RTPA DEMONSTRATIONS                ;*
00006      ;*
00007      ;* THIS PROGRAM STARTS AT 0200H; IT LOADS MEMORY LOCATIONS ;*
00008      ;* 0250H TO 0275H WITH ZEROS, THEN JUMPS TO 0375H AND   ;*
00009      ;* LOADS MEMORY LOCATIONS 0375H TO 0350H WITH ZEROS; IT ;*
00010      ;* THEN JUMPS TO 0300H AND STOPS AT 0301H.              ;*
00011      ;*
00012      ;*****
00013
00014      0200  >          ORG          0200H
00015      0200  F3          START      DI
00016      0201  117502      LXI        D,0275H ;LOAD MEMORY CHECK
00017      0204  215002      LXI        H,0250H ;GET START OF MEMORY
00018      0207  3600        LOOP1     MVI        M,0          ;CLEAR MEMORY LOC 0250H
00019      0209  7D          MOV        A,L          ;CHECK LOW BYTE
00020      020A  BB          CMP        E            ;IS IT 75?
00021      020B  C21102 >          JNZ        CONT1      ;NO? - CONTINUE
00022      020E  C37503 >          JMP        NEXT      ;IT IS? JUMP TO MEM LOC 0375H
00023      0211  23          CONT1     INX        H            ;INR MEMORY REGISTER
00024      0212  C30702 >          JMP        LOOP1     ;KEEP ZEROING
00025
00026      0375  >          ORG          0375H
00027      0375  012503      NEXT      LXI        B,0325H ;LOAD MEMORY CHECK
00028      0378  215003      LXI        H,0350H ;GET START OF MEMORY
00029      037B  3600        LOOP2     MVI        M,0          ;CLEAR MEMORY LOC 0350H
00030      037D  7D          MOV        A,L          ;CHECK LOW BYTE
00031      037E  B9          CMP        C            ;IS IT 25?
00032      037F  C28503 >          JNZ        CONT2      ;NO? - CONTINUE
00033      0382  C30003 >          JMP        STOP      ;IT IS? JUMP TO MEM LOC 0300H
00034      0385  2B          CONT2     DCX        H            ;DCR MEMORY REGISTER
00035      0386  C37B03 >          JMP        LOOP2     ;KEEP ZEROING
00036
00037      0300  >          ORG          0300H
00038      0300  00          STOP      NOP                ;LAST RTPA BKPT BEFORE HALT
00039      0301  76          HLT                ;SUSPENDS EXEC AWAITING INT
00040      0200  >          END          START
    
```

Fig. D-1. List file of sample 8080 program.